Extending the Identification of Object-Oriented Variability Implementations using Usage Relationships

Johann Mortara[1] Xhevahire Tërnava[2] Philippe Collet[1]
Anne-Marie Dery-Pinna[1]

[1]Université Côte d'Azur, CNRS, I3S, France
[2]Université de Rennes 1, INRIA/IRISA, France

SPLC, REVE, September 07, 2021

Context

- Most modern object-oriented systems are variability-rich
  - ~ their variability is hardly documented or made explicit in code
  - ~ lack of approaches on identifying their variability
  - ~ lack of approaches on representing (visualizing) their variability

Context

- Most modern object-oriented systems are variability-rich
  - ~ their variability is hardly documented or made explicit in code
  - ~ lack of approaches on identifying their variability
  - ~ lack of approaches on representing (visualizing) their variability

) *sym nder*: a tooled approach (Java [1] and C++ [2])

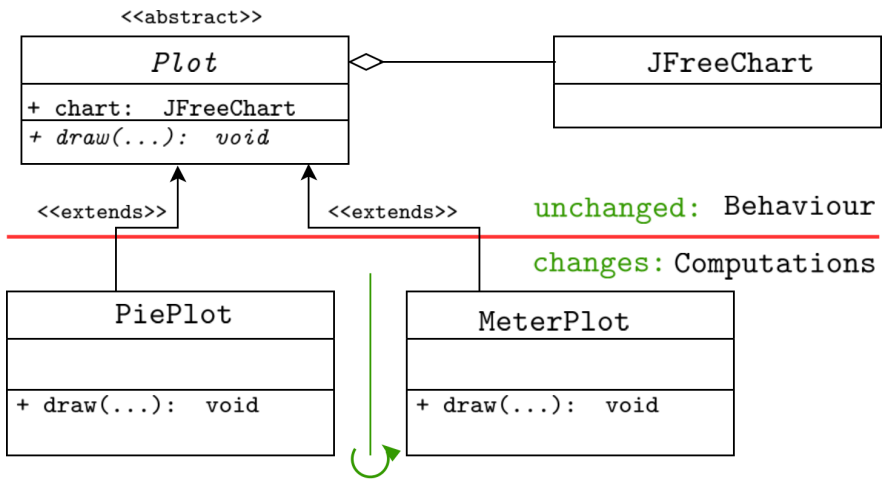
- ~ Identifying variability implementation places (Variation Points with Variants) in single codebase object-oriented systems
  - * based on the property of symmetry in 7 traditional techniques
- ~ Visualizing of Variation Points (VPs) with Variants
  - * based on their density

[1] https://doi.org/10.1145/3336294.3336311

[2] https://doi.org/10.1145/3382026.3431251

Example of symmetry in object-oriented techniques

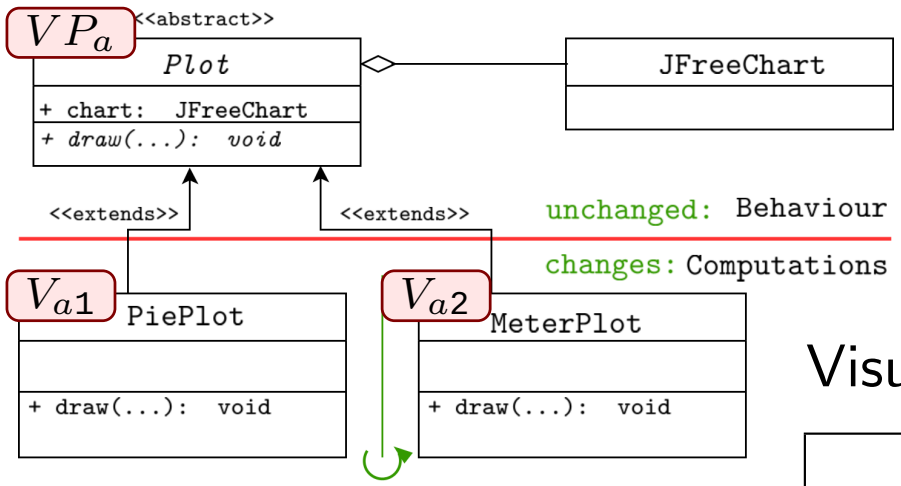
Variability in JFreeChart:



~ Most of the traditional techniques can be interpreted in terms of symmetry

Example of symmetry in object-oriented techniques

Variability in JFreeChart:



