This is a post-print (final draft post-refeering) published in final edited form as **Vázquez-Ingelmo, A., García-Peñalvo, F.J., Therón, R., Amo Filvà, D., Fonseca, D.** Connecting domainspecific features to source code: towards the automatization of dashboard generation. Cluster Computing 23, 1803–1816 (2020). https://doi.org/10.1007/s10586-019-03012-1

Connecting domain-specific features to concrete source code: towards the automatization of dashboard generation

Andrea Vázquez-Ingelmo, Francisco J. García-Peñalvo, Roberto Therón, Daniel Amo Filvà, David Fonseca Escudero

Abstract. Dashboards are useful tools for generating knowledge and support decision-making processes, but the extended use of technologies and the increasing available data asks for user-friendly tools that allow any kind of user profile to exploit their data. Building tailored dashboards for any potential user profile would involve several resources and long development times, taking into account that dashboards can be framed in very different contexts that should be studied during the design processes to provide effective tools. This leads to the necessity of searching for methodologies that could accelerate these processes. The software product line paradigm is one recurrent method that can decrease the time-to-market of products by reusing generic core assets that can be tuned or configured to meet specific requirements. However, although this paradigm can solve issues regarding development times, the configuration of the dashboard is still a complex challenge; users' goals, datasets and context must be thoroughly studied to obtain a dashboard that fulfills the users' necessities and that fosters insight delivery. This paper outlines the benefits and a potential approach to automatically configuring information dashboards by leveraging domain commonalities and code templates. The main goal is to test the functionality of a workflow that can connect external algorithms, such as artificial intelligence algorithms, to infer dashboard features and feed a generator based on the software product line paradigm.

Keywords SPL · Domain engineering · Meta-model · Information dashboards · Feature model · Artificial Intelligence · Automatic configuration

1 Introduction

Nowadays, a lot of technological contexts ask for tailored products; new user profiles have arisen due to the extended use of technologies, and these profiles might demand distinct features in their products. The use of software is not restricted and limited to technical profiles anymore, which have expanded the variety of products that one can find to solve specific problems.

This increase in technology usage has also increase the quantity of available and generated data. People can use technology to explore information and to support their decision-making processes [1].

However, the exponential growth of data is a challenge for analyzing and generating knowledge from them, as complex patterns, for example, are less easy to detect at a glance. That is why disciplines like information visualization study how data can be encoded to foster insight delivery.

Tools like information dashboards arrange data into different views to ease the analysis of large datasets and to generate knowledge [2]. Dashboards are powerful tools, but designing and developing them are not trivial tasks; the aforementioned variety of user profiles induces the necessity of crafting dashboards according to specific user needs. In other words, dashboards need to be tailored to enhance user experience and to achieve particular user goals. But tailoring dashboards for any potential user profile would consume significant resources and the most important: significant time.

Developing a user-specific dashboard involves the analysis of the user goals and required tasks, the analysis of the user herself/himself, etc., in addition to the actual development of the dashboard, which also consumes time and resources.

Given this situation, different paradigms arose to ease the tailoring of these kind of products by reusing artifacts or by developing dashboard generators [3]: from configuration wizards that enable users to build their own dashboards, structured files that are rendered into final products, agents that personalize the tools based on the users' behavior, etc., to software engineering frameworks like model-driven engineering or the software product line paradigm.

The latter software engineering methods provide a framework to build systems based on high-level resources (meta-models, core assets, etc.) that can be instantiated into concrete products. Indeed, one of the main benefits of the Software Product Line (SPL) paradigm is the capacity of tailoring products to match particular user needs without consuming several resources and development time [4,5] by increasing reutilization not only at code-level, but also at knowledge-level.

The SPL paradigm is a powerful approach to leverage solutions framed within a specific domain [5,6]. It has been used in several contexts [7-12] and achieved significant results, decreasing development times and time-to-market. This

paradigm is organized in two phases: domain engineering and application engineering [5,6]. The domain engineering phase is focused in analyzing the target domain in which the products will be framed, obtaining a series of artifacts that will be the inputs for the application engineering phase, in which the developed models, assets, etc., are employed to build specific products according to the clients' needs.

In this paradigm, the domain engineering phase is essential. This phase thoroughly analyzes the domain to find reusability opportunities. Through the analysis and extraction of commonalities among potential products, it is possible to model abstract properties that can be instantiated and configured into concrete solutions. These properties or features are thus, essential to understand the nature of the domain's products.

There exist different methods to account for a domain's features [13]. However, one of the most used methodologies to model a SPL's features is the feature-oriented domain analysis (FODA) [14]. This method allows the representation of every possible product of a SPL in terms of a set of mandatory, optional and alternative features. The variability points (i.e., the locations in which the product line variability are injected) can be materialized through different methods [15]; selecting the right method is based on available resources, time and the product line's features characteristics, like the required granularity [16].

Although information dashboards can present different forms, they share common features (visual encodings, layouts, low-level tasks, etc.), which makes the SPL paradigm a suitable approach for building tailored dashboards. In fact, successful examples of the application this approach to information dashboards can be found in the literature [17,18].

However, although the SPL paradigm can support the generation of dashboards, the features still need to be manually selected. The configuration process can be a challenge, as users might not know what dashboard configuration better suits their needs and thus, the dashboard designer wouldn't be able select the appropriate dashboard features. In this case, the configuration process asks for automation.

How can a set of proper dashboard's features be inferred from users? This paper presents a proposal for generating dashboards employing domain engineering, and the possibilities of connecting the SPL's core assets with external prediction algorithms to obtain suitable features.

