spl-js-engine: a JavaScript tool to implement Software Product Lines

Alejandro Cortiñas[*]
Universidade da Coruña
Centro de Investigación CITIC
Laboratorio de Bases de Datos
A Coruña, Spain
alejandro.cortinas@udc.es

Miguel R. Luaces[*]
Universidade da Coruña
Centro de Investigación CITIC
Laboratorio de Bases de Datos
A Coruña, Spain
luaces@udc.es

Óscar Pedreira[*]
Universidade da Coruña
Centro de Investigación CITIC
Laboratorio de Bases de Datos
A Coruña, Spain
oscar.pedreira@udc.es

ABSTRACT

In 2015, our research laboratory started the definition and implementation of a Software Product Line (SPL) for the generation of web-based Geographic Information Systems. Tooling support for SPL was scarce, and we did not found any suitable alternative to implement the mentioned product line. Therefore, we built *spl-js-engine*, a JavaScript library that, following the annotative approach, can generate final product source code from the annotated code, the feature model of the product line, and a product specification. *spl-js-engine* validates the specification of the product against the feature model prior to the generation. Since its first implementation, we have used this tool in many occasions both in the academia and the industry contexts.

CCS CONCEPTS

• **Software and its engineering → Software product lines**.

KEYWORDS

Software product lines, tool, JavaScript

ACM Reference Format:
Alejandro Cortiñas, Miguel R. Luaces, and Óscar Pedreira. 2022. *spl-js-engine*: a JavaScript tool to implement Software Product Lines. In *26th ACM International Systems and Software Product Line Conference - Volume B (SPLC '22), September 12–16, 2022, Graz, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3503229.3547035

1 INTRODUCTION

In this paper we present *spl-js-engine*, a tool for the implementation of software product lines based on JavaScript. The motivation for the development of this tool emerged from our own experience in technology-transfer and industrial projects.

In the Databases Laboratory, we have been working since 2015 in the field of Software Product Lines (SPL), both from the academic and the industrial perspectives. At that time we started two projects: LPS-Bigger, a technology transfer project carried out with large companies in Spain that included, among other things, developing a SPL able to generate products with functionalities related to Big Data; and GISBuilder, a research project to design and develop a SPL to generate web-based Geographic Information Systems (GIS) [3, 4]. The development of these two projects presented some constraints regarding the approach to follow, briefly described next:

- Source code is multi-language, like all current web development. Specifically, our technological stack includes Java, JavaScript, HTML, CSS, property files, yaml, SQL, and XML.
- The variability of these projects requires fine granularity changes. For example, we need to annotate Java Spring annotations, or the number of attributes of an element in a HTML template.
- Each product of GISBuilder has its own data model. To handle this, the tool should support model transformation to generate source code based on the specification of the data model of a product.
- The source code of the generated products needs to be independent of the platform; this is, it should be possible for a development team to continue developing features on top of the generated products using standard tools. This leads to another requirement: the code needs to be as readable as possible.

When we started the development of these SPLs, we evaluated many tools, which included: (i) Compositional-based tools: AHEAD, FeatureHouse, XAK. (ii) Annotative-based tools: Antenna, Munge, CPP, GPP, FeatureJS, CIDE. (iii) Other techniques: AspectJ, DeltaJ. From this set, only the general preprocessors such as GPP could be applied to our context, but their usage is not exempt of complexity, specially handling model transformations. Therefore, we decided to develop our own tool, following the annotative approach, which supports the configuration and assemble of any source-code text-file, independently of its language. *spl-js-engine* is implemented and based in JavaScript, so the annotations in the source-code files of the core components are written in this language, which has the advantage of flattening the learning curve for any web developer. Finally, we have created several types of annotations to allow interpolating data, that combined with the powerful JavaScript language allow the implementation of model transformations.

