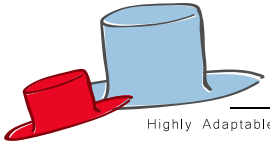
Delta Modeling Workflow

Michiel Helvensteijn

CWI, Amsterdam

LIACS, Leiden University

VaMoS, 26-01-2012



HATS

Highly Adaptable and Trustworthy Software using Formal Models

SEVENTH FRAMEWORK PROGRAMME

http://www.hats-project.eu

Two Talks

The two related talks today

- "Delta Modeling Workflow": The theory, an example
- "Delta Modeling in Practice": Industrial Case Study

Slides and Handouts

`http://mhelvens.net/professional/talks/dmw-vamos2012`
`fas-vamos2012`

Features and Product Lines

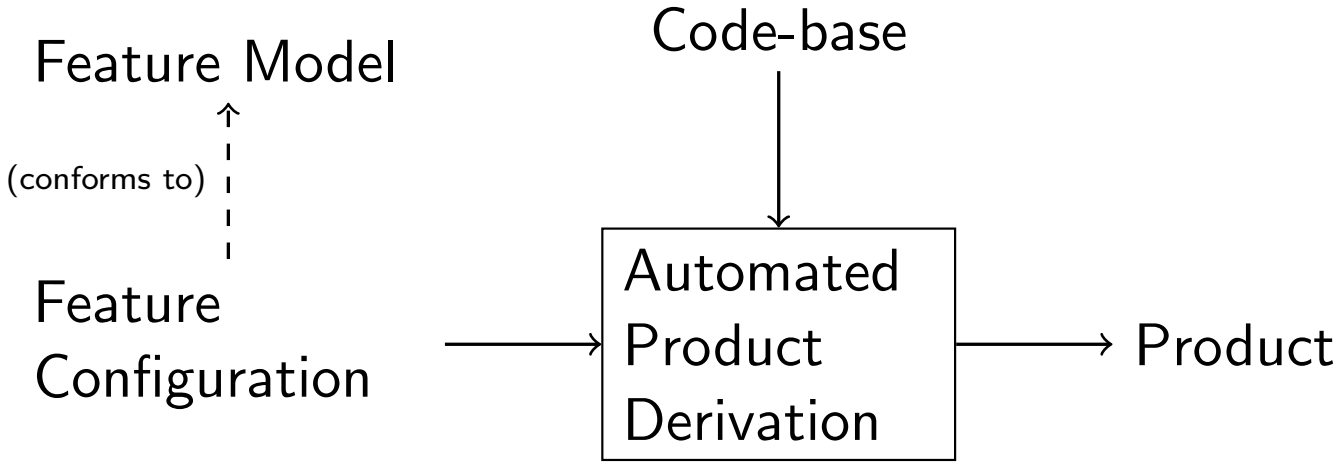
A (software) product line consists of a number of (software) products that differ in which features they support:

- Linux Kernel:
  - Loadable Module Support
  - Power Saving Support
  - Support for various hardware
- Payment System:
  - Chipknip Support
  - Creditcard Support
  - NFC Support
  - Cash Support
- Code Editor:
  - Syntax Highlighting
  - Error Checking
  - Printing

Features and Product Lines

The code-base for an SPL should be organized in some way to reflect which features correspond to which code. This gives us:

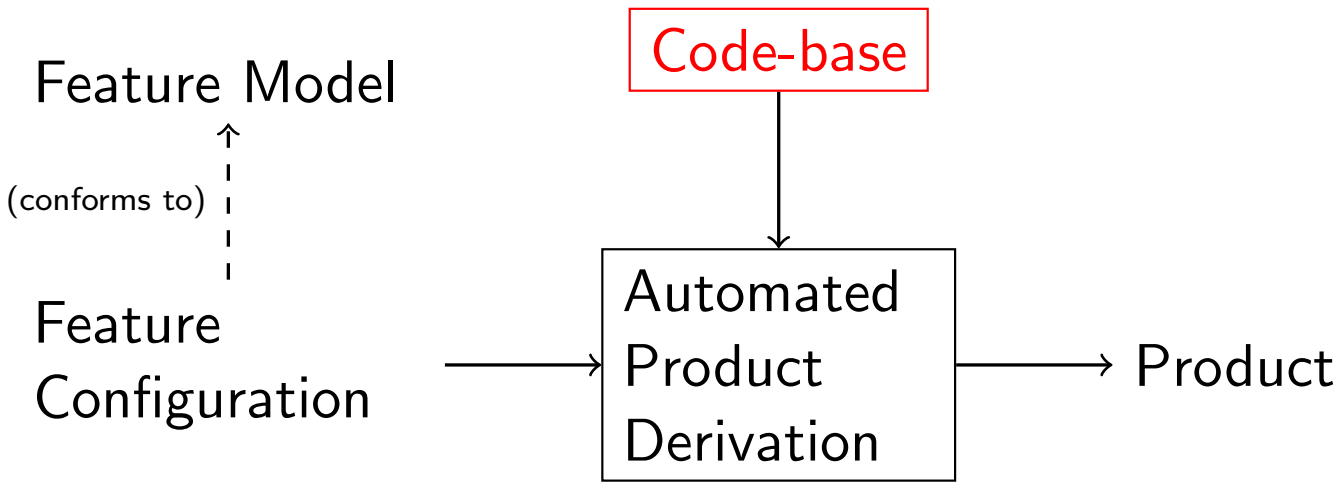
- isolated and concurrent development of features and
- automated product derivation.



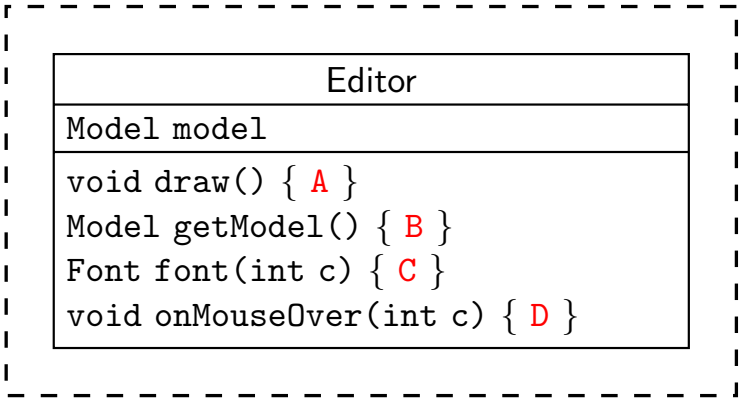
Features and Product Lines

The code-base for an SPL should be organized in some way to reflect which features correspond to which code. This gives us:

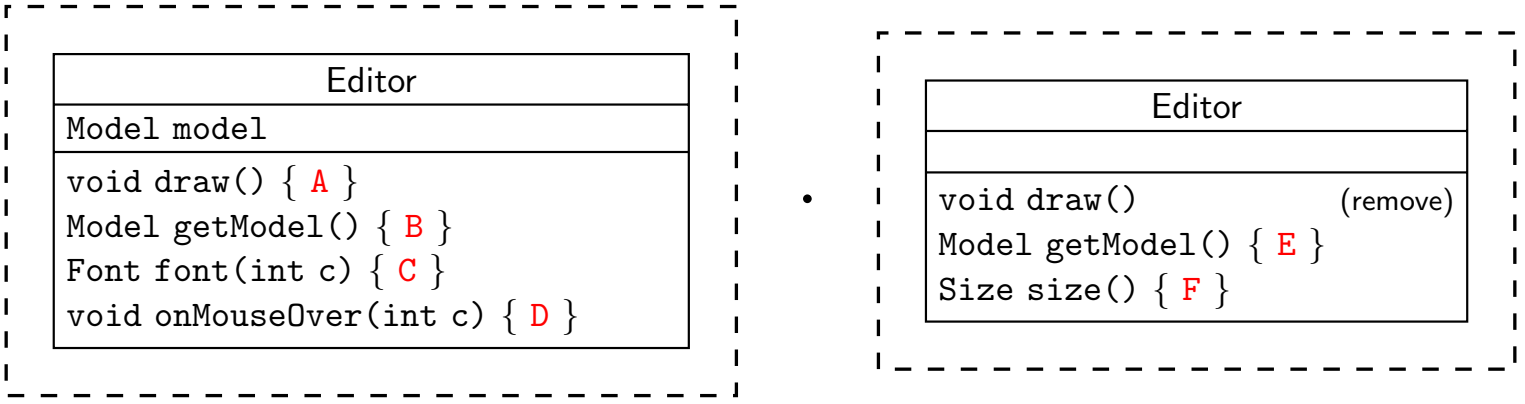
- isolated and concurrent development of features and
- automated product derivation.



