Integration of Flamapy.js in a Software Product Line Engineering Platform to Enhance the Analysis of Feature Models

Antón Soto Ríos, Víctor L. Sardiña, and Miguel R. Luaces

Centro de Investigación CITIC, Universidade da Coruña, 15071 A Coruña, Spain
Correspondence: anton.soto@udc.es

DOI: *https://doi.org/10.17979/spu.23.c27*

Abstract: SPLALM is a Product Line Engineering factory development environment that introduces two key innovations. It integrates with GitLab and git to enable consistent and coordinated changes in complex product families. It also supports parsing, persisting, analyzing, and visualizing Feature Models (FMs) using the Universal Variability Language, a community-driven standard for variability representation. To support FM analysis directly in a browser, SPLALM uses Flamapy.js, a Pyodide-based WASM wrapper of the Flamapy framework. This paper focuses on the integration of Flamapy.js and how it upgrades the analysis of FMs in SPLALM. Together, these features align SPLALM with ISO/IEC 26580 standards, offering a reliable solution for managing high-variability product lines.

1 Introduction

Nowadays, one of the main problems in software development is the *reuse of code* and how to handle *variability* across different projects with similar features (Pohl et al. (2005)). A Software Product Line (SPL) is a collection of related software systems that share a common set of core assets but also present well-defined variations to satisfy the specific needs of different products within the same domain (Clements and Northrop (2001)). By adopting SPLs, organizations can raise productivity, reduce costs, shorten time-to-market, and ensure better consistency and quality across products. In order to achieve such management of variability, Feature Models (FMs) are employed. FMs are structured representations of the commonalities and variability of a product line. There are different types of feature models, ranging from basic tree-based notations to more advanced extensions that incorporate constraints, attributes, and multi-level variability, allowing developers to formally capture and reason about product configurations (Kang et al. (1990)).

Currently, in the Database Laboratory (LBD) we are working on a Software Product Line Engineering (SPLE) platform called **SPLALM** (Buján et al. (2024)). This platform introduces several new functionalities that distinguish it from other existing approaches. One of its main contributions is the integration with Git and GitLab, which enables version control for the product lines generated through the platform. In addition, SPLALM provides support for parsing, persisting, analyzing, and visualizing feature models in order to manage variability in its products. For this purpose, it adopts the *Universal Variability Language* (*UVL*) (Sundermann et al. (2021)), a community-driven standard that offers a simple yet powerful way of representing and handling feature models.

Besides representing variability, feature models are also analyzable and can be used to exploit interesting information about both the model and its configurations. Examples include counting valid configurations, checking for dead features, identifying core features that are always

selected, etc. While such data can be extracted manually for simple models, this task quickly becomes infeasible as models grow in size. To address this challenge, several tools exist, one of which is **Flamapy** (Flamapy Contributors (2025)) a framework developed at the University of Seville that implements *Automatic Analysis of Feature Models* (*AAFM*) in Python using SAT and BDD solvers, among others.

However, SPLALM is a web application and therefore cannot use Flamapy directly, since it is implemented in Python. In this paper we present **Flamapy.js**, a Pyodide-based[1] WebAssembly wrapper of Flamapy written in JavaScript. This integration enables automatic feature model analysis directly in web browsers. As a result, users can perform advanced reasoning tasks efficiently and seamlessly, without leaving the web platform. This paper focuses on presenting Flamapy.js as a tool, alongside the key tests conducted to ensure its proper functioning and its integration into SPLALM. The rest of the paper is structured as follows: Section 2 introduces the background for this research. Then, the architecture of flamapy.js and its integration with SPLALM are presented in Section 3. After that, in section 4, the results of the comparison between Flamapy and Flamapy.js are shown. Finally, conclusions and future work are presented in Section 5.

2 Background

Feature models are the most widely used formalism to represent variability in Software Product Line Engineering. They provide a hierarchical and structured representation of features, the functionalities or characteristics of a product line by distinguishing between mandatory, optional, alternative, and or-features. This structure allows developers to capture both *commonalities*, which are shared across all products, and *variability*, which defines the possible differences between products in the same family. In addition to the tree structure, modern FMs incorporate *cross-tree constraints* (e.g.,"if feature A is selected, feature B must also be included"), attributes, and multi-level variability. These extensions enable more expressive models that are not only descriptive but also analyzable, making FMs a key artifact for configuration, reasoning, and automated derivation of products within SPLs (Felfernig et al. (2024)).