The rest of this paper is organized as follows. Section 2 outlines previous works regarding the application of artificial intelligence within the SPLs and visualization recommendation domains. Section 3 describes the methodology used in each stage of this work. Section 4 presents the proposed workflow for automatically generating dashboards, followed by section 5, in which a small proof-of-concept to test the feasibility of the approach is performed. Finally, sections 6 and 7 discuss the results and raise some conclusions about the work done, respectively.

2 Background

2.1 Variability mechanisms in software product lines

There exist different techniques to implement variability points in SPLs. It is important to choose wisely given the requirements of the product line itself (i.e. complexity of the software to develop, its number of features, their granularity requirements, etc.).

Variability points that correspond to a certain feature will be spread across different source files generally [15]. That is why separating concerns at implementation level is essential to avoid this scattering, as this feature dispersion would decrease code understandability and maintainability.

Implementing each feature in individual code modules can help with the separation of concerns [15], but it is difficult to achieve fine-grained variability through this approach. A balance between code understandability and granularity should be devised to choose both a maintainable and highly customizable SPL.

This section will briefly describe different methods that are potentially suitable to the dashboards' domain given their particular features, and that could be employed to connect the outputs yielded from an external configuration algorithm. However, there are more approaches to implement variability in SPLs that can be consulted in [15].

2.1.1 Conditional compilation

Conditional compilation uses preprocessor directives to inject or remove code fragments from the final product source code. This method allows the achievement of any level of feature granularity due to the possibility of inserting these directives at any point of the code, even at expression or function signature level [19]. Also, although pretended to the C language, preprocessor directives can be used for any language and arbitrary transformations [20]. The main drawback of this approach is the decrease of code readability and understandability as interweaving and nesting these preprocessor directives makes the code maintainability a tedious task [21].

2.1.2 Frames

Frame technology is based on entities (frames) that are assembled to compose final source code files. Frames use preprocessor-like directives to insert or replace code and to set parameters [15]. An example of a variability implementation method based on frame technology is the XML-based Variant Configuration Language (XVCL) [22]. Through this approach, only the necessary code is introduced in concrete components by specifying frames that contain the code and directives associated to different features and variants. XVCL is independent of the programming language and can handle variability at any granularity level [23].

2.1.3 Template engines

Template engines allow the parameterization and inclusion/exclusion of code fragments through different directives. If the template engine allows the definition of macros, features can be refactored into different code fragments encapsulated through these elements, improving the code organization and enabling variability at any level of granularity. Templating engines can be also language-independent, providing a powerful tool for generating any kind of source file [24] by using programming directives such as loops and conditions.

2.1.4 Aspect-Oriented Programming

Aspect-Oriented Programming (AOP) allows the implementation of crosscutting concerns through the definition of aspects, centralizing features that need to be present in different source files through unique entities (aspects) thus improving code understandability and maintainability by avoiding scattered features and "tangled" code [25].

AOP is a popular method to materialize variability points in SPLs due to the possibility of modifying the system behavior at certain points, namely join points [26-28]. However, AOP could lack of fine-grained variability (i.e. variability at sentence, expression or signature level) and particular frameworks or language extensions are necessary to implement aspects in certain programming languages.

2.2 Automatic visualization recommendations

Selecting right information visualizations is a challenging step when developing dashboards. This step is also crucial, because the effectiveness of the dashboard or visualization can be compromised if the design decisions are not based on the context or the final users.

To ease this process, several authors in the literature aim at automatically recommending visualizations. As the majority of works point out, the particular design of a visualization depends on different aspects, like the domain knowledge of the users, the tasks that the visualization must support or the data characteristics [29].

Developing a dashboard or a visualization need expert knowledge to help with design decisions based on the audience and context to which the product is aimed for. But can this developing process and design decisions be automated? There are different approaches to tackle this challenges.

For example, some methods use visual mapping and rules to recommend a certain visualization based on the target data to be displayed [29]. Through this method, the dataset is analyzed and a series of rules are applied to select a proper visual mark, scale, channel, etc. based on the type or properties of the data variables. Some tools take advantage of this method to develop suitable visualizations, like Tableau's Show Me [30], Manyeyes [31] or Voyager [32].

On the other hand, VizDeck [33] proposes a set of visualization based on the input data and the user selects the ones that prefers the most. The system learns from these user interactions, providing the user with similar visualizations subsequently.

Other solutions take into account the context in which the visualization is framed, attending to the domain, experience or knowledge. For example, in [34] a visualization ontology named VISO is employed no only to annotate data, but also "to represent context and factual knowledge". Then, a ranking process is executed to yield the recommendations using rules to rate the suitability of visual encodings.

User behavior is also considered as a crucial aspect when recommending visualizations. For example, in [35,36], the user behavior when interacting with a visualization is logged and used later to search for interesting patterns that could mean that the user asks for a different visualization. When the visual task is inferred from the user behavior, a set of ranked annotated visual metaphors is retrieved.

Furthermore, an application of content and collaborative filtering to recommend visualizations can be found in [37]. Through this methods, potentially suitable visualizations for a specific user can be identified.

Finally, applications of neural networks to infer specific features of a visualization are also present in the literature. VizML [38] used the Plotly API[1] to retrieve information about how the Plotly community crafted different visualizations for different datasets to train a set of models. The outputs of these models are a set of visualization design choices at visualization-level and encoding-level, including mark types or properties regarding axes, like if a variable is encoded on the X or the Y axis or if the axis is shared.

