Taking advantage of the software product line paradigm to generate customized user interfaces for decision-making processes: a case study on university employability

Andrea Vázquez-Ingelmo [Corresp., 1] , **Francisco J García-Peñalvo** [1] , **Roberto Therón Sánchez** [1, 2]

[1] GRIAL Research Group, Department of Computer Science and Automatics, University of Salamanca, Salamanca, Spain

[2] VisUSAL Research Group, Department of Computer Science and Automatics, University of Salamanca, Salamanca, Spain

Corresponding Author: Andrea Vázquez-Ingelmo
Email address: andreavazquez@usal.es

University employment and, specifically, employability has gained relevance since research in these fields can lead to improvement in the quality of life of individual citizens. However, empirical research is still insufficient to make significant decisions, and relying on powerful tools to explore data and reach insights on these fields is paramount. Information dashboards play a key role in analyzing and visually exploring data about a specific topic or domain, but end users can present several necessities that differ from each other, regarding the displayed information itself, design features and even functionalities. By applying a domain engineering approach (within the software product line paradigm), it is possible to produce customized dashboards to fit into particular requirements, by the identification of commonalities and singularities of every product that could be part of the product line. Software product lines increase productivity, maintainability and traceability regarding the evolution of the requirements, among other benefits. To validate this approach, a case study of its application in the context of the Spanish Observatory for University Employability and Employment system has been developed, where users (Spanish universities and administrators) can control their own dashboards to reach insights about the employability of their graduates. These dashboards have been automatically generated through a domain specific language, which provides the syntax to specify the requirements of each user. The domain language fuels a template-based code generator, allowing the generation of the dashboards' source code. Applying domain engineering to the dashboards' domain improves the development and maintainability of these complex software products given the variety of requirements that users might have regarding their graphical interfaces.

1   **Taking advantage of the software product line**
2   **paradigm to generate customized user interfaces for**
3   **decision-making processes: a case study on**
4   **university employability**

6   Andrea Vázquez-Ingelmo[1], Francisco J. García-Peñalvo[1], Roberto Therón[1,2]
7
8   [1] GRIAL Research Group. Department of Computer Science and Automatics, University of
9   Salamanca, Salamanca, Salamanca, Spain
10 [2] VisUSAL Research Group. Department of Computer Science and Automatics, University of
11 Salamanca, Salamanca, Salamanca, Spain
12
13 Corresponding Author:
14 Andrea Vázquez-Ingelmo[1]
15 Paseo de Canalejas 169, Salamanca, Salamanca, 37008, Spain
16 Email address: andreavazquez@usal.es

ABSTRACT

University employment and, specifically, employability has gained relevance since research in
these fields can lead to improvement in the quality of life of individual citizens. However,
empirical research is still insufficient to make significant decisions, and relying on powerful
tools to explore data and reach insights on these fields is paramount. Information dashboards
play a key role in analyzing and visually exploring data about a specific topic or domain, but end
users can present several necessities that differ from each other, regarding the displayed
information itself, design features, and even functionalities. By applying a domain engineering
approach (within the software product line paradigm), it is possible to produce customized
dashboards to fit into particular requirements, by the identification of commonalities and
singularities of every product that could be part of the product line. Software product lines
increase productivity, maintainability and traceability regarding the evolution of the
requirements, among other benefits. To validate this approach, a case study of its application in
the context of the Spanish Observatory for University Employability and Employment system
has been developed, where users (Spanish universities and administrators) can control their own
dashboards to reach insights about the employability of their graduates. These dashboards have
been automatically generated through a domain specific language, which provides the syntax for
specifying the requirements of each user. The domain language fuels a template-based code
generator, allowing the generation of the dashboards' source code. Applying domain engineering
to the dashboards' domain improves the development and maintainability of these complex
software products given the variety of requirements that users might have regarding their
graphical interfaces.

INTRODUCTION

The concept of employability has increasingly gained relevance over the last decades. There is a
reason: knowing which factors increase the possibility to obtain a job or to perform better in
current job positions could be decisive to improve individual and collective life quality.

However, this concept is still far away from having a straightforward definition (Chadha &
Toner 2017). As the literature suggests, employability can be seen as a capability to gain
employment or as a set of skills and knowledge required to perform effectively in the workplace,
among other definitions (Confederation of British Industry 2009; Hillage & Pollard 1998; Yorke
2006). This lack of consensus when defining employability makes the research in this field a
complicated task, given the fact that the definition of its factors depends on the perspective used
to evaluate it, as well as the socioeconomic context in which employability and employment
studies are framed. For these reasons, nowadays research on employability asks for an
exploratory approach, to build stronger theoretical foundations.

Researching on employability has many potential benefits, aiming not only at knowing the
variables that affect the capability to gain employment and have a successful work career, but

58 also to exploit this knowledge to help policymakers and institutions with their missions. This
59 knowledge can contribute to the creation of greater policies, focusing on the detected factors to
60 enhance people's chances to obtain better employment. Specifically, educational institutions like
61 universities could benefit from this knowledge. These institutions play a vital role regarding the
62 employability of individuals (García-Peñalvo 2016), as they are in charge of transmitting
63 knowledge and a series of skills to their students. By promoting the most relevant skills and
64 capabilities that affect employability, it could be possible to increase the alignment of education
65 with the labor market.
66
67 However, generating knowledge in such a study field is not a trivial task. As it has been
68 introduced, there could be several variables involved in the research of students' employment
69 and employability, so it is necessary to collect significant data volumes to be able to reach
70 valuable insights. In addition to data collection, performing data analysis (Albright et al. 2010) is
71 required to be able to reach useful insights. It is worth noting that analyzing employability data
72 to identify and understand its factors could become a cornerstone in decision-making processes
73 within educational institutions.
74
75 Nevertheless, even after performing data analysis, identifying patterns and indicators derived
76 from the analysis outcomes remains a complex challenge. That is why it is crucial to assist
77 decision-makers with powerful tools that allow reaching insights about the domain of the
78 problem, to support decisions with complete and quality information (especially in the academic
79 context, where these processes might have a series of social implications), that is, information
80 and knowledge that has been extracted through visual analysis.
81
82 Information dashboards are one of the most commonly used software products for visual data
83 analysis and knowledge extraction (Few 2006; Sarikaya et al. 2018). In a domain like
84 employability, these tools can support exploratory studies through a set of graphical and
85 interactive resources, allowing users to envision data more understandably (Tufte & Graves-
86 Morris 2014) and identify relevant relations, indicators or patterns among large sets of data. It is
87 essential to bear in mind that information dashboards are not just a set of aesthetic graphs and
88 visualizations; they have to effectively transmit information to answer the questions of the users
89 regarding the target domain. Moreover, this is not a trivial job, because of two main reasons: data
90 and users themselves.
91
92 On the one hand, users do not have a set of standard and static requirements; they could demand
93 different features or design attributes given their specific goals or needs. On the other hand, data
94 is continuously increasing and evolving nowadays, so it is foreseeable that new information
95 requirements will arise in time. Returning to the employability subject, information requirements
96 in this domain might change in many different ways as this concept could demand new kind of

97   variables or larger amounts of data to explore emerging dimensions or to perform more in-depth
98   analyses.
99
100   For these reasons, information dashboards not only need to be useful concerning functionality
101   but also be customizable to adapt to specific user requirements. Also, they should be flexible and
102   scalable regarding its data sources and structures, making the development and maintenance of
103   information dashboards even more complicated. Of course, these issues could be addressed by
104   developing particular dashboards for each involved user to achieve every specific goal, but
105   clearly, this solution would be time-consuming and would require a lot of resources during the
106   development and maintenance phases. Also, scalability would be almost impossible, as new
107   users or changes in the requirements would necessarily imply more resources.
108
109   There are, nevertheless, a series of strategies to deal with these challenges. Specifically, software
110   engineering paradigms like software product lines (Clements & Northrop 2002; Gomaa 2004;
111   Pohl et al. 2005) provide powerful theoretical frameworks to address flexibility, scalability and
112   customization in software products that share sets of features within a common domain. Through
113   the analysis of commonalities and variability points in the product domain, it would be possible
114   to reduce the development and maintenance effort of building tailor-made solutions. This
115   paradigm is potentially applicable to dashboards since these software products could be factored
116   into sets of configurable components with configurable features. This paper describes the
117   application of the SPL methodology to the dashboards' domain through the study of their
118   characteristics and the definition of a DSL to manage the product derivation automatically. The
119   main focus of this research is to test the potential usefulness and feasibility of this approach to
120   manage fine-grained features that can be scattered through different code assets, and
121   consequently, to provide a base method for generating personalized dashboard solutions to fit
122   concrete user requirements.
123
124   The remainder of this work is structured as follows. Section 2 (Background) discusses the
125   background of the problem of generating customized dashboards as well as their application to
126   the employment and employability domain. Section 3 (Context) presents the application context
127   and the motivation behind this pilot framework to generate dashboards to support visual analysis
128   on university employment and employability data (framed within the Spanish Observatory for
129   University Employability and Employment studies). Section 4 (Materials and Methods)
130   describes the techniques used for the development of an initial approach to a generative
131   dashboard framework. Finally, Section 5 exhibits the outcomes of this research to conclude with
132   the discussion of the developed SPL and the conclusions derived from these results.
133

BACKGROUND

135   The main idea behind software product lines (SPLs) is that the final products can be derived
136   from a set of configurable core assets, allowing their adaptation to fit specific requirements.

137 These core assets are developed during the domain engineering phase, in which commonality
138 and variability of the target product domain are identified to build a common base of
139 components. Core assets are developed with variability points in which specific functionalities
140 could be injected to obtain new products. Functionalities in SPLs are seen as features; the
141 combination of the defined features within the scope of the line (generally following a feature
142 model (Kang et al. 1990) allow stakeholders to build personalized products by reusing and
143 assembling software components.

144

145 The SPL paradigm has been applied to a variety of domains: Mobile applications (Marinho et al.
146 2010; Nascimento 2008; Quinton et al. 2011); Applications for visualizing population statistics
147 (Freeman et al. 2008); Sensor data visualizations (Logre et al. 2014); Variable content
148 documents (Gómez et al. 2014); or e-Learning systems (Ezzat Labib Awad 2017).

149

150 These practical applications have proved the benefits of this paradigm. However, features usually
151 refer to the software's logic, deflecting attention to the presentation layer. The idea of generating
152 customized dashboards can be seen as a specific case of graphical user interfaces (GUI)
153 automatic generation within SPLs. User interfaces require additional work regarding their
154 implementation; they not only need to be functional but also usable to allow users to complete
155 their tasks efficiently and achieve their goals. That is why the design of user interfaces is present
156 through the whole development process, being time- and resource-consuming job.

157

158 Automation regarding GUI generation in software product lines has already been faced in several
159 works. Generally, there is a lack of usability on the generated products that can be addressed by
160 manually designing every product GUI. But this approach is highly inefficient in the SPL
161 paradigm context since all the development time saved could be lost by introducing a manual
162 task (Hauptmann et al. 2010). Integration of the GUI design process and the SPL paradigm is
163 required to leverage the benefits of the two approaches (Pleuss et al. 2010). There is, as Pleuss et
164 al. (2012) pointed out, a dilemma between automation and usability (Pleuss et al. 2012a; Pleuss
165 et al. 2012b). To address this challenge, they utilized Model-Based UI Development (MBUID)
166 methods to separate the functionality and the appearance of the GUI (Pleuss et al. 2010).

167

168 On the other hand, Gabillon et al. (2015) demonstrated the possibility of creating adaptive user
169 interfaces through the Dynamic SPL (DSPL) paradigm and MBUID models by developing a
170 context-aware data visualization tool that can be adapted during runtime.