As we said, Flamapy is a framework that provides a unified infrastructure for (*AAFM*). As we can see in their GitHub (Flamapy Contributors (2025)) they offer a considerable amount of operations. Among the core ones that it supports are:

- **Consistency checking**: verifies whether at least one valid product configuration exists given the constraints of the FM.
- **Counting configurations**: computes the total number of valid products that can be derived from the model.
- **Detection of dead features**: identifies features that cannot be part of any valid product, usually due to contradictory constraints.
- **Detection of core features**: finds features that are always included in every possible product configuration.

Some other interesting operations that we can see in the repository are: atomic sets, average branching factor, commonality (of a feature), configuration list, count leaf features, estimate the number of configurations, check some false optional features, check the ancestors (of a feature), max depth of the model, check if the model is satisfiable, etc. Internally, Flamapy combines different solving technologies, mainly *SAT* and *BDD* (*Binary Decision Diagram*) solvers, to ensure scalability and precision across different types of FMs. By combining these algorithms with its modular architecture, Flamapy has become a useful and extensible tool for FM analysis.

[1] *https://pyodide.org/en/stable/*

3 Proposal

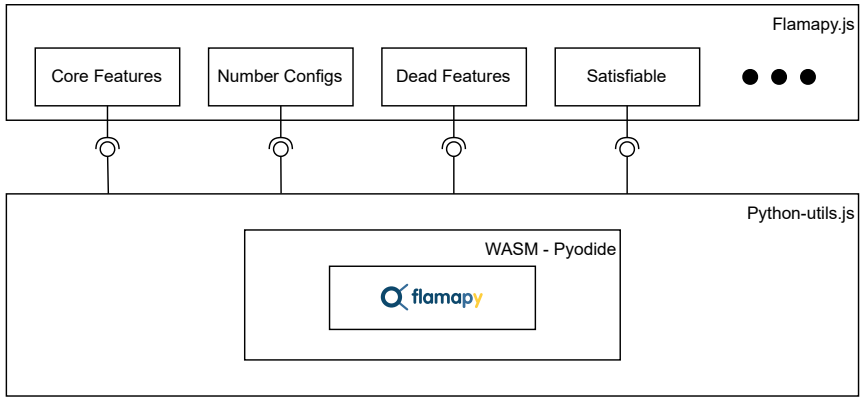

Figure 1: Architecture of Flamapy.js integrating Flamapy and Pyodide.

As mentioned, *Flamapy* is a Python-based tool for feature model analysis that cannot run directly in browsers. By integrating it with Pyodide, we were able to create a JavaScript package called Flamapy.js. This design enables external applications to interact solely with Flamapy.js, abstracting the complexity of running Python code in a WebAssembly environment. A visual representation of this architecture is shown in Figure 1. The resulting architecture can be described as follows:

- **Flamapy.js**: Exposes a constructor to create an object so the Flamapy operations that we mentioned before can be executed, working as an interface.

- **python-utils.js**: Provides a wrapper around Pyodide functions to enable seamless interaction between JavaScript and Python code.

- **Pyodide**: WebAssembly-based (Haas et al. (2017)) Python environment responsible for executing Flamapy and its dependencies.

- **Flamapy**: Python libraries implementing the feature model analysis algorithms (FMAA).

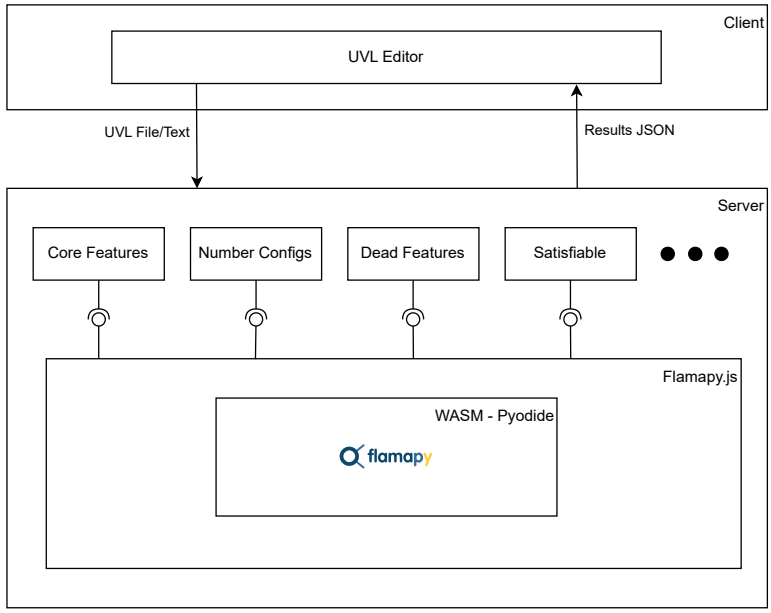

Figure 2: Integration of Flamapy.js into the SPLALM platform.