A similar approach is taken in Data2Vis [39], where the inputs of a sequence to sequence model are data characteristics and the output is a visualization specification in Vega-Lite.

As well as with the variability mechanisms of software product lines, it can be seen that there are many methods to address the automatic recommendation of suitable visualizations. The next section details the specific methodology followed for this work.

3 Methodology

3.1 Conceptualization: meta-modeling and domain engineering

As introduced, the employed framework involves two software engineering approaches: meta-modeling and the software product line paradigm. The combination of meta-modeling to obtain a domain abstraction with the SPL philosophy of systematically reuse assets, provides a powerful framework for building families of products.

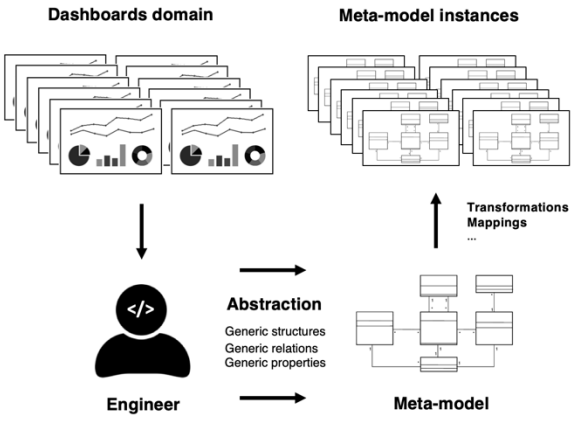
The first (and essential) phase in developing SPLs is domain engineering. This phase is conducted to understand the domain in which the potential products of the SPL are framed. But what exactly is the meaning of "understand the domain"? Using a domain engineering approach, to understand the domain means to analyze it, to find commonalities and variability points among the domain's products. By identifying commonalities and variations, generic and abstract components can be developed to subsequently combine them, parameterize them, derive them, or to execute any other operation with the goal of obtaining a concrete product [15].

However, how can these domain characteristics be captured? Meta-models can support this conceptualization process. These elements are artifacts from the model-driven architecture paradigm [40,41] that capture high-level and abstract concepts and document them in a structured representation. Through meta-modeling, the development of general rules, constraints, etc., for a set of related problems is fostered by abstracting common structures and associations found in different domain's instances.

In the end, meta-models can be mapped to concrete model instances and products, following the OMG four-layer meta-model architecture [42]: meta-meta-model layer (M3), meta-model layer (M2), user model layer (M1) and user object layer (M0).

The dashboards domain is indeed complex. There is a huge diversity of dashboards in terms of designs, interactions, supported tasks, displayed data, layouts, etc.

However, using meta-models and domain engineering as conceptualization tools, it can be possible to extract commonalities shared by dashboards and arrange them into abstract models that can be concretized. Figure 1 illustrates this approach.



[1] https://plot.ly/

Fig 1. Using meta-models to understand complex domains.

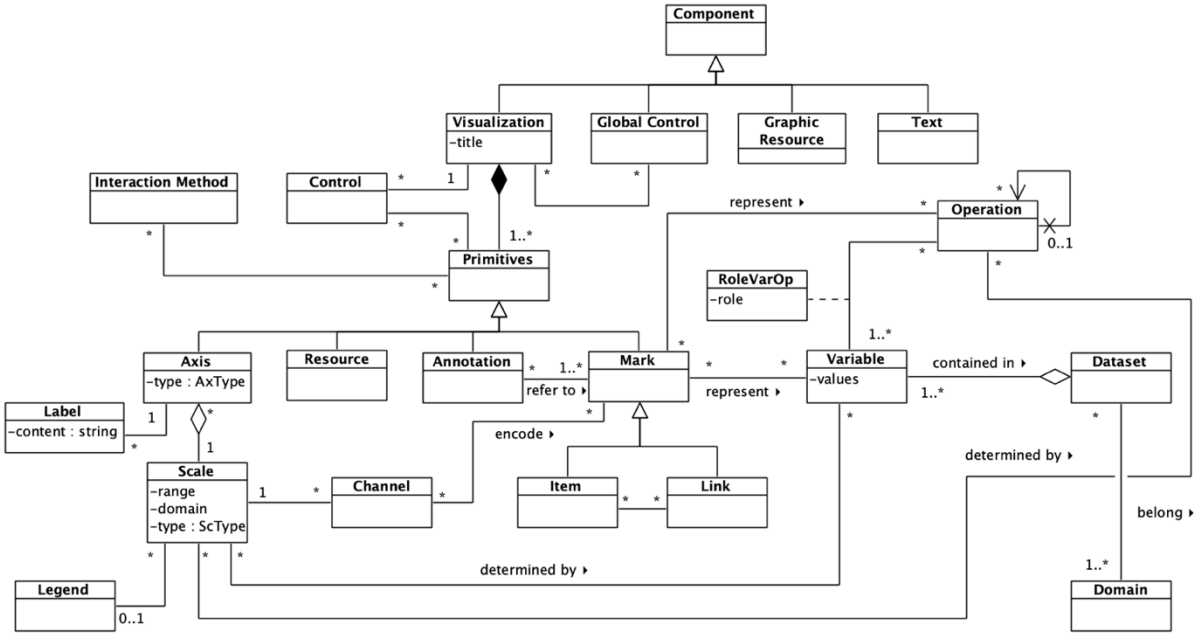


Fig 2. An extract of the dashboard meta-model where the visualization components are decomposed into abstract primitives [43].