171

172 DSPLs provide a useful paradigm for adapting code at run-time, obtaining adaptive GUIs.
173 Kramer et al. (2013) proposed document-oriented GUIs with run-time variations through XML
174 documents (Kramer et al. 2013). This context-adaptable feature has also been achieved by Sboui
175 et al. (2018), by developing a mobile application that is also runtime adaptable through MBUID
176 models and reusable artifacts. In this particular case, the code generation is based on eXtensible

177 Stylesheet Language Transformations (XSLT) and XML files (Sboui et al. 2018). These works
178 shows not only the viability of GUI generation in the SPL/DSPL paradigms context but also their
179 valuable benefits.
180
181 It seems evident that GUI customization requires fine-grained features to achieve the desired
182 usability and design attributes. Fine-grained features mostly require annotative approaches
183 regarding their implementation, given their specialization. Annotative approaches can address
184 this issue because annotations can be arbitrarily specified at different source code fragments
185 (Kästner & Apel 2008; Kästner et al. 2008), and provide a framework for fine-grained automated
186 software composition through feature structure trees (Apel et al. 2009).
187
188 There are different approaches to manage the implementation of variability at a fine-grained
189 level (Gacek & Anastasopoules 2001). Especially, frame- and template-based approaches
190 provide valuable solutions to address this fine-grained level of variability, allowing the injection
191 of particular fragments of code at any point of the base source code. Frame-based languages, like
192 XML-based Variant Configuration Language (XVCL) (Jarzabek et al. 2003), provide a syntax to
193 combine and insert fragments of code through the definition of frames, allowing the separation
194 of concerns regarding the SPL implementation (Zhang et al. 2001). Templating can also achieve
195 valuable results; templating libraries such as Jinja2 (Ronacher 2008) provide powerful
196 functionalities to annotate the source code independently of the target programming language
197 (Clark 2018; Ridge et al. 2017).
198
199 The generation of GUI within the context of a product family is still a convoluted field, although
200 the previous work has enlightened the path to improve and leverage the automation and
201 generation of these complex software elements. The complexity mainly comes from human
202 factors and the vast variety of requirements regarding user interfaces.
203
204 This work aims to present an application of the SPL paradigm, in this case on the dashboards'
205 domain, considering the fine-grained nature of their features and the necessity of customizing its
206 interaction methods and visual appearance.
207
208 CONTEXT
209 The application of this work is framed within The Spanish Observatory for University
210 Employment and Employability. The following subsections describe this organization's mission
211 and the motivation to generate personalized dashboards to explore its data.
212
213 The Observatory for University Employment and Employability
214 The Observatory for University Employment and Employability (also known as OEEU, its
215 Spanish acronym, http://oeeu.org) is an organization with the vision of becoming an information
216 reference for understanding and exploiting knowledge about employment and employability of

217   students from Spanish universities. To do so, this network of researchers and technicians conduct
218   studies about these fields in the academic context (Michavila et al. 2018a; Michavila et al. 2016;
219   Michavila et al. 2018b), through a data-driven approach to recollect, analyze, visualize and
220   disseminate employment and employability data of graduates from Spanish universities.
221
222   Firstly, in the data collection phase, universities provide their administrative records and, once
223   this phase is completed, their students answer a questionnaire about different aspects of their
224   education and work career. This process leaves the Observatory with a significant set of
225   variables from the students' sample. For instance, in the 2015 study edition, more than 500
226   variables were gathered from 13006 bachelor students. Moreover, in the 2017 study edition, 376
227   variables were gathered from 6738 master degree students.
228
229   The volume of the data collected makes the presentation of the study results to the Observatory
230   ecosystem's users a challenge, as the latter may have different requirements and necessities
231   regarding the studies' data. For these reasons, an approach based on domain engineering fits the
232   OEEU's needs, allowing an efficient generation of customized dashboards that meet different
233   requirements.
234

235   **Motivation**
236   As it has been introduced, employment and employability are complex study fields that mainly
237   ask for exploratory analysis, given its relatively initial status of research. In the context of the
238   Spanish Observatory for University Employment and Employability, where a vast set of
239   variables from significant quantities of students are recollected, it is crucial to rely on
240   exploratory visualizations that allow users and administrators to identify at a glance unusual
241   patterns or important data points by enhancing the understanding of the presented information
242   (Card 1999).
243

244   In contrast with explanatory visualizations, in which the primary purpose is to tell a story
245   through data, exploratory tools aim to facilitate users to pose more questions as data is being
246   explored. In essence, explanatory analyses start from a question and use data to answer it.
247   Exploratory analysis, on the other hand, uses data to detect new avenues of research. For
248   instance, when a user does not have a clear question about the data, it will use an exploratory
249   research to find patters or relations among variables. This same user could employ the acquired
250   knowledge to explain the insights reached through previous explorations using an explanatory
251   visualization.
252

253   Exploratory visualizations rely intensely on interaction to provide their functionality and to allow
254   users to drill-down datasets, being able to discover new aspects of the domain by directly
255   communicating with the graphical interface. However, an interaction can take many forms, and
256   there is not a single solution to obtain usable and intuitive interfaces valid for every user.

257

258   For instance, some users could find useful a visible control panel to manage data if they are
259   going to apply filters, aggregations and so on intensively. On the other hand, other users can
260   demand in-place interaction if they give more importance to having more space for the
261   visualizations (instead of having a permanent control panel consuming screen space). Another
262   example is that users that speak a left-to-right (LTR) or a right-to-left (RTL) language would
263   demand different layouts for the same task, according to their sociodemographic or cultural
264   context (Almakky et al. 2015; Marcus & Gould 2000). Also, visualization novices could require
265   task-oriented dashboards to support their visual analysis, since their past experience with this
266   kind of tools is a relevant factor when interacting with a system (Elias & Bezerianos 2011).

267

268   Once patterns, relations between variables and interesting dimensions have been identified
269   through the exploration of data, even the exploratory nature of a dashboard can change for a
270   more explanatory purpose to present the results understandably and strikingly.

271

272   For all these reasons dashboards, their components, their interaction, and even their primary
273   purpose need advanced configuration and customization to fit into different contexts and
274   requirements. Moreover, as it has been aforementioned, SPLs provide a potential solution to
275   efficiently address this customization since visual components and interaction methods could be
276   treated as features of the product line, decreasing the resources needed during the development
277   and maintenance of dashboards.

278

279   ## MATERIALS & METHODS
280   This section presents the materials and techniques used during the development of this first
281   approach to a framework for generating dashboards to explore employment- and employability-
282   related variables.

283

284   **Meta-model**
285   The problem to address requires abstract modelling to capture basic features within the
286   dashboards' domain. To do so, a meta-model is proposed. Meta-models are a crucial artefact in
287   model-driven engineering and model-driven architectures (Kleppe et al. 2003), as they allow to
288   define a high-level view of the domain without depending on specific technologies. Therefore,
289   meta-models should remain as simple as possible to eventually, through a series of mappings and
290   transformations, obtain concrete models (Álvarez et al. 2001).

291

292   For this generic dashboards' domain, the meta-model found in Fig. 1 is proposed. First of all, a
293   specific user could handle a dashboard. This dashboard could be composed of one or more
294   pages, being these last composed, in turn, by one or more containers. A container could be seen
295   as a row or a column, and it can recursively contain more containers. The container recursion
296   ends with a component, which is any graphic element that can be used in a dashboard. The

297 recursion mentioned above allows the arrangement of any layout by the recurrent combination of
298 rows and columns.
299
300 This meta-model eases the vision of the dashboards' domain, and it also allows to identify the
301 common base of any dashboard.
302

Feature model
303
304 The meta-model gives a high-level vision of the dashboards' domain. However, it does not
305 capture concrete features. That is why software product lines rely on feature models (Kang et al.
306 1990) to identify common and variable assets.
307
308 Feature models not only serve as a documentation element but also as an important artifact
309 within the development process. The implementation of the core assets and the materialization of
310 variability points on the code must be guided by the previously defined feature model.
311
312 In this domain, the feature model will capture the dashboards' visualization components, as well
313 as individual features and restrictions of each visualization. The hierarchical structure of the
314 feature model allows to define high-level characteristics and refine them through the tree
315 structure until reaching the lower-level features (i.e., fine-grained features). This structure makes
316 the scalability of features easier, since adding new features involves the addition of new nodes to
317 the feature tree uniquely.
318
319 For the Observatory's dashboards, three main configurable visual components (features) have
320 been defined: a scatter diagram, a chord diagram and a heat map. These visualizations address
321 the requirements of the Observatory's data but can be reused for other data domains. Also, it is
322 possible to specify a global filter that affects the data of all components previously defined.
323 These high-level features of the dashboards' product line are presented in Fig. 2.
324
325 A detailed view of the scatter diagram feature can be seen in Fig. 3. It has a set of subsequent
326 features, either mandatory, optional or alternative. One mandatory feature is the base logic of the
327 scatter diagram (i.e., the component layout construction and its primary logic). Another
328 mandatory feature is the initial data that the diagram will be showing on different dimensions
329 since it must be specified. Among the optional features, it is possible to determine whether a
330 tooltip will show up when hovering on data points if a set of controls will support the data
331 exploration, or the capacity to zoom or export the diagram. Also, a title for the visualization can
332 be included.
333
334 For the sake of simplicity, some of the lower-level features have been omitted in Fig. 3. For
335 instance, the bar and panel control features have subsequent features. The detailed features for a
336 panel type control are shown in Fig. 4 to provide an example. A control panel will rely on its

337   underlying logic, and it can count on different optional features, like data selectors to
338   dynamically change the visualization's presented data; in case of the X and Y axes, these
339   selectors could be located within the control panel space or in-place controls (i.e., situated near
340   the scatter diagram axes). Other possible features involve having an overview that shows a
341   detailed view of a data point when hovering, data filters, among others.
342
343   The feature diagram provides a high-level and organized overview of the SPL, improving the
344   organization of the source code and development tasks.
345
346
347   **Domain-specific language**
348   There is, however, a necessity of connecting the previous models to the dashboards' source code
349   to be generated (Voelter & Visser 2011). A Domain-Specific Language (DSL) has been designed
350   to accomplish this connection. This DSL is based on the identified domain's features, by
351   structuring them with XML technology (Bray et al. 1997) and by validating the model
352   restrictions with an XML schema (Fallside 2000). XML technology provides a readable and
353   easy-to-parse manner for injecting functionalities or requirements in a system, fostering
354   flexibility since these rules are not directly defined (or hard-coded) in the source code.
355
356   The following examples describe the DSL developed for this work. Following the meta-model,
357   every dashboard will be composed by one or more pages, each page with its configuration (i.e.,
358   layout and components, as seen in Fig. 5), and each page component with its setting (given the
359   feature model, as seen in Fig. 6).
360
361   Data resources of each visual component are represented by the XSD generic type "*anyType*", to
362   decouple the data structure and format from the presentation, and also to open up the possibility
363   of injecting dynamic data sources without affecting the DSL syntax.
364
365   In Figures 6 and 7 the resemblance of the XML schema structure with the feature model can be
366   appreciated. The hierarchical nature of XML matches with the hierarchical structure of feature
367   diagrams. This resemblance allows better traceability of the features involved in the product line,
368   because the syntax of the DSL is obtained from the feature model, thus providing a computer-
369   understandable specification of the SPL, necessary to process the requirements and to automate
370   the dashboard generation. In this current approach, the dashboard's feature model serves as
371   documentation, but, as it will be discussed, it would be extremely valuable to create a
372   programmatic link between this model and the DSL specification, in order to propagate and
373   reflecting any feature model change automatically in the DSL, improving maintainability.
374
375   Finally, Fig. 8 shows how the layout of the dashboard is specified in terms of rows, columns and
376   components (following, again, the meta-model previously presented). The DSL combines both

377   the meta-model and feature model designs to obtain a specific syntax to configure all the aspects
378   regarding the generation of final products.