In this short paper we present `spl-js-engine`, which is available under the MIT license on GitHub[1]. Also, the tool is published in the

[*]All authors have contributed equally and the names are listed in alphabetical order

[1]Public repository of the tool (mirror of the private repository where its development is managed): https://github.com/AlexCortinas/spl-js-engine

SPLC '22, September 12–16, 2022, Graz, Austria
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9206-8/22/09…$15.00
https://doi.org/10.1145/3503229.3547035

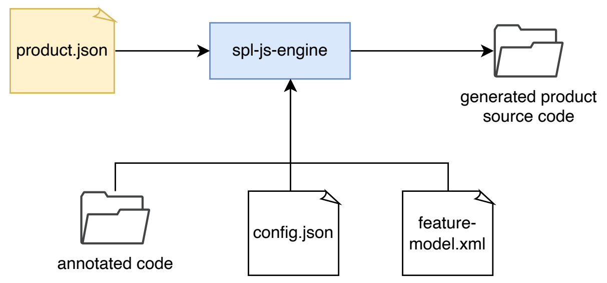

Figure 1: *spl-js-engine* usage: input files

npm repository[2], so anyone can easily install it locally, or integrate it with any NodeJS project.

The article is organized as follows: Section 2 shows how the tool is used, describes the features of the tool, and classifies it according to [7]; we evaluate the tool in Section 3 through the list of projects in which it has been used, trying to give an idea of its importance for us over the years; finally, we end with some conclusions and future work in Section 4.

2 FEATURES OF SPL-JS-ENGINE

In this section, we first explain the usage of the tool and the types of annotations supported, so the reader can get a clear idea of how it works. Then we describe the general features of the tool.

2.1 Usage

The tool can be used both as a terminal tool and integrated as a library in any NodeJS application. In order to implement a SPL with the tool, a series of files are required, shown in Figure 1: the SPL feature model, which can be in the FeatureIDE XML format, or in the tool's own JSON format; a directory with the annotated source code of the components of the SPL; and a configuration file in JSON format that we will explain below. Using the tool to generate a product requires a last file, shown with yellow background in Figure 1, the specification of the product in JSON format. This last file is the only one that varies to generate different products within the same SPL.

The aforementioned configuration file has two functions. On the one hand, the developer of the SPL defines which are the delimiters that will be used in the annotations. Each file extension[3] can have its own delimiters. That is, the delimiters `/*` and `*/` can be set for Java or JavaScript code, while setting `<!--` and `-->` delimiters for XML files. Thanks to this configuration, we can use delimiters that correspond to the language comments, if the language supports comments[4]. In this way, IDEs will continue to be able to interpret these files correctly in most cases to facilitate the development of annotated code.

After installing the tool via `npm`[5], the tool can be run with the command `spl-js-engine`, as shown in Listing 1. The input files are set via parameters, together with the path where the new product should be generated. The generation of the product's source code is immediate on any modern computer. If the product feature selection does not comply to the feature model, an error is displayed and the build stops. This happens, for example, if there are no sub-features selected for an alternate feature.

```
spl-js-engine --featureModel model.xml --product product.json
      --config config.json --code code --output outputFolder
```

Listing 1: Running the cli tool

More information and a complete example can be seen in the documentation of the GitHub repository of *spl-js-engine*, including a descriptive video showing how to use the tool and the main features[6].

2.2 Annotations

The tool supports three types of annotations. The *basic annotation*, identified by the delimiters set in the configuration file of the SPL, is used to indicate whether or not a piece of code is included in the source code of the product. An example in Java language can be seen in Listing 2, referencing features `add`, `divide`, `multiply`, and `subtract`. Features are accessed through the JavaScript object `feature`, so the annotations are pretty easy to understand. It is important to note that the annotations are removed from the final source code, regardless of whether the feature has been selected or not. This allows the final product to be independent of the product line, and its development can be continued by an external team.

```java
public float calculate(float first, float second, Operation op) {
    switch (op) {
    /*% if (feature.add) { %*/
    case ADD:
        return add(first, second);
    /*% } if (feature.divide) { %*/
    case DIVIDE:
        return divide(first, second);
    /*% } if (feature.multiply) { %*/
    case MULTIPLY:
        return multiply(first, second);
    /*% } if (feature.subtract) { %*/
    case SUBTRACT:
        return subtract(first, second);
    /*% } %*/
    }
    return 0;
}
```