An extract of the developed dashboard meta-model is shown in Figure 2, in which different primitives that a visualization could have are presented.

This meta-model also accounts for user goals and characteristics, including visualization literacy, domain knowledge, bias, preferences, etc. [44]. Also, components are arranged among different containers to compose the dashboard. The full version of the meta-model can be consulted at [43].

The meta-model helps with the conceptualization of the automatic generation approach and to gain knowledge about abstract features regarding dashboards.

However, to test this first approximation and the configuration process of dashboards, an abbreviated feature model has been developed, which can be consulted in Figure 3.

This feature model reflects the elements that a dashboard display belonging to the product line could have at the moment. In this case, a dashboard display can be composed of different pages with different visualizations.

As presented in the meta-model, a visualization is composed by marks that could encode data through different channels.

The “Base Structure” feature is the core of every visualization, which supports its generation and its basic functionalities.

Of course, this feature model can be extended to support more characteristics (scale types, annotations, interaction patterns, etc.), but in order to test the feasibility of this approach in a straightforward way, the feature model has been kept simple.

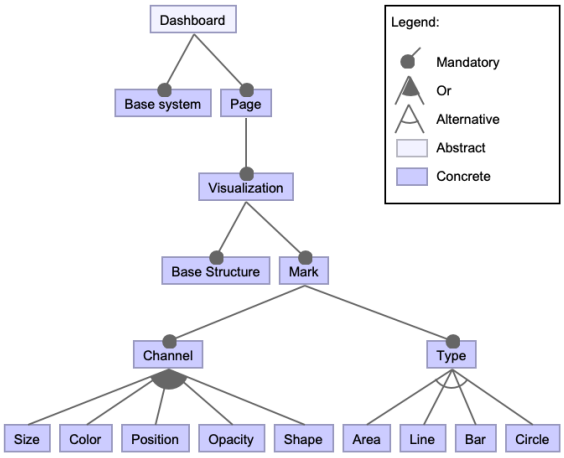


Fig 3. The dashboard product line's feature model.

3.2 Variability implementation: code templates

One of the challenges regarding the development and configuration of SPLs is the desired features' granularity. In the dashboards' domain, features' are fine-grained; variability is focused on visual and interactive elements of the presentation layer, because slight modifications on different aspects, like interaction patterns, layouts, color palettes, etc., can influence the perceived usability and user experience [45]. In other words, dashboards' features can involve statement-level, and even expression-level granularity [16], meaning that tiny fragments of code can be affected by the SPL configuration process.

Given these granularity requirements, the chosen mechanism to implement the SPL is based on template engines. These tools allow developers to tag sections and to parameterize fragments of source code to subsequently instantiate them with particular values, thus obtaining concrete source files [17].

Jinja2 has been selected as the template engine given its powerful features like the possibility of enclosing code within macros and the multiple directives available (loops, conditional directives, variable definitions, etc.). On the other hand, the chosen technology to implement the SPL's assets (i.e., the code templates' content) is the data visualization framework Vega-Lite[2]. This is not an arbitrary selection, Vega-Lite allows building custom visualizations through a fine-grained configuration of their properties, which matches the requirements of this product line approach and fits with the identified domain's features.

3.3 Configuration proposal: artificial intelligence and visual mapping

As presented in section 2.2, artificial intelligence has arisen as a powerful tool to recommend visualization types and configurations. The fact that users and developers might be biased when designing a dashboard makes artificial intelligence algorithms a useful means to drive design decisions.

Although creating a product line of dashboards can boost development processes, products still need to be configured. And configuration processes are still manual processes in which user requirements have to be analyzed and materialized by selecting the appropriate features of the product line to obtain a suitable display. This is a complex and crucial process that can compromise the effectiveness of the dashboard.

[2] https://vega.github.io/vega-lite/

However, the configuration of a product line with several features involved can be overwhelming, even for an artificial intelligence algorithm. As previously introduced (in section 3.1) the dashboard meta-model contemplates several fine-grained features; inferring every single exact value for every single feature might involve significant quantities of data and resources to get good results.

That is why using visual mapping approaches are also useful for these problems, because general guidelines must be followed to deliver correct visualizations. In this case, a visualization recommendation engine, such as CompassQL [46,47], can be selected to act like a visual mapping engine.

Specifically, CompassQL allows partial specifications of visualizations, and integrates methods for choosing and ranking the best specification.

The following section explains how artificial intelligence and visual mapping can be introduced in a dashboard product line configuration process.

4 Proposed workflow

This section proposes a workflow to address the challenge of automatically configuring a product line of dashboards based on user necessities.

Of course, the first step is to identify the inputs and outputs of the whole process. Broadly speaking, there are three main inputs based on the previously presented meta-model in section 3.1: the user's goals, the user's characteristics and the target datasets. Following previous visualization recommendation methods [29], this method combines different aspects to obtain a hybrid approach.

The reason to use a hybrid approach is because using a single dimension for configuring dashboards or visualizations, for example using only data characteristics or using only user preferences, can lack effectiveness. Data characteristics need to be taken into account to build appropriate charts [48], as some visual encodings are not suitable for some type of variables.

However, while a chart can be correctly built it can be inappropriate if its context and audience is not considered. User goals are crucial, as visual metaphors, supported tasks or displayed data depend on what are the user's data necessities, as well as users' characteristics, because the visualization literacy, domain knowledge or even bias, can compromise the users' insight delivery process.

On the other hand, in this case, the selected method to manage tailoring capabilities is the software product line paradigm, meaning that the output of the configuration process must be a set of suitable features from the feature model. Figure 4 outlines the inputs and outputs of the process.