379

380   The whole schema for the DSL can be consulted at the following GitHub repository
381   https://github.com/AndVazquez/dashboard-spl-assets (Vázquez-Ingelmo 2018).

382

Code generator

384   To put together all the developed assets and concepts, a code generator has been developed to
385   manage the generation of functional dashboards. The generator interprets the DSL (i.e., XML
386   configuration files) and selects the appropriate template (i.e., core assets of the SPL) to configure
387   them by injecting the chosen features, obtaining the dashboards' final source code. The code
388   templates and XML configuration files are managed by the developers following the elicited user
389   requirements.

390

391   The inputs and outputs of the code generator can be seen in Fig. 9.

392

Code templates

394   The next challenge regarding the implementation of this SPL involves the choice of the
395   techniques for materializing the product line's variability points. In this case, personalization is
396   focused on the visual elements of the system's presentation layer, which require fine-grained
397   variability (Kästner et al. 2008). Coarse-grained variability involves the addition and removal of
398   full components, which is also useful for this approach (users may prefer a scatter diagram over a
399   chord diagram to achieve their goals, removing the last from the dashboard). However, visual
400   components themselves (referring to the elements that compose them) also require high
401   variability to fit into different requirements, so fine-grained variability needs to be accomplished.
402   There exist different approaches to implement fine-grained software composition, as in the case
403   of FeatureHouse (Apel et al. 2009), which uses superimposition and feature structure trees
404   (FSTs), however, not every method supports the currently required granularity, which involves
405   even statement-level variability. Fine granularity often prohibits superimposition approaches
406   (Apel et al. 2013).

407

408   The mechanism chosen to reach the desired feature granularity is based on template engines.
409   Template engines allow to tag sections and parameterize units of source code to inject concrete
410   values later and obtain complete source files. This mechanism accomplishes the necessity of
411   materializing the variable features of a tangible product of the line.

412

413   Jinja2 (Ronacher 2008) was selected as the engine for developing the core assets of this SPL.
414   This template engine allows the definition of custom tags, filters and even macros, being the last
415   one of the essential features to organize the core assets. As described in (Kästner et al. 2008),
416   fine-grained approaches can make the source code tedious to read and maintain. By declaring

417   every variant feature on different macros to compose them subsequently, it is possible to achieve
418   high cohesion and loose coupling on the SPL feature implementation process, improving
419   reusability and source code organization by grouping the different functionalities by its parent
420   feature. There was no need to implement extensions of the Jinja2 implementation and
421   mechanisms, as its current syntax was sufficient for the annotative approach followed.
422

423   A diagram of the detailed workflow for generating the source code can be seen in Fig. 10. The
424   code templates for this case study can be also consulted at
425   https://github.com/AndVazquez/dashboard-spl-assets (Vázquez-Ingelmo 2018).
426

RESULTS

427

428

Generated dashboards

429

430

431   As it has been already introduced, the Observatory collects important datasets to research the
432   employability and employment of graduates from Spanish universities. Relying on a
433   customizable exploratory tool would increase the chances of discovering interesting patterns or
434   relations within these complex fields. The dashboards of this case study have a series of
435   particular requirements due to the data domain and the specific characteristics of the Observatory
436   studies. For instance, the developed data visualizations exploit different dimensions of the
437   Observatory's collected variables. Also, the generated Observatory's dashboards needed to be
438   connected to the organization's GraphQL API (Facebook 2016) that allow users to retrieve data
439   statistics on demand (Vázquez-Ingelmo et al. 2017; Vázquez-Ingelmo et al. 2018a; Vázquez-
440   Ingelmo et al. 2018b; Vázquez-Ingelmo et al. 2018c), decoupling the data resources from the
441   visual components' logic.
442

443   In this section, the results derived from the application of the presented dashboard product line
444   within the university employment and employability domain are described. By tuning SPL
445   through particular configurations, it is possible to obtain tailored solutions for different
446   requirements and tasks.
447

448   **Configuration #1.** Comparison of different values is one of the most relevant tasks regarding the
449   exploration of university employability and employment data. These comparisons could
450   enlighten which factors affect employability and employment to a greater or lesser extent,
451   leading to the possibility of conducting deeper analyses.
452

453   For example, by configuring a dashboard with two scatter diagrams side by side, it is possible to
454   apply different filters to each one and observe how data patterns evolve (Fig. 11). Also, adding
455   the global reference feature to both diagrams helps to make comparisons by adding a reference
456   line marking the unfiltered and disaggregated values.

457
458   It is possible to appreciate, for example, that men graduates are more optimistic when
459   commenting opinions about their future wages and the possibility of developing a working career
460   in Spain (Michavila et al. 2018b). However, these diagrams also allow seeing at a glance that
461   Arts and Humanities and Sciences graduates are more pessimistic about their future than their
462   counterparts in other branches of knowledge, which are more clustered. For instance, only 40%
463   of Sciences women graduates think that they could have a working career in Spain within five
464   years.
465
466   This configuration enables the user to explore data through the combination of different
467   aggregations, variables and filters.
468
469   **Configuration #2.** The previous configuration, however, could be complex for some users by
470   having to control two diagrams at the same time to align different factors. A single scatter
471   diagram could be added to the dashboard to drill-down data. It is possible to add another
472   dimension to the scatter diagram component by mapping numerical variables through the radius
473   of the visualization's data points.
474
475   For instance, following the same example of the first configuration, the differences between male
476   and females can be observed by a gender aggregation of the data. In this case, the population of
477   each group is mapped through the radius of the points (Fig. 12).
478
479   However, to see how the branch of knowledge affects the value of these variables, similarly to
480   the previous configuration, it is necessary to continuously filter data by every single branch (Fig.
481   13). This configuration is then not recommendable when continuous, and more complex
482   comparisons (such as the one made in the previous scenario) are required.
483
484   If, on the other hand, data exploration is not continuously required by a user, the controls could
485   be allocated within a top bar (Fig. 14) that can be hidden to give more space to the visualizations.
486
487   **Configuration #3.** On the other hand, different pages focused on different data variables or data
488   dimensions could be configured. This functionality allows freedom when arranging the content
489   of the dashboards' pages to make it understandable for every particular user.
490
491   In the Observatory's case, a user might prefer having the dashboard screens organized by the
492   study edition, being able to navigate through them thanks to a navigation bar (Fig. 15).
493
494   Or if preferred, it could be specified that each page will exploit a different set of data variables;
495   for example, having a single tab to explore the students' competences (Fig. 16).
496

497 Through this view it is possible to see a misalignment between the perceived level that the
498 graduates have about their skills and the perceived level of contribution of the studies regarding
499 the acquisition of that skills, and also between that possessed level and the perceived required
500 level in their job positions (Michavila et al. 2018b).
501
502 The previous dashboards are a quite tiny set of the available combinations that can be achieved
503 through the SPL configuration, but they should serve as an example to show the possibilities of
504 having a framework for generating personalized dashboards.
505
Product metrics
507
508 The metrics for the SPL are the following regarding its feature model:
509 • Feature model height: 9
510 • Features: 146
511 • Optional features: 106
512
513 The number of valid configurations has been omitted, given the recursion of the dashboards'
514 composition (as highlighted in the dashboard meta-model), so infinite valid configurations can
515 be generated.
516
517 Regarding the core-assets (i.e. the templates' source code), the following metrics have been
518 calculated (El-Sharkawy et al. 2018):
519 • Lines of feature code (LoF): 2638 lines of feature code. This metric is the addition of
520 every line of code affected by any Jinja2 directive (i.e. every annotated line of code). It is a
521 size metric that gives a high-level view about the source code associated to the SPL features.
522
523 • Fraction of annotated lines of code (PLoF): 48.39%. This is a variability density metric
524 showing that the SPL's products have a 51.61% of common code (2814 lines of code are not
525 annotated).
526
527 • Scattering of variation points: this metric counts the number of times that a feature
528 appears in the code (i.e. appears in a Jinja2 condition directive). High scattering values
529 decreases the readability of the code. By refactoring the code into macros that contain all
530 code associated to a specific feature, the scattering is reduced.
531
532 Given the complex domain in which the product line has been applied (i.e. the dashboards'
533 domain), the scattering of the variation points was one of the main concerns, as high scattering
534 would make the code even more complex. That was the reason to arrange the feature code into
535 macros as a solution to address the scattering of variability points.
536

537 **Development time improvement**

538

539 The development of the presented SPL, including its conceptualization and design, took 82 days,
540 as illustrated through a simplified Gantt diagram in Fig. 17. The core assets development task
541 includes all the artifacts regarding the SPL (i.e. the DSL, the templates, etc.).

542

543 Before implementing this approach, a dashboard template with the same components and KPIs
544 was the solution to offer all the results held in the Observatory's study, so universities could
545 compare their individual results with the global, aggregated results. The development of the
546 mentioned dashboard template took 15 days. However, this static approach limited universities
547 to freely explore their data, as mentioned in other sections.

548

549 Five of the 50 universities were interviewed to capture their dashboard requirements and to
550 estimate the elicitation process time consumption. However, this estimation should be considered
551 as speculative given the variability of the complexity of the elicitation process, and specially,
552 given the number of different universities (i.e. users) involved. Nevertheless, the requirement
553 elicitation took one day for the interviewed universities.
554 Given the project's potential continuity, the dashboard implementation process would mainly
555 consume time regarding requirements elicitation by using the presented SPL approach,
556 decreasing the time spent on development processes. Without this approach, the information
557 dashboards implemented for future Observatory's employability study editions would remain
558 static and generalized for each involved user.

559

560 Building a personalized dashboard consume resources in terms of requirement elicitation and
561 design, but also in terms of implementation or development. If the development phases are
562 automated, then the main benefit is not only decreasing the development time of individual
563 dashboards, but also, if necessary, devoting more time to the requirements identification and
564 design phases, which, in the end, are the backbone of well-constructed dashboards. That is why,
565 although significant time was consumed for the implementation of the dashboard SPL (82 days),
566 it can be seen as an investment for the future, specifically in environments where significant
567 quantities of user profiles are involved.

568

569 **DISCUSSION**

570 The application of domain engineering and the SPL paradigm to identify and factorize
571 information dashboard functionalities has shown its usefulness to generate different dashboards
572 with a set of common assets through the study of the dashboards' domain. The obtained results
573 are fairly valuable, and open new paths for applying this approach to other data domains with
574 new requirements.