~ Most of the traditional techniques can be interpreted in terms of symmetry

Visualization by *sym nder*

Problem statement

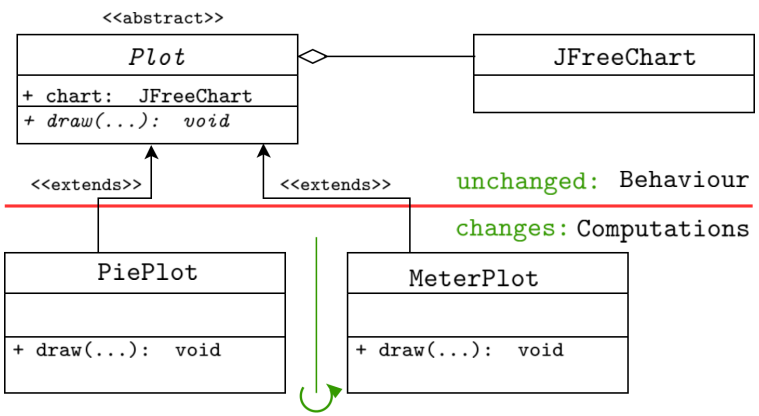
symfinder: Applied: $>$ 15 real open-source systems

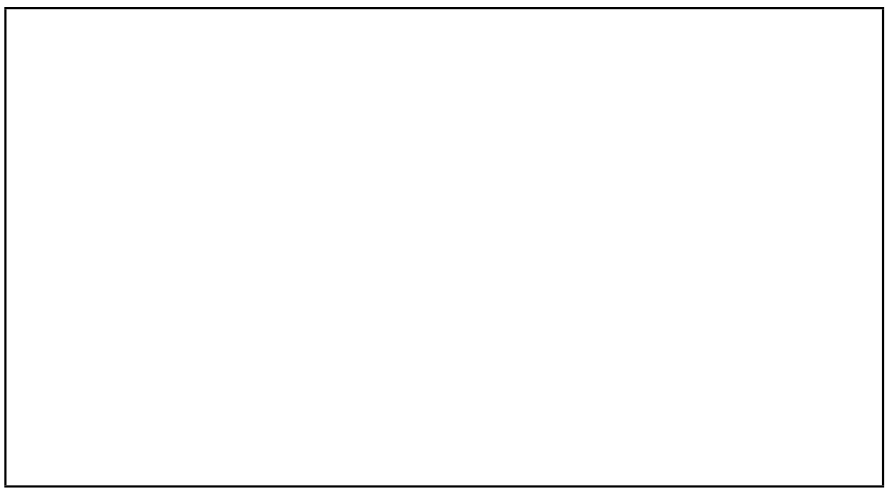
- Identified: 200 - 11K potential VPs with Variants
- Precision: potential $\supset$ real VPs with Variants