In the end, the problem is summarized as the necessity of tuning the core assets parameters to optimize the user experience and the effectiveness of the generated dashboard. To automatize the whole process, users' goals, characteristics and datasets must be structured into machine-readable documents to allow their processing. The output as well, in this case a selection of features, can be transformed into a structured file readable by the code generator which will inject the specific parameter values into the templates to generate the final dashboard.

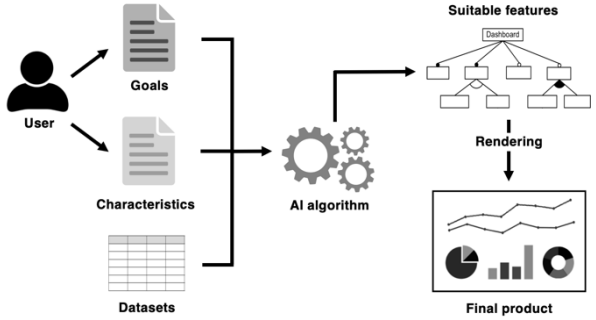


Fig 4. Inputs and outputs of the proposed dashboards' configuration approach.

However, as previously shown in Figure 2, a single visualization needs several primitives, properties and features to be selected. Identifying and configuring every potential feature would consume several resources and time. This also means that, to train a model, it is necessary to have not only relevant quantities of data, but also information about different types of visualizations.

That is why the following configuration process is proposed. Using user data and data schemes, some basic but relevant features are predicted, for example: predict the number of visualizations that a dashboard should hold given all of the users' goals and expertise. This could be also applied at "visualization-level", for example: predict the appropriate number of encodings/channels for a visualization or the mark type.

Then, once this information has been collected, the inferred data can be converted into a set of concrete visualizations by using visual mapping.

In this case, to test functionality, CompassQL has been selected as the visual mapping/recommendation engine to deliver the final dashboard. CompassQL enables the automatic selection of channels given the dataset and the variables to visualize. The structure of this query language allows the connection of the outcomes from an external algorithm to a visualization query through the mentioned code templates, and the mapping results can also be used to generate visualizations with different frameworks.

5 Proof-of-concept

This section outlines the functionality of the product line generation process.

The code templates hold basic and generic code for every visualization, and they can be parameterized to inject concrete features. Figure 5 shows a code template snippet, where external data is used to parameterize the final code after the rendering process


```
{% for visualization in Dashboard.Visualizations %}
var query{{ visualization.id }} = {
  "spec": {
    "data": {"url": '{{ Dashboard.Data }}'},
    "mark": "{{visualization.mark}}",
    "encodings": [
    {% for channel in visualization.channels %}
      {
        "channel": "?",
        "aggregate": "?",
        "field": "{{channel.field}}",
        "type": "?"
      },
    {% endfor %}
    ]
  },
  "chooseBy": "effectiveness"
};
```

Fig 5. Code snippet for a code template in which a CompassQL query is instantiated using external parameters.

This means that, if the outcome of a previous prediction process using users' characteristics is as in Figure 6, the delivered dashboard could be configured as in Figure 7.

```
{"Dashboard": {
    "Data": "node_modules/vega-datasets/data/cars.json",
    "Visualizations": [{
        "id": 1, "mark": "bar",
        "channels": [{"field": "Cylinders"}, {"field": "Acceleration"}]
    },
    {
        "id": 2, "mark": "circle",
        "channels": [{"field": "Horsepower"}, {"field": "Acceleration"},
                     {"field": "Cylinders"}]
    }]
}}
```

Fig 6. Example dashboard configuration.

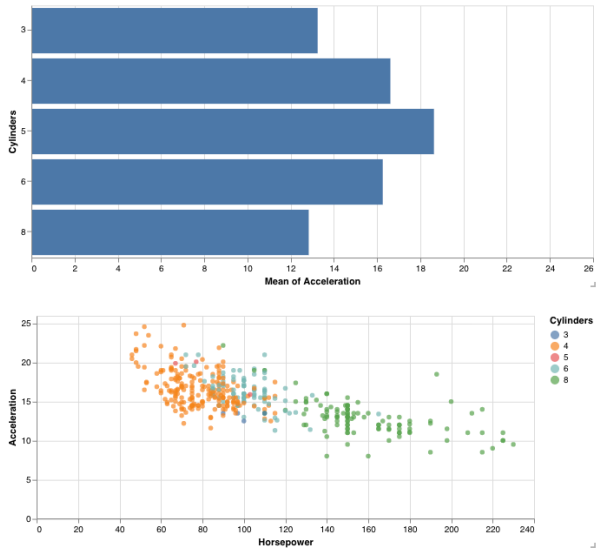


Fig 7. Example outcome of the generation process.

In this case, the visualizations are generated through Vega-Lite, but the CompassQL recommendation outcome could be employed with other visualization frameworks due to the flexibility and language independence of the code templates.

For example, the previous visualizations can be generated with the visualization framework Semiotic [53], using the CompassQL outcomes after the proper channels had been inferred through visual mapping. Figure 8 shows a code template for generating XY charts, while Figure 9 shows the final generated visualization, which coincides with the scatter plot generated at the beginning of this section.

```
{% for visualization in Dashboard.Visualizations %}
  {% if visualization.type == "xy" %}
    var props{{ visualization.id }} = {
      "data": '{{ Dashboard.Data }}',
      "type": "{{visualization.mark}}",
      {% for encoding in visualization.encodings %}
        {% if encoding.channel == "x" %}
          xAccessor: "{{encoding.field}}",
        {% elif encoding.channel == "y" %}
          yAccessor: "{{encoding.field}}",
        {% elif encoding.channel == "color" %}
          pointStyle: d => {
            return {
                fill: colorScale(d.{{encoding.field}})
                }
          },
        {% endif %}
      {% endfor %}
```

Fig 8. Using the visualization framework Semiotic to generate charts using properties from external files.