575

576 Dashboards are complex software solutions that could be highly beneficial when adequately

577 designed and tailored for specific users. These products can support decision-making processes,
578 assisting visual analysis by presenting information in an understandable manner. However, the
579 variety of profiles involved in these processes and their different definitions of "understandable"
580 makes the implementation of dashboards a time- and resource-consuming task, since a dashboard
581 configuration that is highly useful for one user could be pointless for the rest of them. What is
582 more, dashboards can be composed of several elements, from simple visualizations to different
583 linked views, cross-filtering capabilities, interaction methods, handlers, etc., thus making the
584 dashboards' domain a complex domain not only because of the different profiles of potential
585 users, but because of the great quantity of feasible combinations of these "dashboard elements"
586 to build a proper solution. In addition, these features can be very fine-grained; in user-centered
587 systems, a slight modification on visualization types, interaction patterns, layouts, color palettes,
588 etc. could be crucial regarding the final perceived usability of the product.
589

590 Relying on a framework to easily generate information dashboards would allow stakeholders to
591 focus on the information requirements and their refinement to provide better results when
592 seeking valuable insights on large datasets. Also, it opens up the possibility to automatically
593 adapt the dashboards' configurations to match dynamic requirements based on the device used
594 (Cruz-Benito et al. 2018b) or other factors.
595

596 The factorization of the dashboards' components into individual features allow fine-grained
597 reusability and a set of customization options. This fine-grained customization enables the
598 possibility of having highly functional and exploratory-centered visualizations as well as more
599 basic visual components more centered on the explanation of insights through the addition or
600 removal of low-level features. The achieved granularity provides a foundation to develop not
601 only whole visualization components, but also new interaction methods and design features that
602 can be easily interchangeable to fulfill particular sets of user requirements.
603

604 An annotative method of implementation was undertaken using macros to encapsulate individual
605 functionalities. This method takes all the benefits from the annotative approach (fine-grained
606 customization) and avoids its code verbosity and scalability issues by dividing the core assets
607 into base templates and macros (Kästner et al. 2008). Although there were other possibilities to
608 implement the variability points, such as superimposition approach (which did not fulfilled the
609 requirements for performing this approach, as discussed in the Materials & Methods section) like
610 the FeatureHouse framework (Apel et al. 2009; Apel & Lengauer 2008) or the XVCL (Jarzabek
611 et al. 2003) mechanism (which fits the feature granularity requirements of this domain), the final
612 decision of using a templating engine allowed the direct connection of the designed DSL with
613 the final source code, providing a higher level language to specify the dashboards' features, as
614 well as the possibility of organizing the variability points into macros to increase readability,
615 traceability and maintainability by having all the code associated to a feature in the same source
616 file.
617

618 The chosen technology to implement the DSL was XML. The decision of implementing directly
619 the DSL in XML technology was made because of the hierarchical nature of XML, and its
620 resemblance to the hierarchical structure of the feature diagram, thus being the designed DSL a
621 computer-understandable "translation" of the feature model for the dashboard generator to
622 process. However, this language could be not as human-readable as other DSL solutions,
623 generating issues if a non-expert user wants to specify its dashboards requirements by himself.
624 Creating a friendly user interface to allow the dashboards' feature selection without involving
625 direct manipulation of the XML files can be a valuable solution to address these issues and ease
626 the product configuration process in the future.
627
628 Customization at functionality level has illustrated to be straightforward, as it is possible to
629 easily vary the behavior of the visual components through the DSL. Visual design attributes
630 customization, however, needs to be faced more deeply, as only the layout composition can be
631 specified in detail at the moment. The visual customization challenge cannot be overlooked since
632 dashboards not only have to provide valuable functionality; they should offer that functionality
633 through a pleasant and usable interface (Few 2006; Sarikaya et al. 2018).
634
635 On the other hand, this work has addressed customization focused on the presentation layer of
636 dashboards, but with the SPL paradigm, architectural design can also be customized in order to
637 provide different functional features regarding data processing, interoperability, storage,
638 performance, security, etc., achieving a complete customizable dashboard solution, not only
639 focusing on the visual presentation.
640
641 Regarding data acquisition, the developed tool was integrated with the Observatory's GraphQL
642 API to provide dynamic data exploration. The connection to this particular type of data source
643 involved the implementation of specific connectors to decouple the visualizations from the
644 particular source. The variability of data sources is another identified challenge to be addressed
645 through this approach, to support different data formats or data structures. Although counting on
646 a GraphQL API facilitated the data retrieval by the unification of data requests, it is essential to
647 enable the specification of other data retrieval methods.
648
649 Product metrics showed that significant feature code was needed to address high customizability
650 of the dashboards (48.39% of the source code was annotated). Also, arranging the feature code
651 into macros helped to increase features' traceability as well as to decrease the scattering of the
652 variability points throughout the code, making the code more readable and maintainable.
653
654 The approach can decrease the development time of individualized dashboards for each involved
655 university. As presented in the results section, the SPL not only offered space for development
656 time improvements, but also enabled the capacity of offering customized solutions, which was
657 previously regarded as unviable given the time constraints of the Observatory's project.

658   Embracing the SPL paradigm can be seen as an investment for the future for projects with a
659   common domain and with continuity over time.
660
661   Finally, it is clear that interesting patterns can be discovered thanks to the application of this
662   dashboard SPL on the employability and employment fields. The Observatory's data provide a
663   great context to perform more advanced analyses to enlighten this complex domain.
664   Having powerful visualization tools allow reaching insights about patterns or factors to guide the
665   execution of more complex analyses and make decisions about the actions to take or the future
666   research directions, like developing machine learning (ML) models (García-Peñalvo et al. 2018).
667   Regarding this last field, having visualization tools to explore the input data before training any
668   ML model could help to build better and more accurate models through an appropriate feature
669   selection phase guided by the previously reached insights (Hall 1999).
670
671   The main weaknesses and limitations of this solution come from the preliminary nature of the
672   framework; it is crucial to further validate the usability of the automatically generated products
673   to show their usefulness to the main beneficiaries of the dashboards: the users, as well as assess
674   its implementation in other domains. The approach needs to be further generalized to provide a
675   more versatile method and to match also development requirements (available technology or
676   preferred programming languages), although results seem promising. Automating the generation
677   of dashboards given their goal, their context, their end users, etc. could be extremely beneficial
678   due to the vast potential of impact that these tools have (Sarikaya et al. 2018).
679

680   ## CONCLUSIONS

681   A domain engineering approach has been applied to the dashboards' domain to obtain a SPL of
682   this type of software solution. By the identification of commonalities and variability points, a
683   dashboard meta-model has been developed as well as a feature model to capture the different
684   customization dimensions of the SPL.
685
686   The SPL has been developed through an annotative approach using code templates and macros
687   (forming the core assets of the family of products). A DSL has been designed to facilitate and
688   automate the application engineering process. The configuration files based on the DSL feed a
689   code generator in charge of adding or removing the product features. The presented approach
690   was applied within the Spanish Observatory for University Employability and Employment
691   system, to provide a variety of dashboard configurations that enable the exploitation and
692   exploration of different dimensions regarding employability and employment data.
693
694   Future research lines will involve refinements of the meta-model and the DSL, usability testing
695   of the obtained products and A/B testing (Cruz-Benito et al. 2018a; Cruz-Benito et al. 2018b;
696   Kakas 2017; Siroker & Koomen 2013) on different configurations. Architectural customization
697   could be supported to add more coarse-grained features like a visualization recommendation

698 engine (Gotz & Wen 2009; Vartak et al. 2017; Voigt et al. 2012), interface language translation
699 or data preprocessing techniques before its presentation. Finally, the customization levels of the
700 dashboards' visual design and data sources need to be further addressed.

701

702 **REFERENCES**

703 Albright SC, Winston W, and Zappe C. 2010. *Data analysis and decision making*: Cengage
704   Learning.
705 Almakky H, Sahandi R, and Taylor J. 2015. The Effect of Culture on User Interface Design of
706   Social Media - A Case Study on Preferences of Saudi Arabians on the Arabic User
707   Interface of Facebook. *World Academy of Science, Engineering*
708 *Technology International Journal of Social, Behavioral, Educational, Economic, Business*
709 *Industrial Engineering* 9:107-111.
710 Álvarez JM, Evans A, and Sammut P. 2001. Mapping between levels in the metamodel
711   architecture. International Conference on the Unified Modeling Language: Springer. p
712   34-46.
713 Apel S, Kastner C, and Lengauer C. 2009. FEATUREHOUSE: Language-independent,
714   automated software composition. Proceedings of the 31st International Conference on
715   Software Engineering: IEEE Computer Society. p 221-231.
716 Apel S, Kästner C, and Lengauer C. 2013. Language-independent and automated software
717   composition: The FeatureHouse experience. *IEEE Transactions on Software Engineering*
718   39:63-79.
719 Apel S, and Lengauer C. 2008. Superimposition: A language-independent approach to software
720   composition. International Conference on Software Composition: Springer. p 20-35.
721 Bray T, Paoli J, Sperberg-McQueen CM, Maler E, and Yergeau F. 1997. Extensible markup
722   language (XML). *World Wide Web Journal* 2:27-66.
723 Card M. 1999. *Readings in information visualization: using vision to think*: Morgan Kaufmann.
724 Chadha D, and Toner J. 2017. Focusing in on employability: using content analysis to explore
725   the employability discourse in UK and USA universities. *International Journal of*
726   *Educational Technology in Higher Education* 14:33.
727 Clark S. 2018. Render your first network configuration template using Python and Jinja2.
728   *Available at* https://blogs.cisco.com/developer/network-configuration-template.
729 Clements P, and Northrop L. 2002. *Software product lines*: Addison-Wesley.
730 Confederation of British Industry. 2009. Future fit: Preparing graduates for the world of work.
731   CBI London.
732 Cruz-Benito J, Sánchez-Prieto JC, Vázquez-Ingelmo A, Therón R, García-Peñalvo FJ, and
733   Martín-González M. 2018a. How different versions of layout and complexity of web
734   forms affect users after they start it? A pilot experience. In: Rocha Á, Adeli H, Reis LP,
735   and Costanzo S, eds. *Trends and Advances in Information Systems and Technologies*.
736   Cham: Springer, 971-979.
737 Cruz-Benito J, Vázquez-Ingelmo A, Sánchez-Prieto JC, Therón R, García-Peñalvo FJ, and
738   Martín-González M. 2018b. Enabling adaptability in web forms based on user
739   characteristics detection through A/B testing and machine learning. *IEEE Access* 6:2251-
740   2265. 10.1109/ACCESS.2017.2782678

741 El-Sharkawy S, Yamagishi-Eichler N, and Schmid K. 2018. Metrics for analyzing variability and
742   its implementation in software product lines: A systematic literature review. *Information*
743   *Software Technology*.
744 Elias M, and Bezerianos A. 2011. Exploration views: understanding dashboard creation and
745   customization for visualization novices. IFIP Conference on Human-Computer
746   Interaction: Springer. p 274-291.
747 Ezzat Labib Awad A. 2017. Enforcing Customization in e-Learning Systems: an ontology and
748   product line-based approach.
749 Facebook. 2016. GraphQL. *Available at* https://facebook.github.io/graphql/ (accessed 24th Jul.
750   2017.
751 Fallside DC. 2000. XML schema part 0: Primer. *W3C, April 2000*.
752 Few S. 2006. Information dashboard design.
753 Freeman G, Batory D, and Lavender G. 2008. Lifting transformational models of product lines:
754   A case study. International Conference on Theory and Practice of Model
755   Transformations: Springer. p 16-30.
756 Gacek C, and Anastasopoules M. 2001. Implementing product line variabilities. ACM SIGSOFT
757   Software Engineering Notes: ACM. p 109-117.
758 García-Peñalvo FJ. 2016. The Third Mission. *Education in the Knowledge Society* 17:7-18.
759 García-Peñalvo FJ, Cruz-Benito J, Martín-González M, Vázquez-Ingelmo A, Sánchez-Prieto JC,
760   and Therón R. 2018. Proposing a machine learning approach to analyze and predict
761   employment and its factors. *International Journal of Interactive Multimedia and*
762   *Artificial Intelligence* 5(2):39-45. 10.9781/ijimai.2018.02.002
763 Gomaa H. 2004. *Designing Software Product Lines with UML: From Use Cases to Pattern-*
764   *Based Software Architectures*: Addison Wesley Longman Publishing Co., Inc.
765 Gómez A, Penadés MC, Canós JH, Borges MR, and Llavador M. 2014. A framework for
766   variable content document generation with multiple actors. *Information and Software*
767   *Technology* 56:1101-1121.
768 Gotz D, and Wen Z. 2009. Behavior-driven visualization recommendation. Proceedings of the
769   14th international conference on Intelligent user interfaces: ACM. p 315-324.
770 Hall MA. 1999. Correlation-based feature selection for machine learning.
771 Hauptmann B, Bauer B, Harhurin A, and Pleuss A. 2010. Supporting derivation and
772   customization of user interfaces in software product lines using the example of web
773   applicationsMaster's thesis. Technische Universität München.
774 Hillage J, and Pollard E. 1998. *Employability: developing a framework for policy analysis*: DfEE
775   London.
776 Jarzabek S, Bassett P, Zhang H, and Zhang W. 2003. XVCL: XML-based variant configuration
777   language. Proceedings of the 25th International Conference on Software Engineering:
778   IEEE Computer Society. p 810-811.
779 Kakas AC. 2017. A/B Testing.
780 Kang KC, Cohen SG, Hess JA, Novak WE, and Peterson AS. 1990. Feature-oriented domain
781   analysis (FODA) feasibility study. Carnegie-Mellon Univ Pittsburgh Pa Software
782   Engineering Inst.
783 Kästner C, and Apel S. 2008. Integrating compositional and annotative approaches for product
784   line engineering. Proc GPCE Workshop on Modularization, Composition and Generative
785   Techniques for Product Line Engineering. p 35-40.