Listing 2: Example of basic annotations

The *interpolation annotation* allows to include specific values from the specification in the product source code. The interpolation syntax is JavaScript, so the developer can use any JavaScript function or expression. Listing 3 shows how the Maven group and artifact values are set from the specification. This annotation type uses the basic annotation delimiters appending an equals symbol (=) symbol after the first delimiter. The data in the specification is accessed through the JavaScript object `data`, and it can be as complex as required (e.g., we have used this annotation to define a complete data model, with its entities, properties and relationships, and generate source code that performs CRUD operations on the data model).

[2]spl-js-engine npm page: https://www.npmjs.com/package/spl-js-engine
[3]If the file does not have an extension, the name can be indicated directly (e.g., a Docker container definition file, `Dockerfile`).
[4]There are languages like JSON that do not support comments, and in this case the annotations will "break" the proper format of the file.
[5]npm comes bundled with NodeJS, which can be downloaded from https://nodejs.org/
[6]GibHub repository wiki: https://github.com/AlexCortinas/spl-js-engine/wiki

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <modelVersion>4.0.0</modelVersion>

    <groupId><!--%= data.maven.group %--></groupId>
    <artifactId><!--%= data.maven.artifact %--></artifactId>
    ...
</project>
```

Listing 3: Example of interpolation annotations

There is one more type of annotation that allows generating new files based on the data of the product specification. These annotations are quite complex to use, intended for cases where the code template needs to be instantiated a number of times. For example, if we have a template to define a Java class, and we need to create ten Java classes for a specific product.

A possible product specification that can be used with the annotations shown in this section is shown in Listing 4.