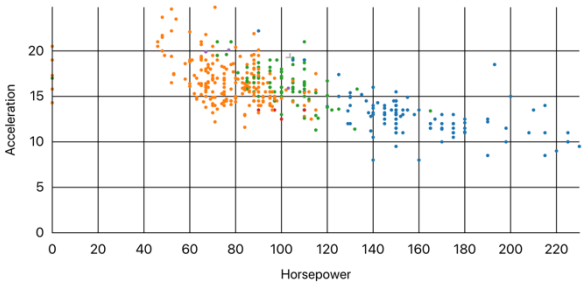


Fig 9. Generated scatter plot using Semiotic's code.

In addition, code templates can be used for other features of the software product line, like the addition of interaction patterns, controls, styles, etc., following the same approach as in [17].

6 Discussion

This paper proposes an approach that takes advantage of domain engineering and code templates for generating tailored dashboards.

The feasibility of connecting external algorithms' outcomes to a software product line's configuration process using code templates has been tested.

Choosing the right implementation technique is a complex task, because several factors must be taken into account: the aforementioned granularity level, the understandability and maintainability of the code, the viability of the technique, etc. Having the power of customizing their features at fine-grained level could be highly valuable, as dashboards usually ask to be user-tailored in order to provide useful support for particular and individual goals.

Code templates not only enable fine-grained variability points to be materialized at code-level, but also allows the introduction of external algorithms, like artificial intelligence or visual mapping algorithms within the rendering process. The parameterization of code through Jinja2 templates lets the generator to be fed with configuration files no matter their source, if provided with a fixed syntax.

Fine-grained features are very important in a domain like this, changing the mark type or some encoding channel could be decisive for users to accomplish their goals regarding datasets. Code templates allow flexibility and even expression-level variability, which are powerful features for this domain.

Although this approach still lacks of powerful maintainability levels, it maintains a good requirements traceability by arranging features in a variety of macro definitions. Using XVCL [22] could have been another solution to manage these fine-grained features, but the decision of wrapping the SPL specification through a DSL asked for a more flexible and customizable method such as a template engine.

What is more, a combination of the AOP paradigm with the templating method could be highly beneficial providing both customization regarding directives and a better technique to manage crosscutting concerns (an issue that a template engine could not solve straightforwardly).

However, although presenting some caveats, the results are promising and prove that a powerful template engine could be a beneficial method to materialize fine-grained variability and to introduce external configuration algorithms within the SPL paradigm context.

Regarding the introduction of artificial intelligence, a hybrid approach for configuring the product line is proposed. The reason behind this approach is that, while artificial intelligence is powerful for predicting values given an input set of features, visual mapping ensures that basic guidelines and constraints are followed.

Artificial intelligence could be useful for identifying user profiles and to "translate" these profiles into a set of visualization features. But, in the end, a series of constraints regarding the construction of graphics and visualizations must be followed. That is why this work proposes predicting high-level visualization features, like the number of channels or visualizations within a dashboard, and then using visual mapping to convert this abstract information into concrete visualizations.

Moreover, a meta-model has been developed to extract commonalities from the domain and to understand the relationships among the identified elements. The user has been included in the meta-model as they are the final consumers of the dashboards. Knowing and understanding their characteristics, backgrounds, preferences, expertise, goals, etc., is crucial to deliver dashboards that support their intentions in an effective manner.

However, there are also challenges regarding the introduction of user characteristics and goals in this kind of AI pipelines. On the one hand, user goals are identified and expressed in natural language, meaning that they need preprocessing before using them to predict values. However, structuring goals is not trivial. There exist different frameworks for identifying and linking goals with low-level tasks that could be very useful for this matter [49-51]. On the other hand, user characteristics like expertise, visualization literacy, bias, are not only difficult to structure but to identify and measure, although there are also frameworks that could help with these tasks [52-54].

Another challenge is that this approach needs to be tested with real-world data and seek for improvements, but the functionality of the workflow seems promising.

However, not only tests with real-world data need to be performed. Users are the backbone and the target of visualization tools. Visualization recommendation approaches aim at boosting their capacity to reach insights. Delivering a functional dashboard is important, but in the end, the metrics that prove that the approach is useful are user-related (usability, engagement, acceptance, insights reached, etc.).

Testing an automatically generated product can refine not only the product itself, but also the whole generation process by obtaining more feedback data that could feed AI algorithms.

7 Conclusions

This paper describes a proof-of-concept for a dashboard generator that uses users and datasets characteristics.

The functionality of the template-based generator has been tested to prove the feasibility of the approach and the necessity of take into account fine-grained features in a complex domain like dashboards'. The configuration process proposal takes into account the user goals and characteristics and not only the dataset schema to infer high-level visualization features.

Future research lines will involve the testing of this approach with real-world data and users, and the iterative improvement of the workflow, to predict more visualizations' features, and thus obtaining a more powerful tailoring process for dashboards.