786 Kästner C, Apel S, and Kuhlemann M. 2008. Granularity in software product lines. Proceedings
787   of the 30th international conference on Software engineering: ACM. p 311-320.
788 Kleppe AG, Warmer J, and Bast W. 2003. MDA Explained. The Model Driven Architecture:
789   Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.
790 Kramer D, Oussena S, Komisarczuk P, and Clark T. 2013. Using document-oriented GUIs in
791   dynamic software product lines. ACM SIGPLAN Notices: ACM. p 85-94.
792 Logre I, Mosser S, Collet P, and Riveill M. 2014. Sensor data visualisation: a composition-based
793   approach to support domain variability. European Conference on Modelling Foundations
794   and Applications: Springer. p 101-116.
795 Marcus A, and Gould EW. 2000. Crosscurrents: cultural dimensions and global Web user-
796   interface design. *interactions* 7:32-46.
797 Marinho FG, Lima F, Ferreira Filho JB, Rocha L, Maia ME, de Aguiar SB, Dantas VL, Viana
798   W, Andrade RM, and Teixeira E. 2010. A software product line for the mobile and
799   context-aware applications domain. International Conference on Software Product Lines:
800   Springer. p 346-360.
801 Michavila F, Martínez JM, Martín-González M, García-Peñalvo FJ, and Cruz Benito J. 2018a.
802   Empleabilidad de los titulados universitarios en España. Proyecto OEEU. *Education in*
803   *the Knowledge Society* 19:21-39. 10.14201/eks20181912139
804 Michavila F, Martínez JM, Martín-González M, García-Peñalvo FJ, and Cruz-Benito J. 2016.
805   *Barómetro de empleabilidad y empleo de los universitarios en España, 2015 (Primer*
806   *informe de resultados)*. Madrid: Observatorio de Empleabilidad y Empleo Universitarios.
807 Michavila F, Martínez JM, Martín-González M, García-Peñalvo FJ, Cruz-Benito J, and Vázquez-
808   Ingelmo A. 2018b. *Barómetro de empleabilidad y empleo universitarios. Edición Máster*
809   *2017*. Madrid, España: Observatorio de Empleabilidad y Empleo Universitarios.
810 Nascimento LMd. 2008. Cores assets development in software product lines-towards a practical
811   approach for the mobile game domain.
812 Pleuss A, Botterweck G, and Dhungana D. 2010. Integrating automated product derivation and
813   individual user interface design.
814 Pleuss A, Hauptmann B, Dhungana D, and Botterweck G. 2012a. User interface engineering for
815   software product lines: the dilemma between automation and usability. Proceedings of
816   the 4th ACM SIGCHI symposium on Engineering interactive computing systems: ACM.
817   p 25-34.
818 Pleuss A, Hauptmann B, Keunecke M, and Botterweck G. 2012b. A case study on variability in
819   user interfaces. Proceedings of the 16th International Software Product Line Conference-
820   Volume 1: ACM. p 6-10.
821 Pohl K, Böckle Gn, and Linden FJvd. 2005. *Software Product Line Engineering: Foundations,*
822   *Principles and Techniques*: Springer-Verlag New York, Inc.
823 Quinton C, Mosser S, Parra C, and Duchien L. 2011. Using multiple feature models to design
824   applications for mobile phones. Proceedings of the 15th International Software Product
825   Line Conference, Volume 2: ACM. p 23.
826 Ridge B, Gaspar T, and Ude A. 2017. Rapid state machine assembly for modular robot control
827   using meta-scripting, templating and code generation. Humanoid Robotics (Humanoids),
828   2017 IEEE-RAS 17th International Conference on: IEEE. p 661-668.
829 Ronacher A. 2008. Jinja2 Documentation. Welcome to Jinja2—Jinja2 Documentation (2.8-dev).
830 Sarikaya A, Correll M, Bartram L, Tory M, and Fisher D. 2018. What Do We Talk About When
831   We Talk About Dashboards? *IEEE Transactions on Visualization Computer Graphics*.

832 Sboui T, Ayed MB, and Alimi AM. 2018. A UI-DSPL Approach for the Development of
833   Context-Adaptable User Interfaces. *IEEE Access* 6:7066-7081.
834 Siroker D, and Koomen P. 2013. *A/B testing: The most powerful way to turn clicks into*
835   *customers*: John Wiley & Sons.
836 Tufte E, and Graves-Morris P. 2014. The visual display of quantitative information.; 1983.
837 Vartak M, Huang S, Siddiqui T, Madden S, and Parameswaran AJASR. 2017. Towards
838   visualization recommendation systems. 45:34-39.
839 Vázquez-Ingelmo A. 2018. Code repository that supports the research presented in the paper
840   "Taking advantage of the software product line paradigm to generate customized user
841   interfaces for decision-making processes: a case study on university employability".
842   *Available at* https://github.com/AndVazquez/dashboard-spl-assets.
843 Vázquez-Ingelmo A, Cruz-Benito J, and García-Peñalvo FJ. 2017. Improving the OEEU's data-
844   driven technological ecosystem's interoperability with GraphQL. In: Dodero JM, Ibarra
845   Sáiz MS, and Ruiz Rube I, eds. *Fifth International Conference on Technological*
846   *Ecosystems for Enhancing Multiculturality (TEEM'17) (Cádiz, Spain, October 18-20,*
847   *2017)* New York, NY, USA: ACM.
848 Vázquez-Ingelmo A, García-Peñalvo FJ, and Therón R. 2018a. Application of domain
849   engineering to generate customized information dashboards. HCI International 2018 20th
850   Conference on Human-Computer Interaction (15-20 July 2018). Las Vegas, Nevada,
851   USA: Springer.
852 Vázquez-Ingelmo A, García-Peñalvo FJ, and Therón R. 2018b. Domain engineering for
853   generating dashboards to analyze employment and employability in the academic
854   context. 6th International Conference on Technological Ecosystems for Enhancing
855   Multiculturality. Salamanca, Spain.
856 Vázquez-Ingelmo A, García-Peñalvo FJ, and Therón R. 2018c. Generation of customized
857   dashboards through software product line paradigms to analyse university employment
858   and employability data. Learning Analytics Summer Institute Spain 2018 – LASI-SPAIN
859   2018. León, Spain.
860 Voelter M, and Visser E. 2011. Product line engineering using domain-specific languages.
861   Software Product Line Conference (SPLC), 2011 15th International: IEEE. p 70-79.
862 Voigt M, Pietschmann S, Grammel L, and Meißner K. 2012. Context-aware recommendation of
863   visualization components. The Fourth International Conference on Information, Process,
864   and Knowledge Management (eKNOW): Citeseer. p 101-109.
865 Yorke M. 2006. *Employability in higher education: what it is-what it is not*: Higher Education
866   Academy York.
867 Zhang H, Jarzabek S, and Swe SM. 2001. XVCL approach to separating concerns in product
868   family assets. International Symposium on Generative and Component-Based Software
869   Engineering: Springer. p 36-47.
870

Figure 1

Dashboard meta-model

The dashboard meta-model allows a high level view of the target domain.

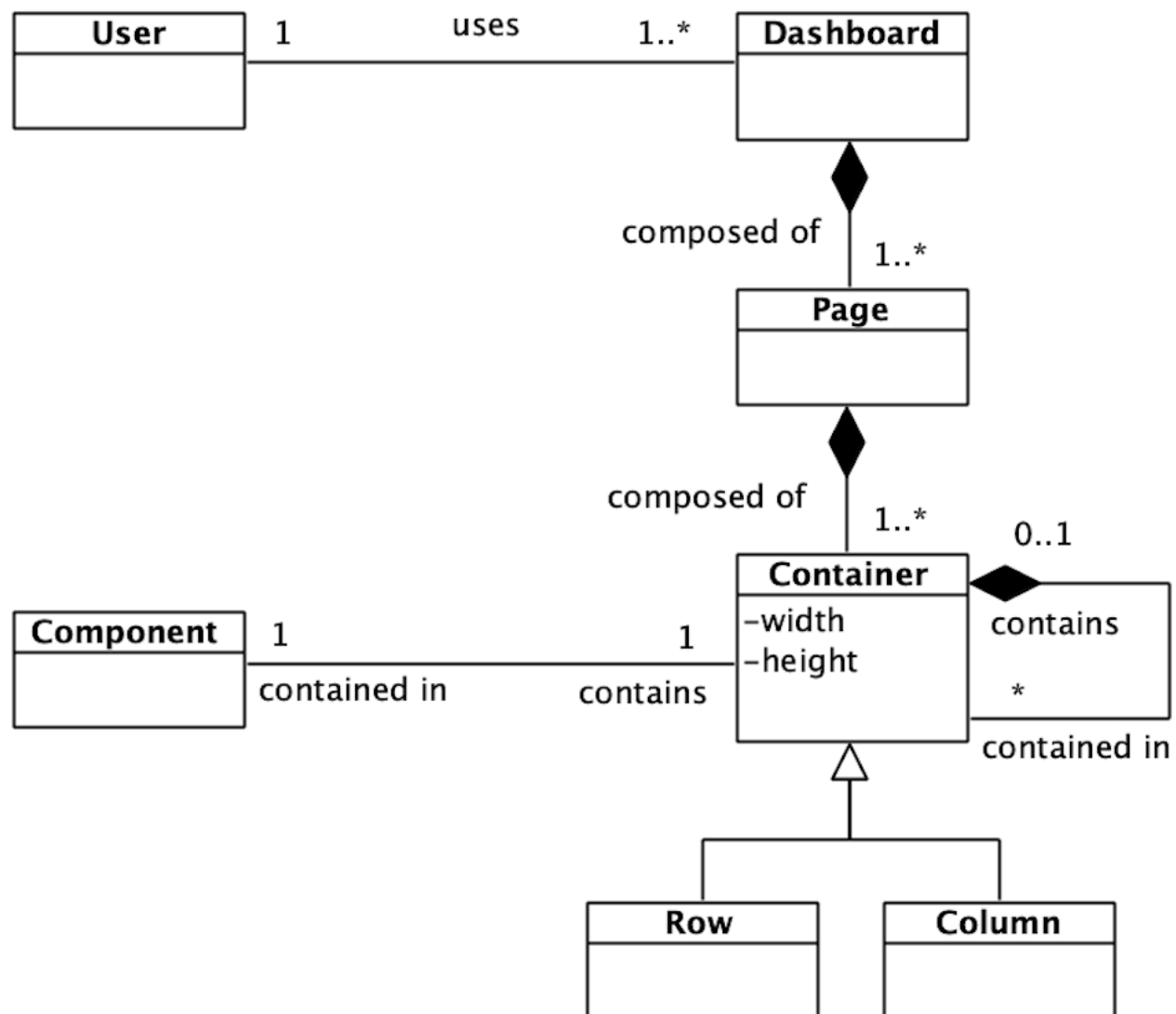



Figure 2

High-level view of the feature diagram

This feature diagram shows high-level components that could compose the dashboard.