Issue 1: Identifying inheritance relationships is not enough

- Composition of instances is not taken into account

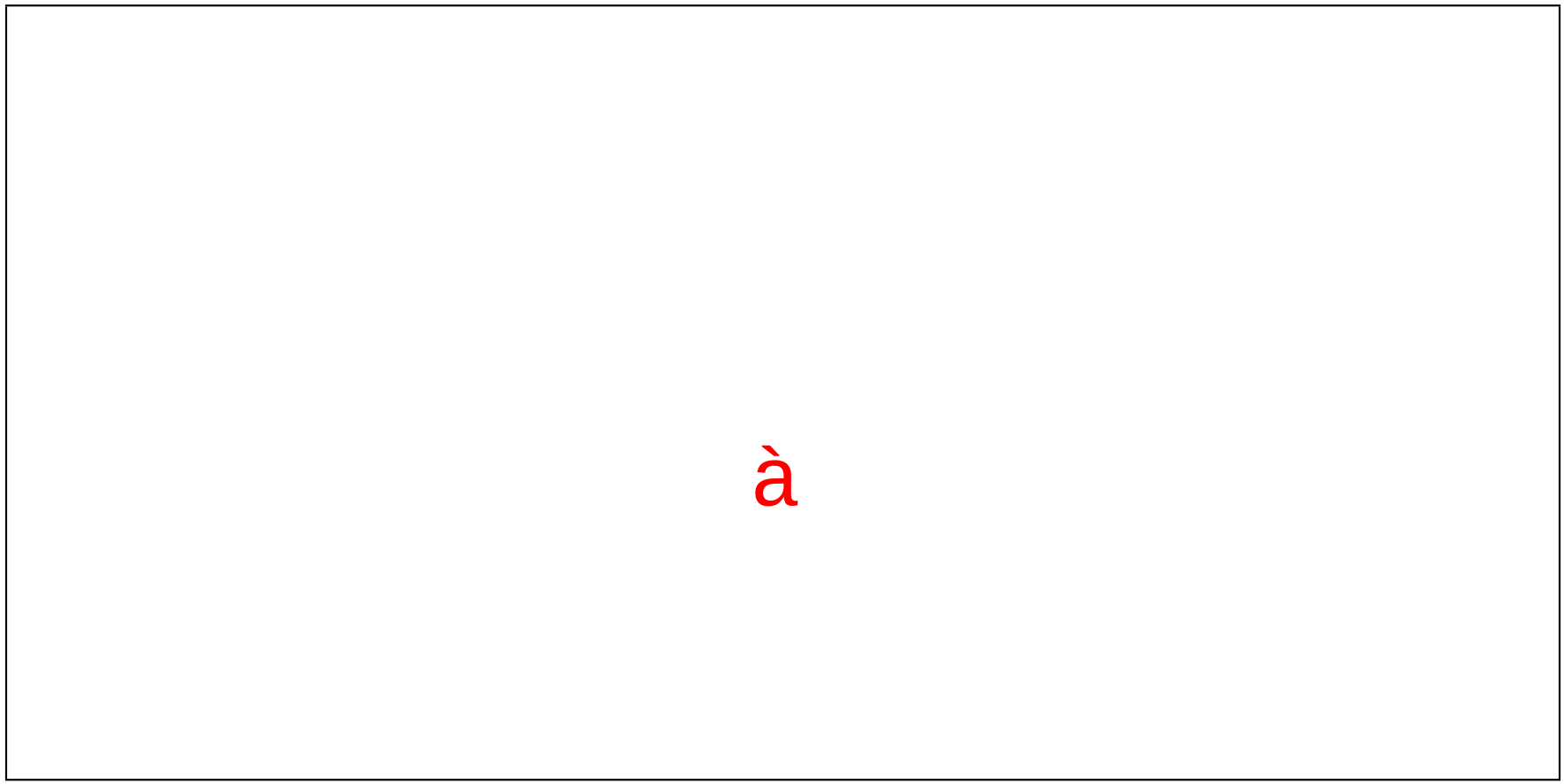
Issue 2: Entry points are missing: for browsing the visualization