Acknowledgements. This research work has been supported by the Spanish *Ministry of Education and Vocational Training* under an FPU fellowship (FPU17/03276). This work has been partially funded by the Spanish Government Ministry of Economy and Competitiveness throughout the DEFINES project (Ref. TIN2016-80172-R), the PROVIDEDH project, funded within the CHIST-ERA Programme under the national grant agreement: PCIN-2017-064 (MINECO, Spain) and the Ministry of Education of the Junta de Castilla y León (Spain) throughout the T-CUIDA project (Ref. SA061P17).

References

1. Albright, S.C., Winston, W., Zappe, C.: Data analysis and decision making. Cengage Learning, Mason, OH, USA (2010)
2. Few, S.: Information dashboard design. O'Reilly Media, Sebastopol, CA, USA (2006)
3. Vázquez-Ingelmo, A., Garcia-Peñalvo, F.J., Therón, R.: Information Dashboards and Tailoring Capabilities - A Systematic Literature Review. IEEE Access **7**, 109673-109688 (2019). doi:10.1109/ACCESS.2019.2933472
4. Clements, P., Northrop, L.: Software product lines. Addison-Wesley, Boston, MA, USA (2002)
5. Pohl, K., Böckle, G., Van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag New York, Inc., New York, NY, USA (2005)
6. Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison Wesley Longman Publishing Co., Inc., Boston, MA, USA (2004)
7. Kramer, D., Oussena, S., Komisarczuk, P., Clark, T.: Using document-oriented GUIs in dynamic software product lines. In: ACM SIGPLAN Notices, New York, NY, USA 2013, vol. 3, pp. 85-94. ACM
8. Marinho, F.G., Lima, F., Ferreira Filho, J.B., Rocha, L., Maia, M.E., de Aguiar, S.B., Dantas, V.L., Viana, W., Andrade, R.M., Teixeira, E.: A software product line for the mobile and context-aware applications domain. In: International Conference on Software Product Lines 2010, pp. 346-360. Springer
9. Ezzat Labib Awad, A.: Enforcing Customization in e-Learning Systems: an ontology and product line-based approach. In. Universitat Politècnica de València, Valencia, Spain, (2017)

10. Gabillon, Y., Biri, N., Otjacques, B.: Designing an adaptive user interface according to software product line engineering. Paper presented at the The Eighth International Conference on Advances in Computer-Human Interactions, Lisbon, Portugal,
11. Pleuss, A., Hauptmann, B., Keunecke, M., Botterweck, G.: 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
12. Quinton, C., Mosser, S., Parra, C., Duchien, L.: Using multiple feature models to design applications for mobile phones. In: Proceedings of the 15th International Software Product Line Conference, Volume 2 2011, p. 23. ACM
13. Achtaich, A., Ounsa, R., Nissrine, S., Camille, S.: Selecting SPL modeling languages: a practical guide. In: 2015 Third World Conference on Complex Systems (WCCS) 2015, pp. 1-6. IEEE
14. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. In. Carnegie-Mellon University, Software Engineering Institute, Pittsburgh, PA, USA, (1990)
15. Gacek, C., Anastasopoules, M.: Implementing product line variabilities. In: ACM SIGSOFT Software Engineering Notes, New York, NY, USA 2001, vol. 3, pp. 109-117. ACM
16. Kästner, C., Apel, S., Kuhlemann, M.: Granularity in software product lines. In: Proceedings of the 30th international conference on Software engineering 2008, pp. 311-320. ACM
17. Vázquez-Ingelmo, A., García-Peñalvo, F.J., Therón, R.: Taking advantage of the software product line paradigm to generate customized user interfaces for decision-making processes: a case study on university employability. PeerJ Computer Science **5**, e203 (2019). doi:10.7717/peerj-cs.203
18. Logre, I., Mosser, S., Collet, P., Riveill, M.: Sensor data visualisation: a composition-based approach to support domain variability. In: European Conference on Modelling Foundations and Applications 2014, pp. 101-116. Springer
19. Liebig, J., Apel, S., Lengauer, C., Kästner, C., Schulze, M.: An analysis of the variability in forty preprocessor-based software product lines. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1 2010, pp. 105-114. ACM
20. Favre, J.-M.: Preprocessors from an abstract point of view. In: Reverse Engineering, 1996., Proceedings of the Third Working Conference on 1996, pp. 287-296. IEEE
21. Spencer, H., Collyer, G.: #ifdef considered harmful, or portability experience with C News. (1992).
22. Jarzabek, S., Bassett, P., Zhang, H., Zhang, W.: XVCL: XML-based variant configuration language. In: Proceedings of the 25th International Conference on Software Engineering 2003, pp. 810-811. IEEE Computer Society
23. Zhang, H., Jarzabek, S., Swe, S.M.: XVCL approach to separating concerns in product family assets. In: International Symposium on Generative and Component-Based Software Engineering, Efurt, Germany 2001, pp. 36-47. Springer
24. Clark, S.: Render your first network configuration template using Python and Jinja2. https://blogs.cisco.com/developer/network-configuration-template (2018).
25. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., Irwin, J.: Aspect-oriented programming. In: European conference on object-oriented programming 1997, pp. 220-242. Springer
26. Waku, G.M., Rubira, C.M., Tizzei, L.P.: A case study using aop and components to build software product lines in android platform. In: Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on 2015, pp. 418-421. IEEE
27. Heo, S.-h., Choi, E.M.: Representation of variability in software product line using aspect-oriented programming. In: Software Engineering Research, Management and Applications, 2006. Fourth International Conference on 2006, pp. 66-73. IEEE
28. Voelter, M., Groher, I.: Product line implementation using aspect-oriented and model-driven software development. In: Software Product Line Conference, 2007. SPLC 2007. 11th International 2007, pp. 233-242. IEEE
29. Kaur, P., Owonibi, M.: A Review on Visualization Recommendation Strategies. In: 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017) 2017, pp. 266-273. SCITEPRESS
30. Mackinlay, J., Hanrahan, P., Stolte, C.: Show me: Automatic presentation for visual analysis. IEEE transactions on visualization computer graphics **13**(6) (2007).
31. Viegas, F.B., Wattenberg, M., Van Ham, F., Kriss, J., McKeon, M.: Manyeyes: a site for visualization at internet scale. IEEE transactions on visualization and computer graphics **13**(6), 1121-1128 (2007).
32. Wongsuphasawat, K., Moritz, D., Anand, A., Mackinlay, J., Howe, B., Heer, J.: Voyager: Exploratory analysis via faceted browsing of visualization recommendations. IEEE transactions on visualization and computer graphics **22**(1), 649-658 (2015).
33. Key, A., Howe, B., Perry, D., Aragon, C.: Vizdeck: self-organizing dashboards for visual analytics. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data 2012, pp. 681-684. ACM
34. Voigt, M., Pietschmann, S., Grammel, L., Meißner, K.: Context-aware recommendation of visualization components. In: The Fourth International Conference on Information, Process, and Knowledge Management (eKNOW) 2012, pp. 101-109. Citeseer
35. Gotz, D., Zhou, M.X.: Characterizing users' visual analytic activity for insight provenance. Information Visualization **8**(1), 42-55 (2009).
36. Gotz, D., Wen, Z.: Behavior-driven visualization recommendation. In: Proceedings of the 14th international conference on Intelligent user interfaces 2009, pp. 315-324. ACM
37. Mutlu, B., Veas, E., Trattner, C.: Vizrec: Recommending personalized visualizations. ACM Transactions on Interactive Intelligent Systems **6**(4), 31 (2016).
38. Hu, K., Bakker, M.A., Li, S., Kraska, T., Hidalgo, C.: VizML: A Machine Learning Approach to Visualization Recommendation. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems 2019, p. 128. ACM
39. Dibia, V., Demiralp, Ç.: Data2Vis: Automatic generation of data visualizations using sequence to sequence recurrent neural networks. IEEE computer graphics and applications (2019).