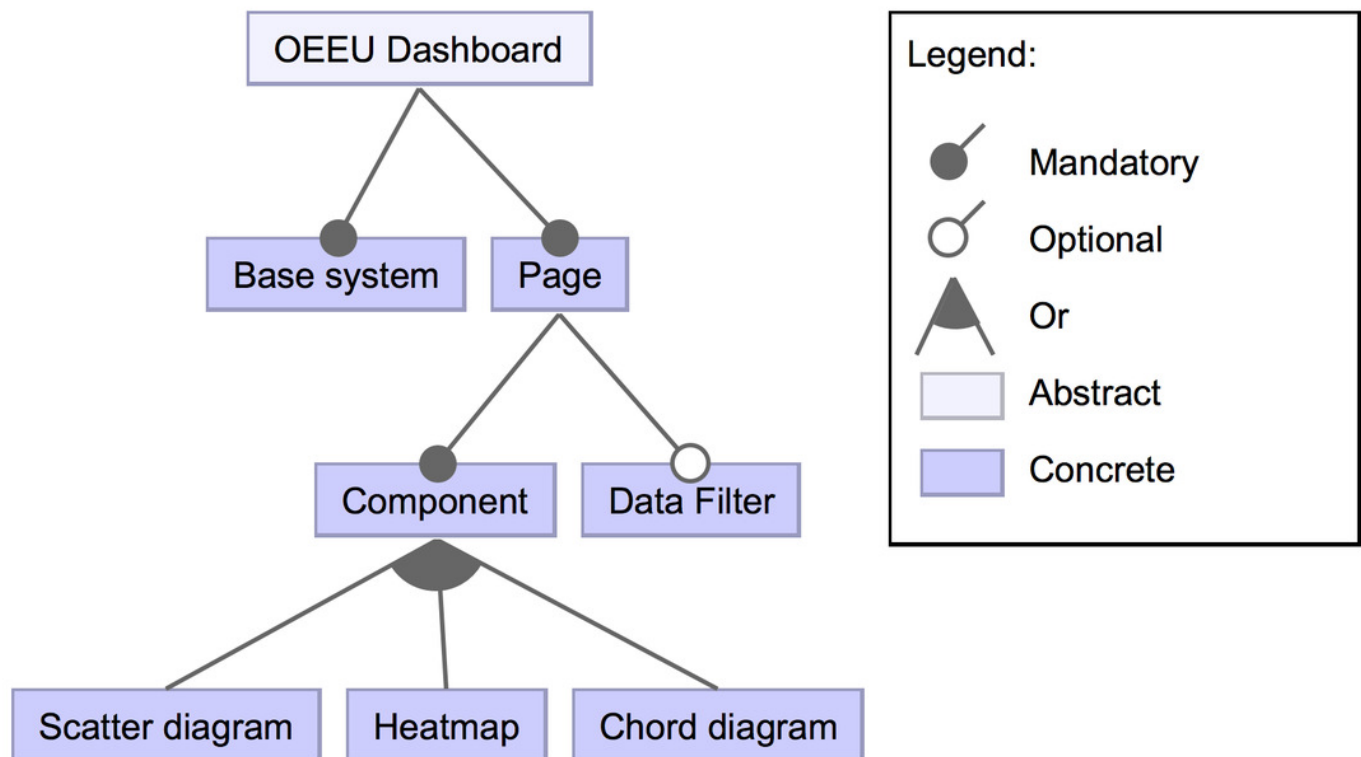



Figure 3

High-level view of the scatter diagram component's features

This snippet of the feature model shows the possible features regarding the scatter diagram component.

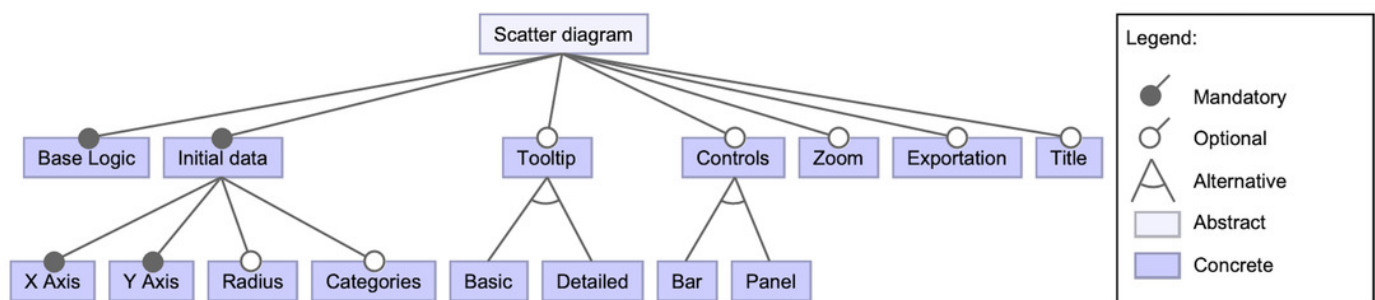



Figure 4

High-level view of a component's panel subsequent features

This part of the feature diagram shows lower-level features regarding the components'
control panel.

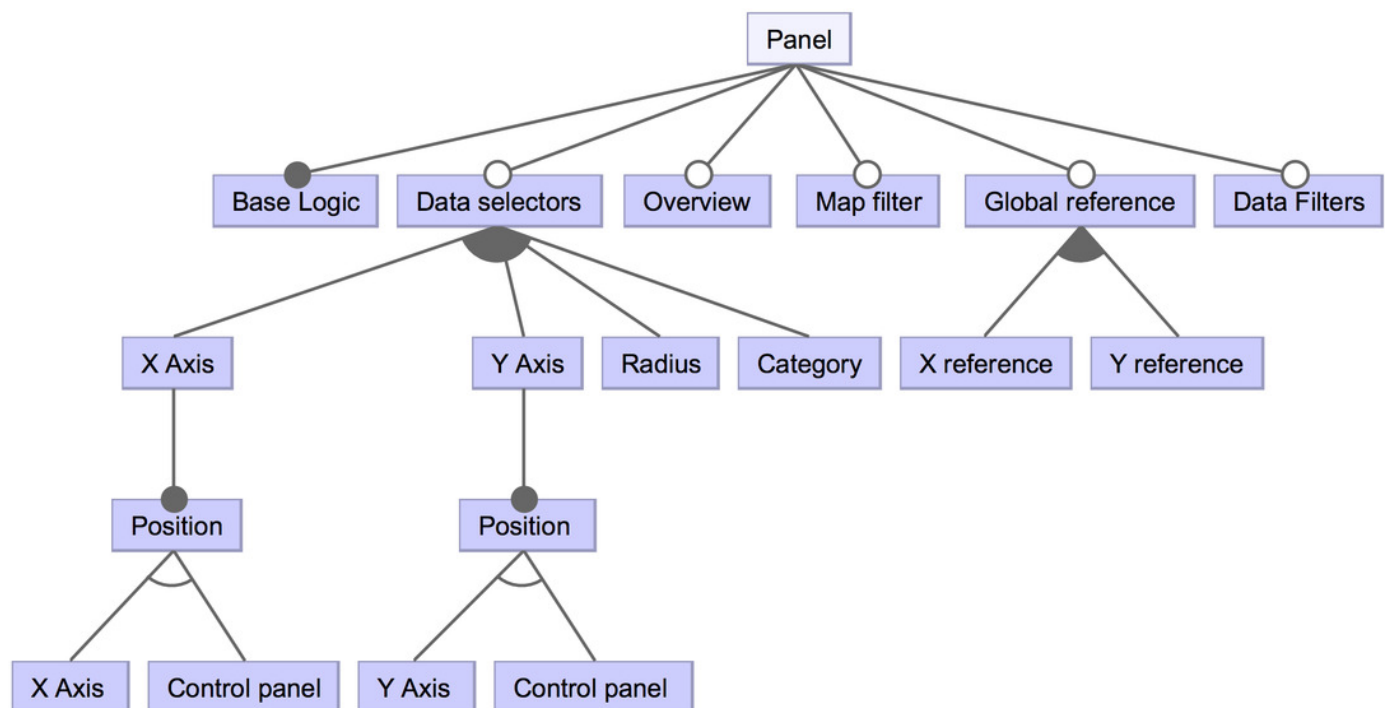



Figure 5

Snippet of the DSL schema

It is possible to specify the dashboard layout and its elements (i.e. data filters, components, etc.)

```xml
<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: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-2">
                    <xs:selector xpath="Page"/>
                    <xs:field xpath="@page_id"/>
                </xs:unique>
            </xs:element>
        </xs:choice>
    </xs:complexType>
</xs:element>
```

Figure 6

DSL schema regarding the specification of the dashboard components

It is possible to see the link between the feature model elements and the XML schema

elements (e.g. the components that could compose the dashboard)

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

Figure 7

DSL schema regarding the specification of the scatter diagram component

This part of the DSL represents the available features for the scatter diagram component.

```xml
<xs:element name="ScatterDiagram">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="Title" type="xs:string"...>
            <xs:element name="Zoom" type="xs:string"...>
            <xs:element name="Exportation"...>
            <xs:element name="InitialData"...>
            <xs:element name="Controls" minOccurs="0"...>
            <xs:element name="Tooltip" minOccurs="0"...>
        </xs:sequence>
        <xs:attribute name="component_id"
                      type="xs:string"/>
    </xs:complexType>
</xs:element>
```

Figure 8

XML type for specifying the dashboard's layout

The dashboard layout (previously modeled through the dashboard meta-model) is specified by creating a custom type.

```xml
<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>
```

Figure 9

Code generator inputs and outputs

The code generator is fed with the code templates and the XML configuration files to provide the final source code of the dashboard.

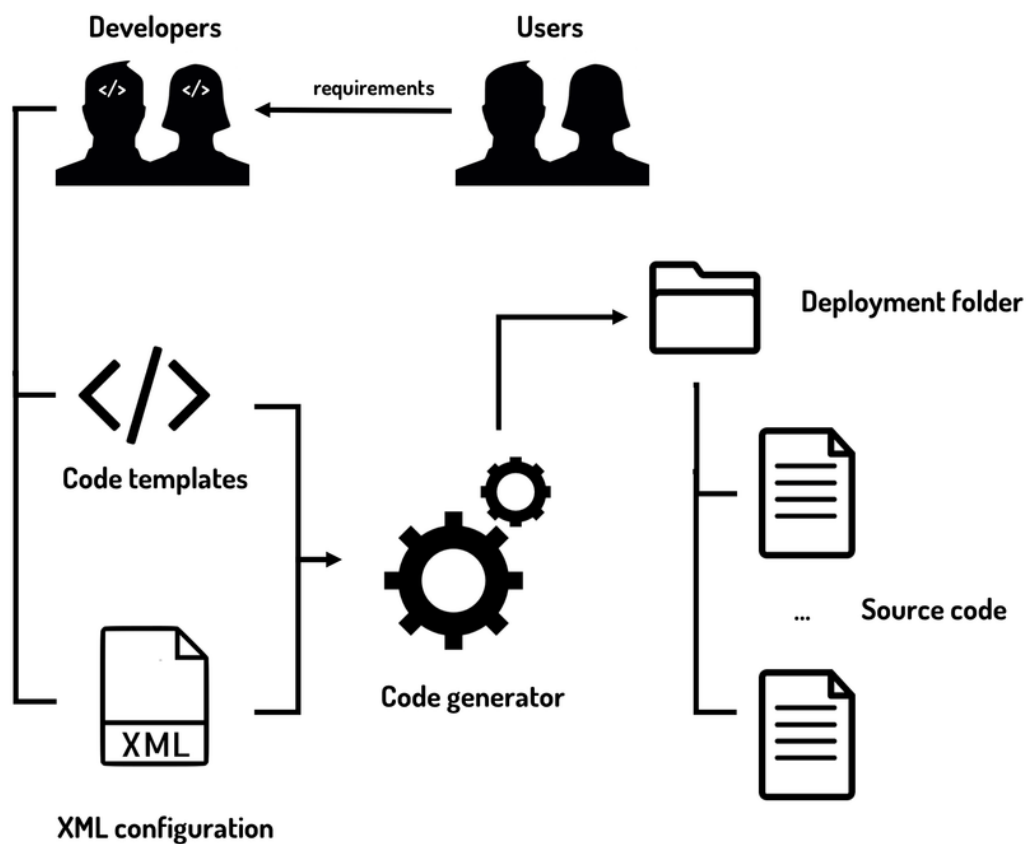



Figure 10

Workflow of the code generation process

A simplified view of the code generator behavior.