Once Flamapy.js is available, the SPLALM platform can fully benefit from automated feature model analysis, enabling it to provide users with metrics, reasoning tasks, and configuration analysis. The resulting integration architecture is illustrated in Figure 2, where Flamapy.js operates as the central bridge between the SPLALM and Flamapy. Flamapy.js operates on the server side, acting as an interface between the client layer and the underlying *Flamapy* operations. The workflow proceeds as follows: the user writes a UVL model through the SPLALM client, which sends it to the server. On the server, a *Flamapy.js* instance validates the syntax and executes all *Flamapy* functionalities via Pyodide. Once the analysis is complete, the results are collected as a JSON object and sent back to the client for visualization. This approach provides a clean and modular architecture: the client and server interact only through well-defined interfaces and do not need to be aware of the internal behavior of the layers beneath them, allowing the platform to leverage *Flamapy*'s capabilities without exposing its complexity to the end user.

4 Results

4.1 Robustness and Efficiency Analysis

This section reports the experimental evaluation of `flamapy.js` and its integration into SPLALM. The evaluation had two main objectives: (i) to validate that all functionalities of `flamapy.js` can be executed correctly on a large dataset of UVL models, and (ii) to compare its performance against the original Python-based `Flamapy`. To this end, we created scripts that systematically execute every operation from both frameworks over the complete set of UVL models in the UVLHub (Variability Community (2023)) repository. We executed them a few times, for each one we recorded whether it succeeded or failed, as well as the corresponding execution time. A failure was counted whenever an operation exceeded the global timeout of 20 seconds, which mainly occurred on computationally expensive analyses. This limit was deliberately introduced to prevent `flamapy.js` from freezing indefinitely, since for large models

some operations would trigger memory exhaustion in the WebAssembly environment, leaving the execution blocked without returning any result. Therefore, the timeout ensures that all executions are bounded in time and that failures correspond to computationally expensive analyses that cannot be completed within the given resources.

Table 1: Execution success and failure counts for `flamapy.js` and the original `Flamapy` across all UVLHub models.

Framework	Avg. Successes	Avg. Failures	% of Successes
`flamapy.js`	1099	192	85%
`Flamapy (Python)`	1145	146	88%

Table 2: Execution time for `flamapy.js` and the original `Flamapy` across all UVLHub models.

Framework	Avg. Success Time	Avg. Total Time
`flamapy.js`	6.09s	2h 56m 47s
`Flamapy (Python)`	0.59s	1h 9m 26s

The data in Table 1 demonstrates that Flamapy.js offers a robust and acceptably alternative to the original Flamapy (Python) framework for web-based analysis because it achieves a high degree of reliability with an 85% success rate (1099 average successes), only slightly behind the Python version's 88% (1145 average successes). Most of the failures in both frameworks are concentrated in the `configurations` and `configurationsNumber` operations, which are known to be computationally demanding and prone to timeouts on large models. The overhead is most evident in the time metrics shown in Table 2. The average success time for flamapy.js is 6.09s compared to the highly optimized 0.59s for Flamapy (Python). Despite this difference, flamapy.js maintains acceptable efficiency for interactive use within SPLALM. Despite these limitations, `flamapy.js` maintains performance within practical bounds, making it a suitable alternative for web-based analysis when a Python backend is not available.

4.2 Integration into SPLALM

In this section, we introduce the integration of `flamapy.js` into SPLALM. Through its web-based interface, users can write feature models, which are analyzed in real time, allowing results to be obtained directly in the browser without requiring a backend. Figure 3 illustrates the SPLALM user interface with `flamapy.js` integration, showing how analyses are launched and how results are displayed.

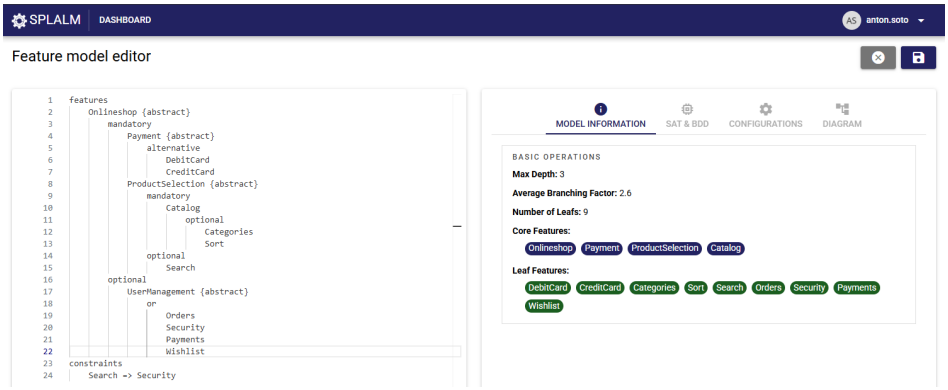

Figure 3: SPLALM user interface integrating `flamapy.js`.