```
{
  "features": [ "add", "multiply" ],
  "data": {
    "maven": {
      "group": "es.udc.lbd",
      "artifact": "web-calculator"
    }
  }
}
```

Listing 4: Example of product specification

2.3 Capabilities

spl-js-engine is a tool built in JavaScript to implement SPLs following an annotative approach. As mentioned in this section, it support three types of annotations, written in JavaScript language. Some of its most significant features are listed below:

- Can be run in any operating system with NodeJS.
- Can be used as a library within any JavaScript project.
- Can be run directly in a web browser. Input and output files can be files and folders from the local file-system, or compressed files stored in memory.
- It allows to programmatically define a feature model. The supported feature model is basic, with cross-tree constraints, validating the product before generating it to check that the feature selection is correct. Since it does not provide a graphical UI to design the feature model, it supports the XML format from the well-known tool FeatureIDE[7].
- Its development follows the test driven development (TDD) methodology[8].

spl-js-engine was made public at the end of 2016 on GitHub under the MIT license, and has been part of the npm repository since September 25, 2018. Since then it has been downloaded 2 196 times[9] (though probably many of those downloads are installations made by ourselves during the years).

[7]FeatureIDE website: https://www.featureide.de
[8]116 test cases on July 10th, 2022
[9]npm-stats for spl-js-engine: https://npm-stat.com/charts.html?package=spl-js-engine&from=2018-09-01, checked on July 10th, 2022

Next we list the characteristics of the tool according to the classification of [7]. Regarding *Domain Analysis* characteristics, the tool only supports basic features and constraints. Regarding *Requirement Analysis* characteristics, we have some statistics and metrics functionalities such as counting the number of features, the number of annotations, the number of annotated files, the list of files in which an annotation appears, etc. However, we have not thought about these features exhaustively, so support is partial at best. *spl-js-engine* also validates the product specification against the feature model of the product line, although the validation of complex cross-tree constraints with binary operations is still under development (see Section 4). The tool does have propagation of constraints when selecting the features for a product. Regarding *Domain Implementation* characteristics, our tool follows an annotation-based approach, with low abstraction level, and supports multi-language artifacts, as mentioned already. Regarding the *Product Derivation* characteristics, we effectively generate products. The tool itself does not validate the generated products (beyond validating the feature selection) since no test is run after the generation. This is something we normally want to do in the application that the tool is embedded into. Regarding the traceability of the features, there is a functionality to see in which files a certain feature is located, and vice versa, but we have not delved further. Finally, the tool does not handle evolution changes (we are using an evolution flow supported by git and a GitLab community server, similar to [8]).

3 EVALUATION

This section enumerates all the projects where the tool has been used over the years.

3.1 Research publications and R&D projects

There are several publications related to the SPL for web-based GIS, including [3, 4] and the thesis of Alejandro Cortiñas [1]. We also have explored runtime product preview, this is, showing prototypes of the web interface of the products directly within the web application that serves to specify and generate the products [2]. Our SPL for the creation of web applications in the field of Digital Libraries has also been published [9, 10]. *spl-js-engine* was also used in a collaborative work combining compositive and annotative approaches [5, 6].

Regarding R&D projects, *spl-js-engine* was used in LPS-Bigger, already mentioned in the introduction, and in GEMA. **LPS-Bigger** was a large research project carried out in collaboration with large IT companies in Spain. The goal of the project was to develop a SPL able to generate applications with a set of features related with big data management, ingestion, and processing. **GEMA** was a large research project with a work package that aimed at developing a SPL for Mobile Workforce Management, which is a sub-type of GIS.

3.2 Technology transfer projects

We have used the product lines built with *spl-js-engine* in several technology transfer projects. In this section, we briefly describe some of them and we provide some metrics to evaluate their complexity.

LEMSI is an e-learning platform. We have used our SPL as an scaffolding tool to create the first version of the application (data

	LOC	Generated LOC	Removed LOC
LEMSI	191 844	117 326	85 810
Besteiro	43 242	34 455	250
WebEIEL	341 446	323 034	553

Table 1: Evaluation of the complexity of the projects

model and full-stack CRUD functionality). We have then used the product source code as the starting point of the final application.

Besteiro is an e-commerce platform. We have used our SPL again to scaffold the first version of the application, which was then completed with other specific functionalities. After that, we have updated the product using *spl-js-engine* to include in the product bug fixes and changes in the functionality.

WebEIEL is a web-based GIS for publishing and visualizing the information on infrastructures and facilities of the Provincial Council of A Coruña (Spain). It is currently implemented using technology from 2005, and we are creating a new version of the application with our SPL for web-based GIS, based on *spl-js-engine*.