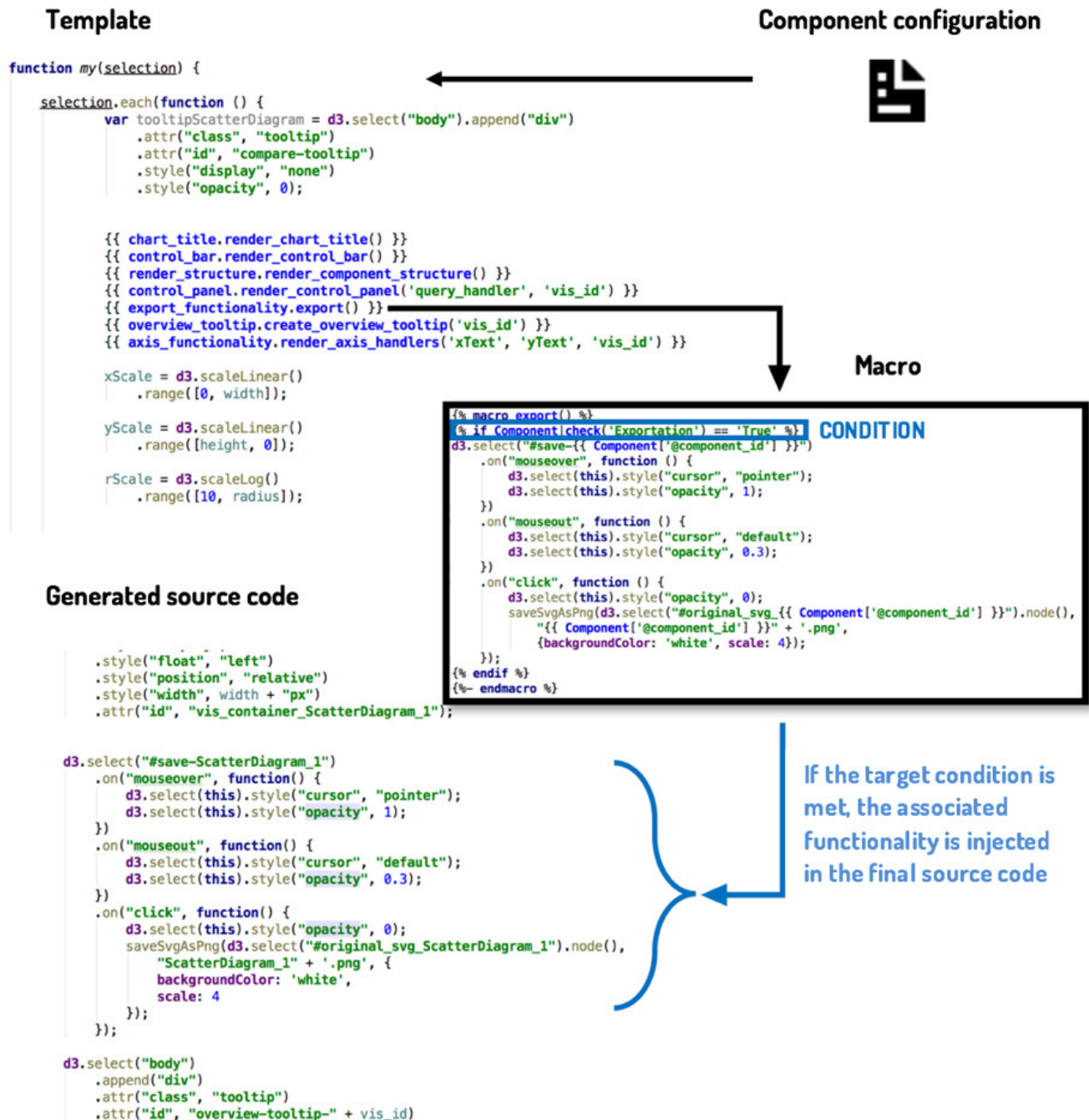



Figure 11

Results derived from the first configuration

Through this configuration is possible to apply different filters simultaneously to each scatter diagrams to observe how patterns evolve

Figure 12

Results derived from the second configuration

The scatter diagram shows the link between different students' opinions classified by gender.

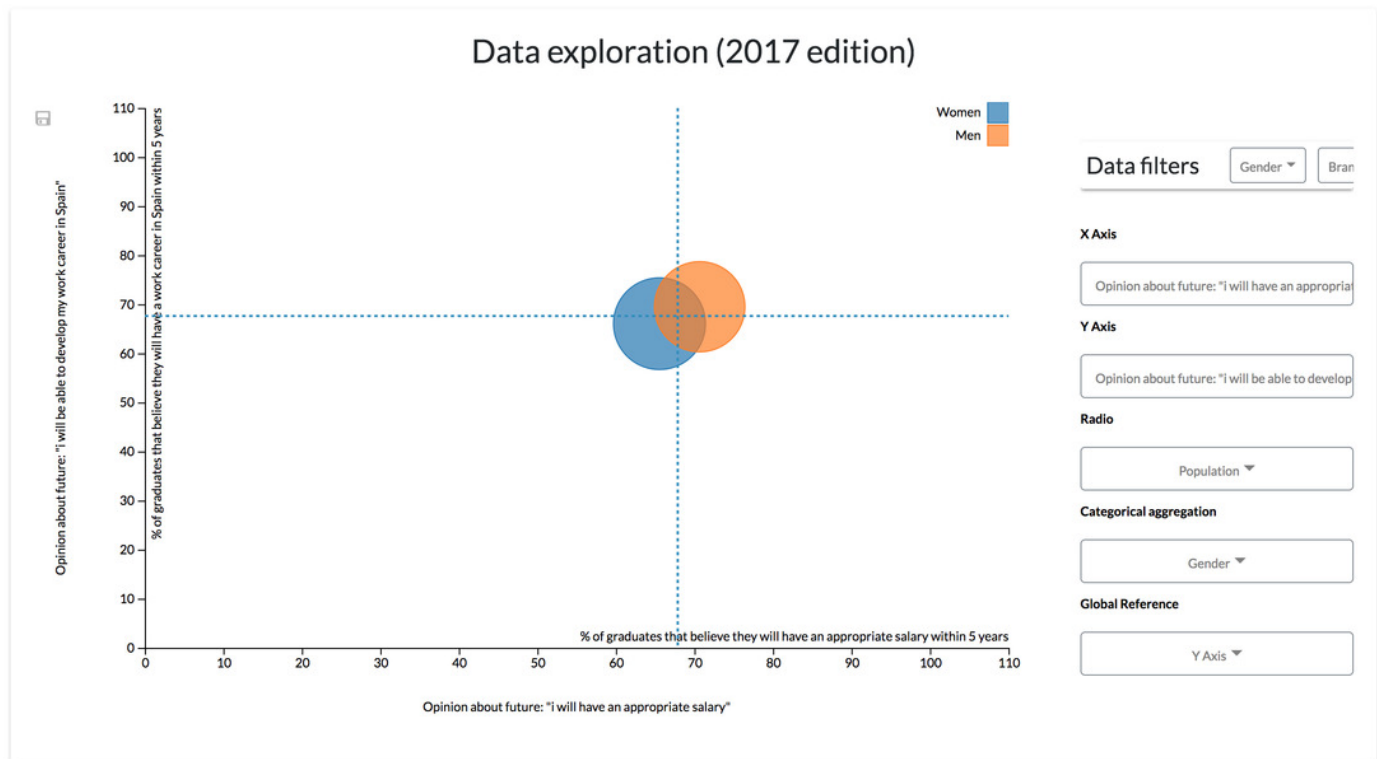



Figure 13

Results derived from the second configuration

The scatter diagram shows the link between different students' opinions classified by gender and filtered by the branch of knowledge, showing only the results related to Science students.

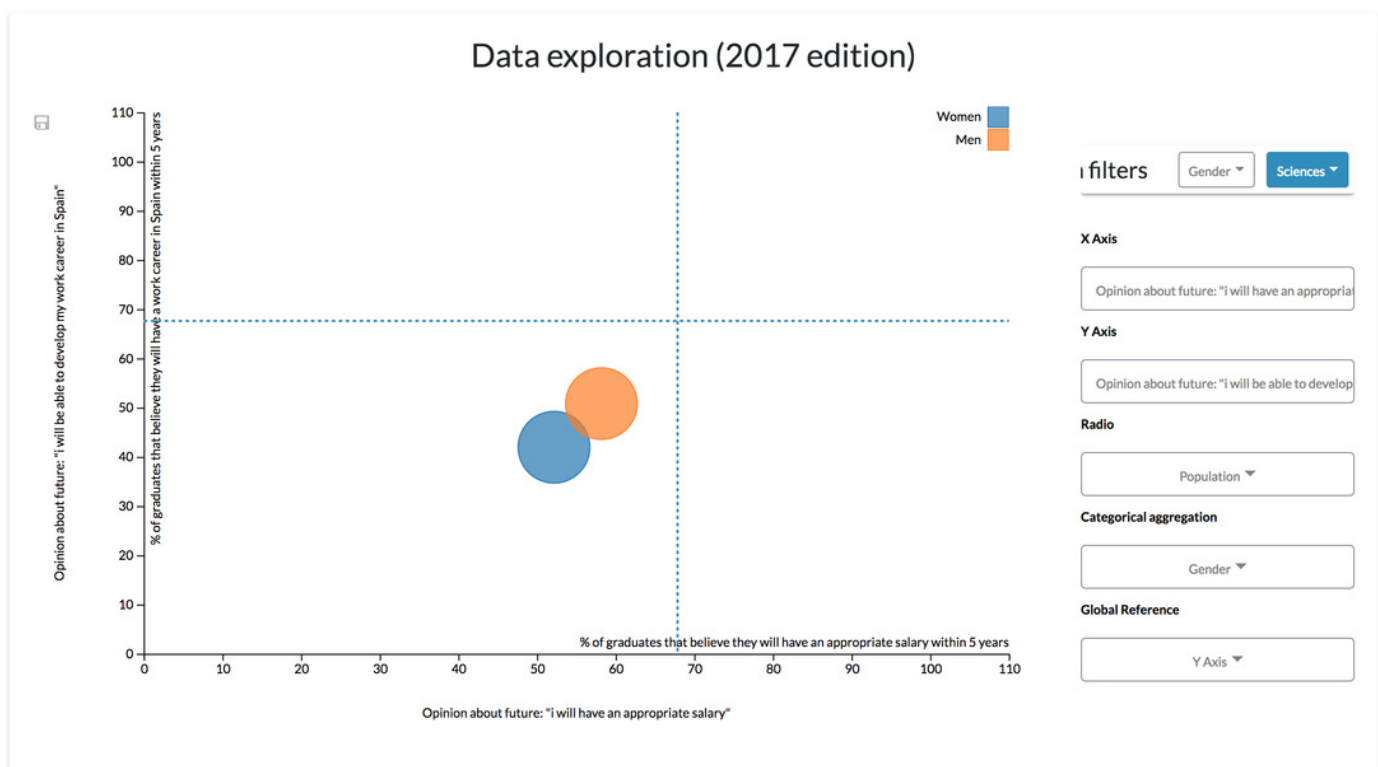



Figure 14

Modification of the second configuration to change the controls location

The controls for the scatter diagram are arranged in a bar on top of the visualization.