sym nder-2

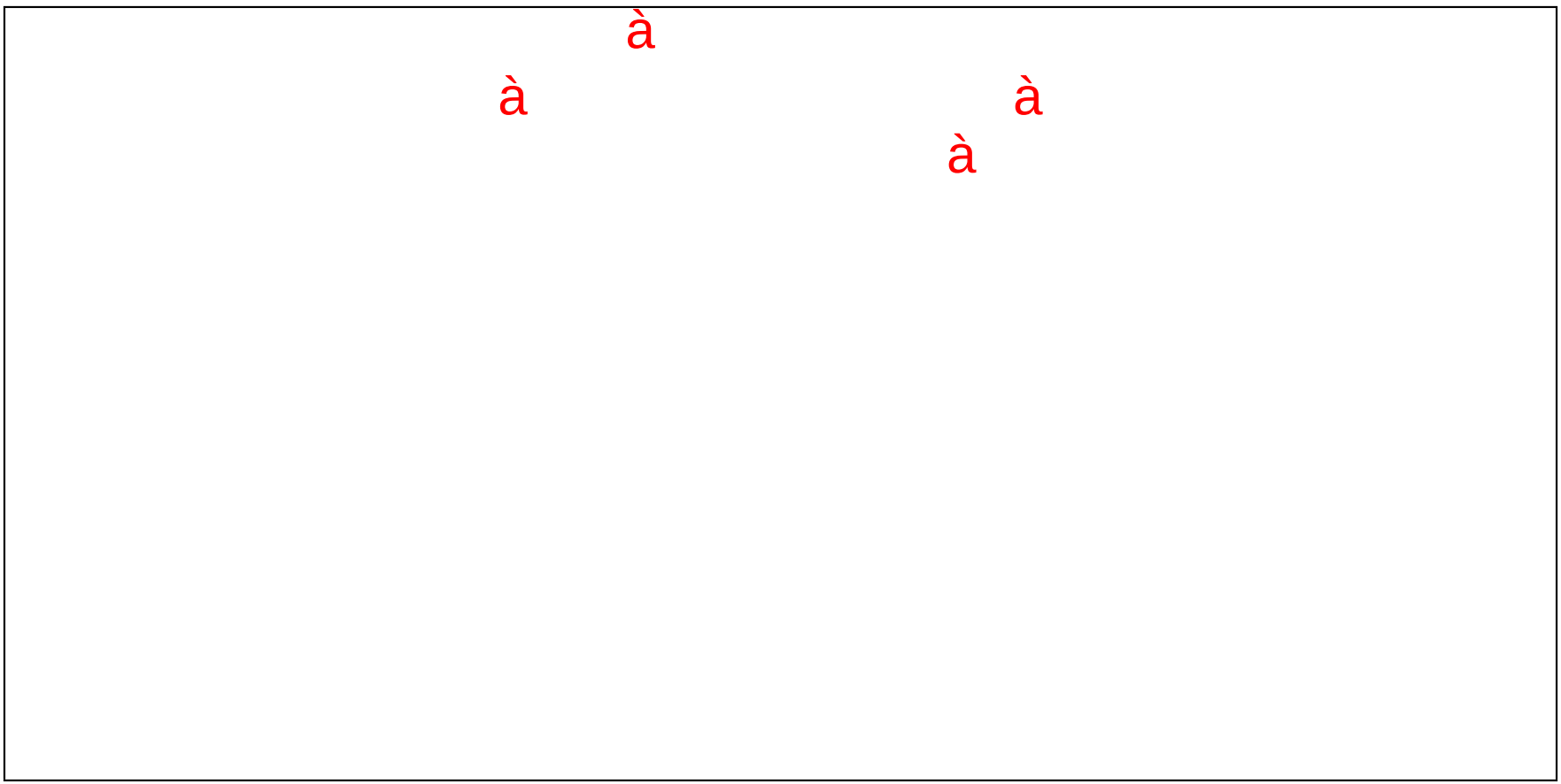
Extension 1: handling usage relationships (+7 variability implementation techniques)



sym nder-2

Extension 2: handling the entry points

- 4 user-de ned entry points à ( )
- Automatically de ned entry points (using the system's API)



Evaluation

- In 10 open-source Java-based systems, regarding 4 research questions

Evaluation

RQ$_1$: Does the identi cation of usage relationships have changed the variability visualization of a given system by sym nder-2?

- Same variability; denser places with variability; less isolated nodes

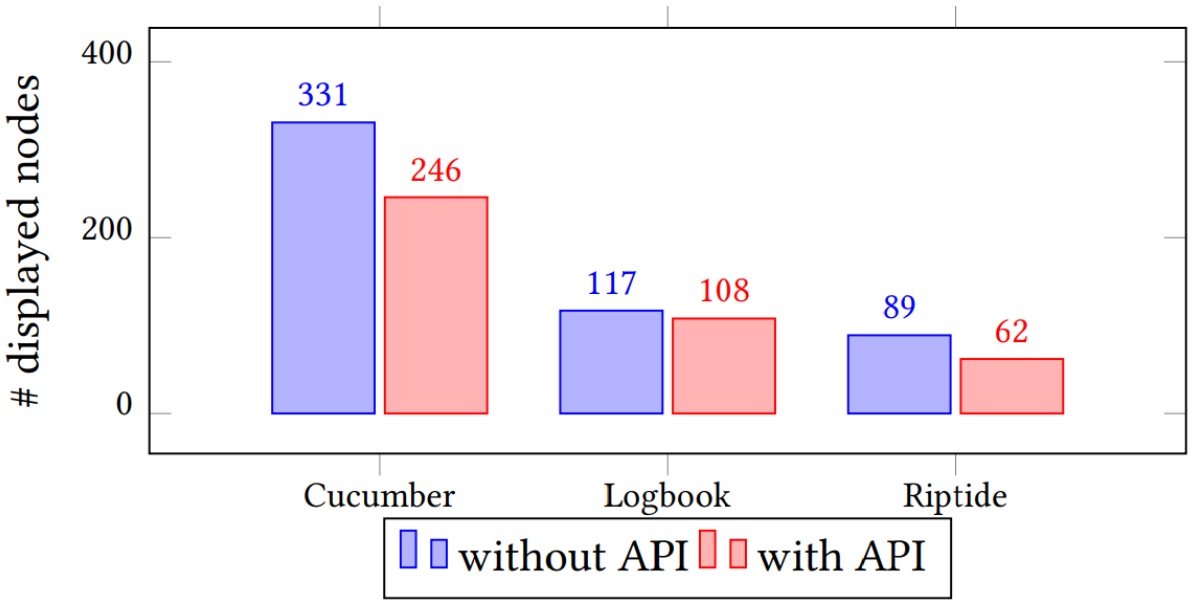
Evaluation

RQ$_2$: What is the starting density threshold to begin with the comprehension of the visualized variability by sym nder-2?