40. Pleuss, A., Wollny, S., Botterweck, G.: 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
41. Kleppe, A.G., Warmer, J., Bast, W.: MDA Explained. The Model Driven Architecture: Practice and Promise. In. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, (2003)
42. Álvarez, J.M., Evans, A., Sammut, P.: Mapping between levels in the metamodel architecture. In: International Conference on the Unified Modeling Language 2001, pp. 34-46. Springer
43. Vázquez-Ingelmo, A., García-Peñalvo, F.J., Therón, R.: Capturing high-level requirements of information dashboards' components through meta-modeling. Paper presented at the 7th International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM 2019), León, Spain,
44. Vázquez Ingelmo, A., García-Peñalvo, F.J., Therón, R., Conde González, M.Á.: Extending a dashboard meta-model to account for users' characteristics and goals for enhancing personalization. Paper presented at the Learning Analytics Summer Institute (LASI) Spain 2019, Vigo, Spain,
45. Vázquez-Ingelmo, A., García-Peñalvo, F.J., Therón, R.: Addressing Fine-Grained Variability in User-Centered Software Product Lines: A Case Study on Dashboards. In: World Conference on Information Systems and Technologies 2019, pp. 855-864. Springer
46. Wongsuphasawat, K., Qu, Z., Moritz, D., Chang, R., Ouk, F., Anand, A., Mackinlay, J., Howe, B., Heer, J.: Voyager 2: Augmenting visual analysis with partial view specifications. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems 2017, pp. 2648-2659. ACM
47. Wongsuphasawat, K., Moritz, D., Anand, A., Mackinlay, J., Howe, B., Heer, J.: Towards a general-purpose query language for visualization recommendation. In: Proceedings of the Workshop on Human-In-the-Loop Data Analytics 2016, p. 4. ACM
48. Berinato, S.: Good charts: The HBR guide to making smarter, more persuasive data visualizations. Harvard Business Review Press, Brighton, MA, US (2016)
49. Lam, H., Tory, M., Munzner, T.: Bridging from goals to tasks with design study analysis reports. IEEE transactions on visualization and computer graphics **24**(1), 435-445 (2017).
50. Munzner, T.: A nested process model for visualization design and validation. IEEE Transactions on Visualization Computer Graphics(6), 921-928 (2009).
51. Brehmer, M., Munzner, T.: A multi-level typology of abstract visualization tasks. IEEE transactions on visualization and computer graphics **19**(12), 2376-2385 (2013).
52. Lee, S., Kim, S.-H., Kwon, B.C.: Vlat: Development of a visualization literacy assessment test. IEEE transactions on visualization and computer graphics **23**(1), 551-560 (2017).
53. Boy, J., Rensink, R.A., Bertini, E., Fekete, J.-D.: A principled way of assessing visualization literacy. IEEE transactions on visualization and computer graphics **20**(12), 1963-1972 (2014).
54. Bedek, M.A., Nussbaumer, A., Huszar, L., Albert, D.: Methods for Discovering Cognitive Biases in a Visual Analytics Environment. In: Cognitive Biases in Visualizations. pp. 61-73. Springer, (2018)