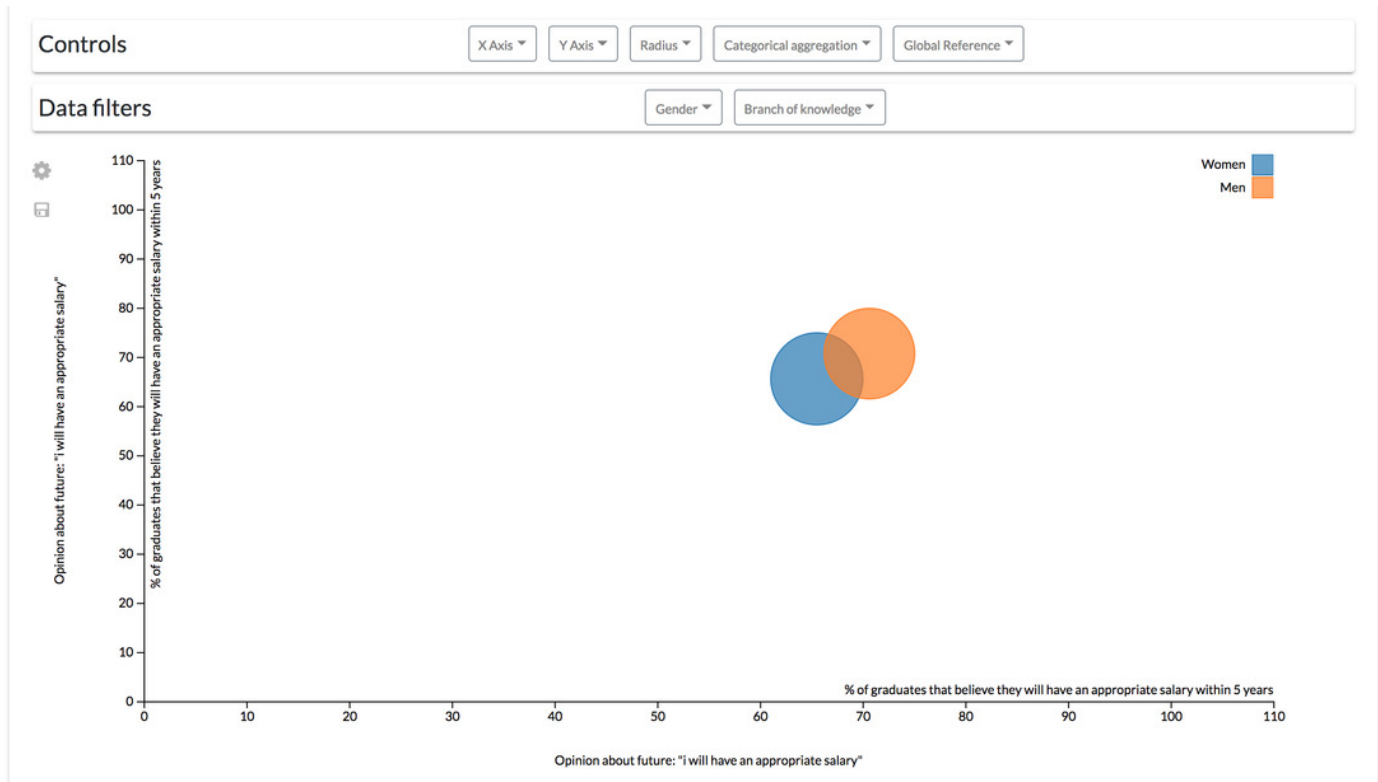
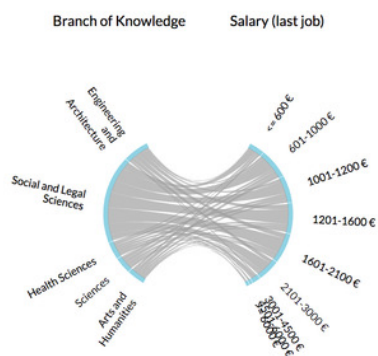
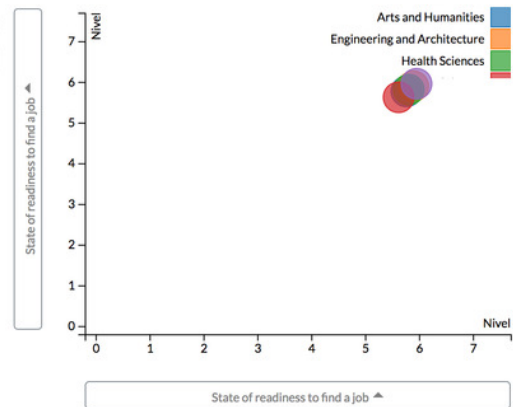



Figure 15

Dashboard involving different information visualizations

By specifying the layout of the dashboard it is possible to achieve dashboards with different components, each one with its own features.

Data Filters   Gender ▼   Branch of knowledge ▼

Categorical variables exploration

Data explorer

Skills comparison

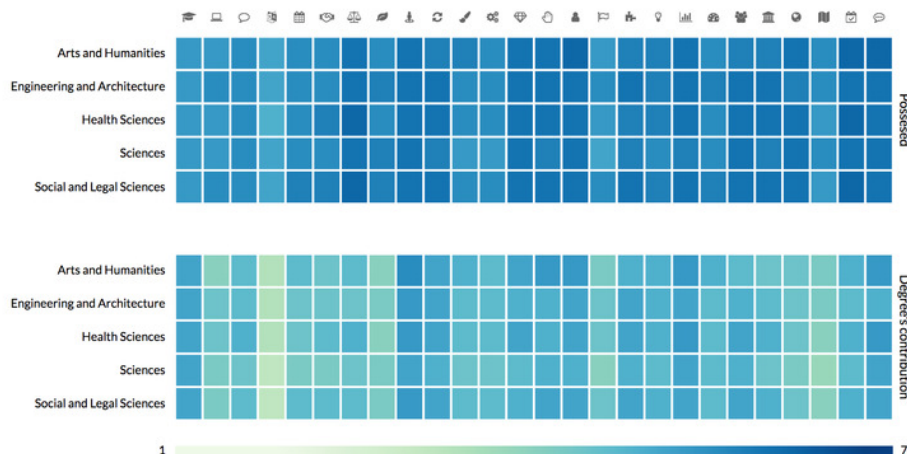



Figure 16

Possible layout configuration for comparing students' skills through different study editions

This configuration can be useful to identify lack of skills at-a-glance or their evolution through time.

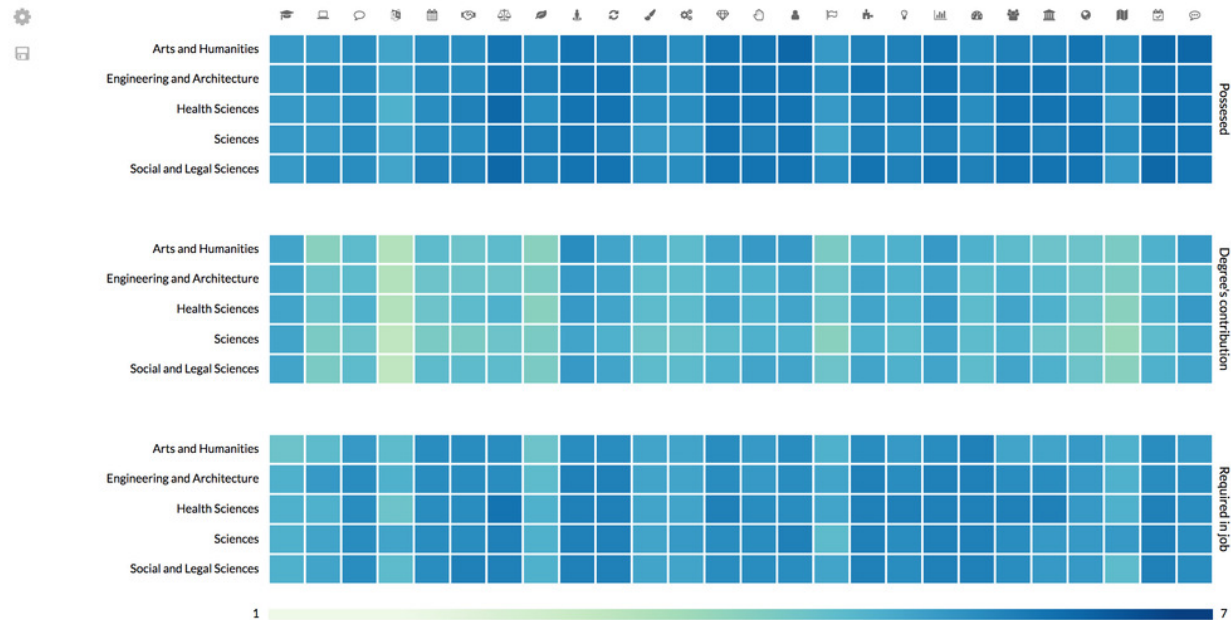
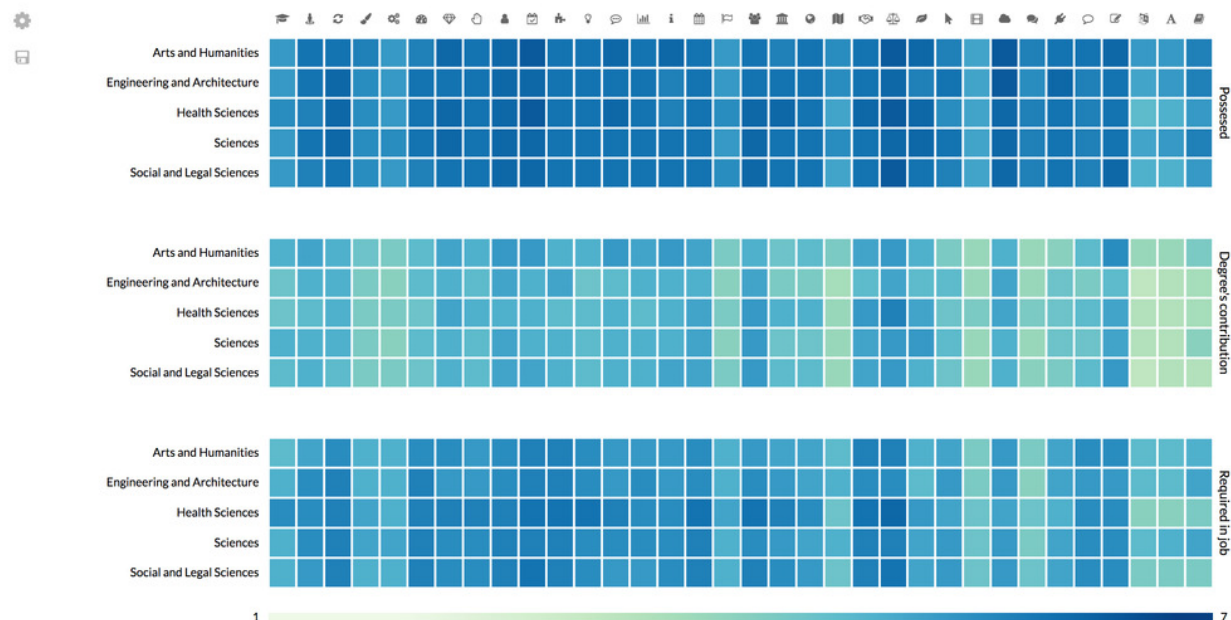



Figure 17

Simplified Gantt diagram of the SPL development

The Gantt diagram shows each task regarding the SPL development including its contextualization and design.