Table 1 shows some metrics of the three applications that were generated using *spl-js-engine*. The first column shows the lines of code (LOC) of the final application. The second column shows the LOC of the product generated by the SPL. The third column shows the LOC that were removed from the generated source-code, this is, lines that were generated but not required for the actual products. We can see that in the final version of LEMSI, there are only around 32k lines of generated code left, which represents the 16% of the code. This is an example of a project without GIS nature, that used the product line only as an starter, and that anyway took advantage of it since almost all the back-end source code is used (the source code of the web user interface was rebuilt entirely by our development team). Both Besteiro and WebEIEL are ongoing projects, and therefore the LOC removed are still very low (less than 1%). Regarding the LOC in the projects that come from the SPL, it is a 79% in Besteiro, and a 94% in WebEIEL.

3.3 Bachelor/Master's thesis

We have developed four prototypes of SPLs with our students:

Blog generation tool, a simple SPL to generate blog applications, presented on September 18th, 2019. This student re-implemented the example of [5], which was later used in [6].

A tool for the semi-automatic generation of software for Digital Libraries, which later evolved in the SPL for BIDI mentioned above [9, 10], presented on September 21st, 2020.

Native application generation in the domain of Mobile Workforce Management, presented on September 21st, 2020. This projects consists on developing a SPL to generate Android applications that implement a subset of the features of the GEMA project.

Software product line for native mobile applications in the field of Mobile Workforce Management, presented on July 7th, 2021. This project is similar to the previous one, but using Flutter instead of native Android code.

4 CONCLUSIONS

In this short paper we have briefly introduced *spl-js-engine*, a tool that we have been using for the last seven years to implement all the SPLs that we worked on. We have developed an annotative SPL derivation engine that can be used in industry-level projects.

As future work, we are working in a few ideas. We will change the way the analysis and validation of the feature models and the feature selection of the products are done to use a SAT Solver. Right now, we have implemented an ad-hoc solution using object orientation with methods that validate each constraint and the relationships between the features. We may also migrate the part of the code related to feature models into an independent project. In fact, the feature model creation, import, export, analysis and validation functionalities are already encapsulated.

Secondly, we are working on synchronizing the web tool that is used to generate products with our GitLab, so that the evolution of the products using an approach based in source code versioning (similar to [8]) becomes automatic.

Finally, the tool will soon enable modular product specifications. The specification of one of our products is becoming too big when in fact is composed by conceptually independent parts (map viewers, static pages, SLD styles). Therefore, we are currently working on allowing the specification to consist of not just a single file, but several referenced files.

ACKNOWLEDGMENTS

Partly funded by: MCIN/AEI/10.13039/501100011033, NextGenerationEU/PRTR, FLATCITY-POC: PDC2021-121239-C31; MCIN/AEI/10.13039/501100011033 EXTRACompact: PID2020-114635RB-I00; GAIN/Xunta de Galicia/ERDF CEDCOVID: COV20/00604; Xunta de Galicia/FEDER-UE GRC: ED431C 2021/53; MICIU/FEDER-UE BIZDEVOPSGLOBAL: RTI-2018-098309-B-C32; MCIN/AEI/10.13039/501100011033 MAGIST: PID2019-105221RB-C41.

REFERENCES

[1] Alejandro Cortiñas. 2017. *Software Product Line for web-based Geographic Information Systems.* Ph. D. Dissertation. Universidade da Coruña.
[2] A. Cortiñas, Carlo Bernaschina, M. R. Luaces, and Piero Fraternali. 2017. Improving GISBuilder with Runtime Product Preview. In *Procs. of 17th Int. Conf. on Web Engineering (ICWE 2017) -LNCS 10360.* Roma, 549–553.
[3] A. Cortiñas, M. R. Luaces, O. Pedreira, and A. S. Places. 2017. Scaffolding and in-browser generation of web-based GIS applications in a SPL tool. In *Proc. 21st Int. Systems & Software Product Line Conf. (SPLC 2017) Vol.2.* 46–49.
[4] A. Cortiñas, M. R. Luaces, O. Pedreira, A. S. Places, and J. Perez. 2017. Web-based Geographic Information Systems SPLE: Domain Analysis and Experience Report. In *Proc. 21st Int. Systems & Software Product Line Conf. (SPLC 2017) Vol.1.* 190–194.
[5] J. M. Horcas, A. Cortiñas, L. Fuentes, and M. R. Luaces. 2018. Integrating the Common Variability Language with Multilanguage Annotations for Web Engineering. In *Proc. 22st International Systems & Software Product Line Conference (SPLC 2018) vol.1.* Gothenburg, 196–207.
[6] J. M. Horcas, A. Cortiñas, L. Fuentes, and M. R. Luaces. 2022. Combining multiple granularity variability in a software product line approach for web engineering. *Information and Software Technology* (2022).
[7] José Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2022. Empirical analysis of the tool support for software product lines. *Software and Systems Modeling* (2022). https://doi.org/10.1007/s10270-022-01011-2
[8] Leticia Montalvillo and Oscar Díaz. 2015. Tuning GitHub for SPL development: branching models & repository operations for product engineers. In *Procs. of the 19th Int. Conf. on Software Product Line Pages - SPLC'15.* ACM, 111–120.
[9] O. Pedreira, D. Ramos Vidal, A. Cortiñas, M. R. Luaces, and A. S. Places. 2021. Development of Digital Libraries with Software Product Line Engineering. *Journal of Web Engineering* (2021), 2017–2058.
[10] D. Ramos Vidal, A. Cortiñas, M. R. Luaces, O. Pedreira, and A. S. Places. 2020. A Software Product Line for Digital Libraries. In *Proc of the 16th Int. Conf. on Web Information Systems and Technologies (WEBIST 2020).* Online, 321–334.