- The threshold parameters depend on the studied system; First threshold can be used as a good starting point

Evaluation

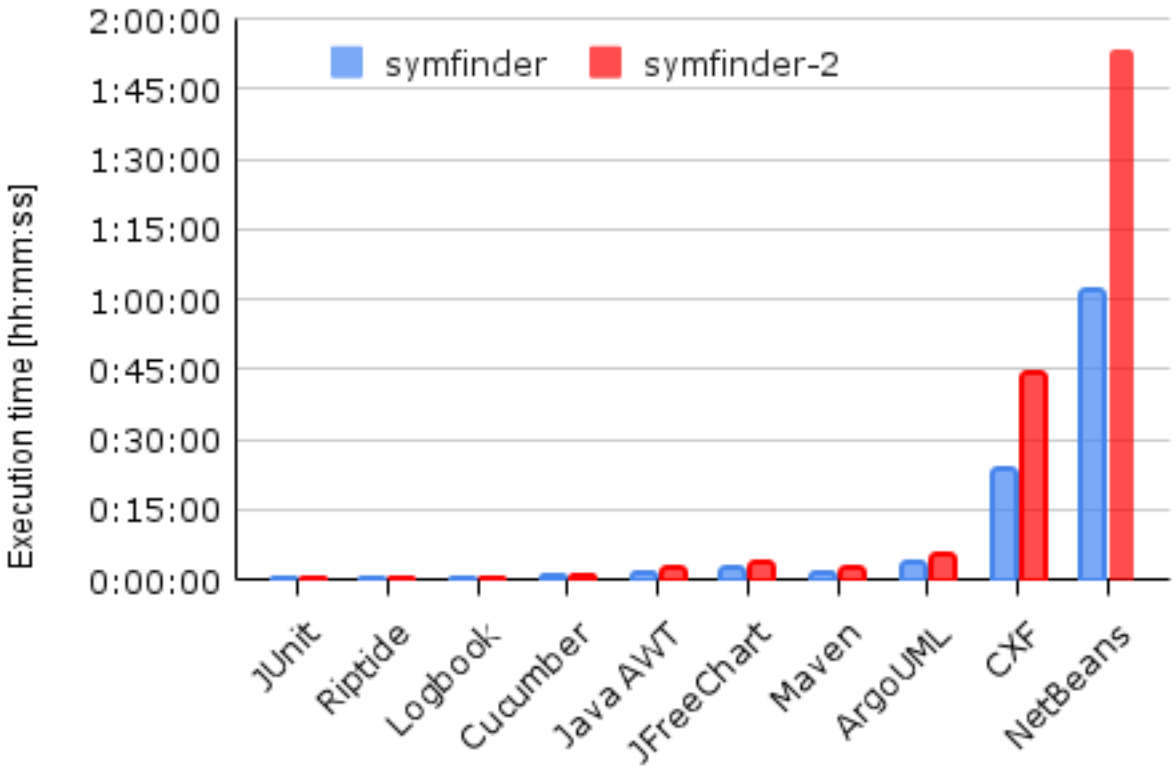
RQ_3: Is the API information of a given system useful to simplify its identi ed variability by sym nder-2?



- Potential VPs with variants) relevant variability places; Integrate different variability information sources

Evaluation

RQ_4: Does the identi cation of usage relationships impact the scalability of sym nder-2?



- Visualization: 700ms in Chrome, 850ms in Firefox for NetBeans
- The time difference is increased with the size of analysed system

Summary

- A tooled approach for I&V potential VPs with Variants in OO systems, with a single code base, implemented in Java
- Extended symfinder) symfinder2
- It identifies the impl. variability by 8 traditional techniques
- It provides 5 entry points for variability comprehension
- Propose to integrate different variability sources

Availability:
https://deathstar3.github.io/symfinder2-demo/