The figure shows the user interface where the feature model analysis is performed. On the left side, the UVL editor allows the user to write feature models, which can then be saved or analyzed. On the right side, the `flamapy.js` analysis panel is displayed, consisting of four tabs: the first tab presents general information, the second tab contains operations that can be executed with different solvers, the third tab focuses on configuration-related operations, and the fourth tab provides a visual representation of the UVL as a diagram. In the second tab, the user can select whether operations are executed using SAT or BDD solvers. In the third tab, a configuration can be selected, allowing the user to generate a product of the Product Line based on that configuration.

5 Conclusions and future work

In our work, we verified that `flamapy.js` is capable of correctly executing the majority of analysis operations over a large benchmark of feature models, demonstrating both robustness and reliability. When compared to the original Python version of `Flamapy`, our results show that, despite some overhead due to the additional architectural layers introduced by the JavaScript/WebAssembly integration, `flamapy.js` maintains reasonable efficiency suitable for web-based development environments, which was its intended use. Thus, our results confirm that `flamapy.js` can provide users with real-time information about feature models without requiring a Python backend. Also, we integrated `flamapy.js` in SPLALM, a Software Product Line management platform that was enhanced by the in-browser analysis of `flamapy.js`, directly supporting design, validation, and reasoning tasks.

5.1 Future Work

There are several directions in which this work can be improved:

- **Model transformations in flamapy.js**: one of the most useful features in product line engineering is the ability to transform feature models into other representations or to refactor them while preserving their semantics. This is already implemented in `flamapy` but its not done for web applications. Bringing transformation capabilities into `flamapy.js` would allow developers to perform tasks such as simplifying models or eliminating redundancies directly in the browser.
- **Integrate UML class modeling in SPLALM (ongoing work)**: we are extending SPLALM to support UML class diagrams as a complementary modeling. This addition will enable users to represent data models as well as UVLs, adding more information to the product line in the same platform.

- **Deeper analysis and reasoning of the FM**: one way to improve `flamapy.js` could be with advanced reasoning techniques that are more complex than the ones presented in `flamapy`. For instance, engineers could simulate the impact of adding or removing features or evaluate the consequences of constraints before applying them to the real model. This could support "what-if" analyses directly in the browser, helping domain experts refine and improve the feature models of their product lines.
- **Better visualization of analysis results**: although `flamapy.js` can compute complex analyses, the raw numerical outputs are often difficult to interpret. A good future line of work is to provide interactive visualizations that represent results such as dead features, core features, or the distribution of valid configurations. This would help users to quickly gain insights into the model and identifying anomalies.

Bibliography

A. Buján, A. Cortiñas, and M. R. Luaces. Development of a ple factory environment with gitlab integration and following iso/iec 26580. In *Proceedings of the 28th ACM International Systems and Software Product Line Conference*, pages 34–37, 2024.

P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, USA, 2001.

A. Felfernig, A. Falkner, and D. Benavides. *Feature Models: AI-Driven Design, Analysis and Applications*. Springer Nature, Cham, Switzerland, 2024.

Flamapy Contributors. Flamapy. *https://github.com/flamapy/flamapy*, 2025. [Online; accessed 25-September-2025].

A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, J. Bastien, and M. J. Franz. Bringing the web up to speed with webassembly. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017)*, pages 185–200, New York, NY, USA, 2017. Association for Computing Machinery. doi: 10.1145/3062341.3062363.

K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 1990.

K. Pohl, G. Böckle, and F. J. Van Der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, Berlin, Germany, 2005.

C. Sundermann, K. Feichtinger, D. Engelhardt, R. Rabiser, and T. Thüm. Yet another textual variability language? a community effort towards a unified language. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume A (SPLC '21)*, pages 136–147, New York, NY, USA, 2021. Association for Computing Machinery. doi: 10.1145/3461001.3471145.

Variability Community. Uvlhub: A repository of uvl feature models. *https://github.com/variability-org/uvlhub*, 2023. [Online; accessed September-2025].