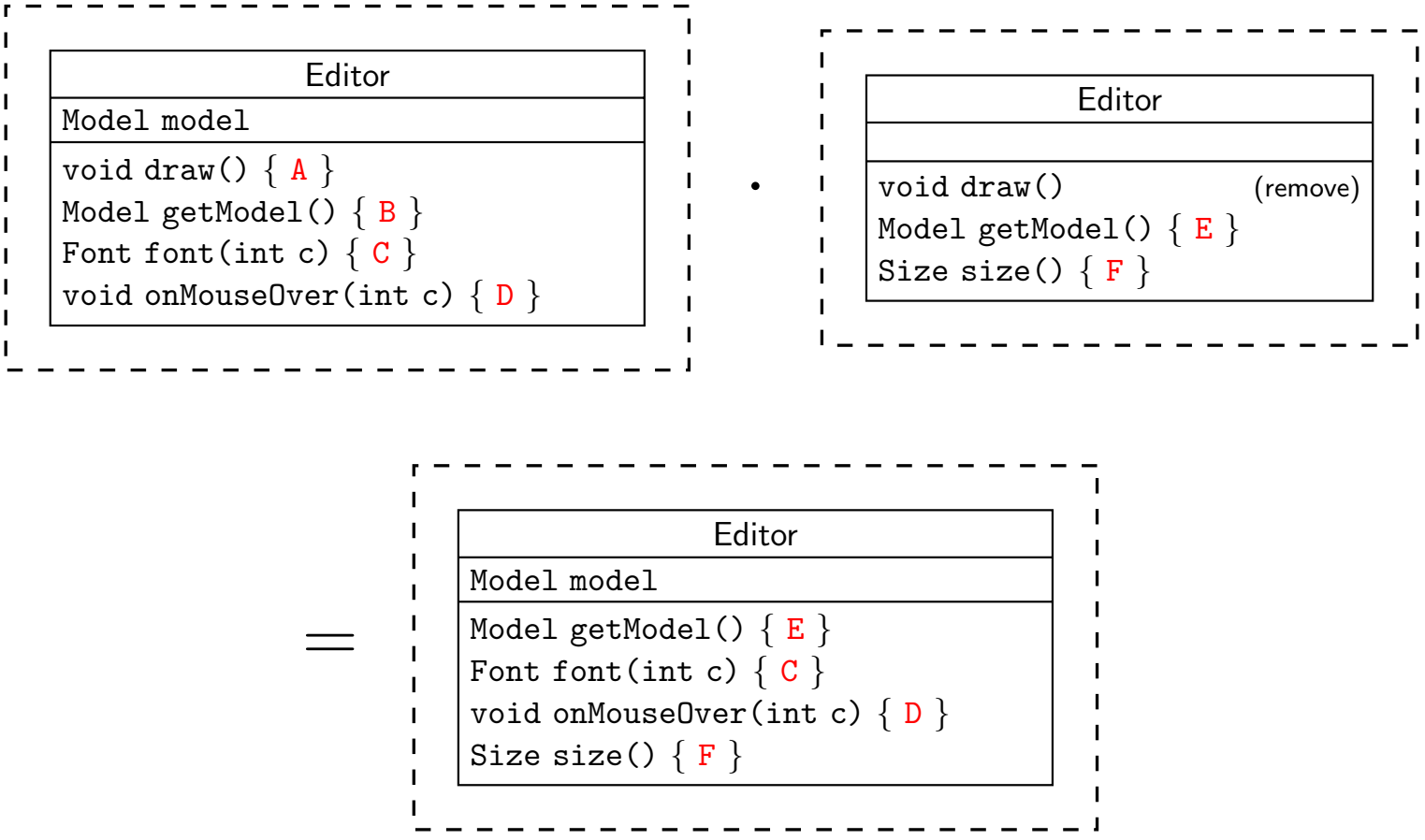
Deltas


Editor
Model model
void draw() { A }
Model getModel() { B }
Font font(int c) { C }
void onMouseOver(int c) { D }

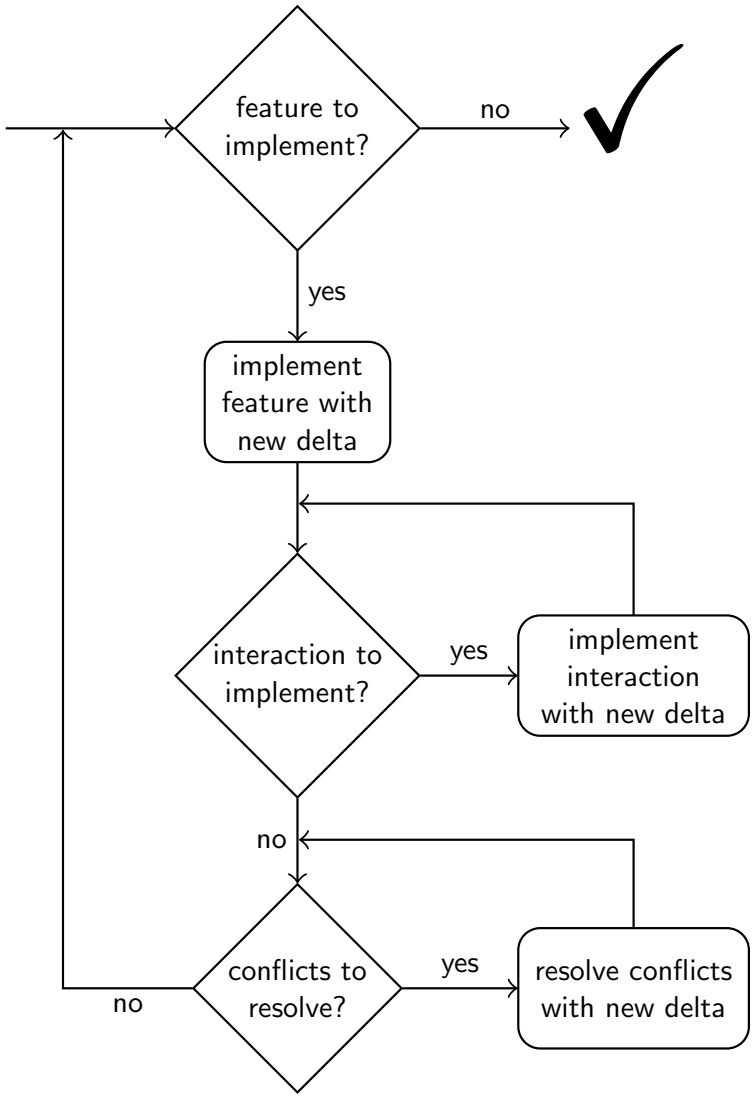
Deltas


Editor
Model model
void draw() { A }
Model getModel() { B }
Font font(int c) { C }
void onMouseOver(int c) { D }
•
Editor
void draw() (remove)
Model getModel() { E }
Size size() { F }

Deltas


Editor
Model model
void draw() { A }
Model getModel() { B }
Font font(int c) { C }
void onMouseOver(int c) { D }
•
Editor
void draw() (remove)
Model getModel() { E }
Size size() { F }
=
Editor
Model model
Model getModel() { E }
Font font(int c) { C }
void onMouseOver(int c) { D }
Size size() { F }

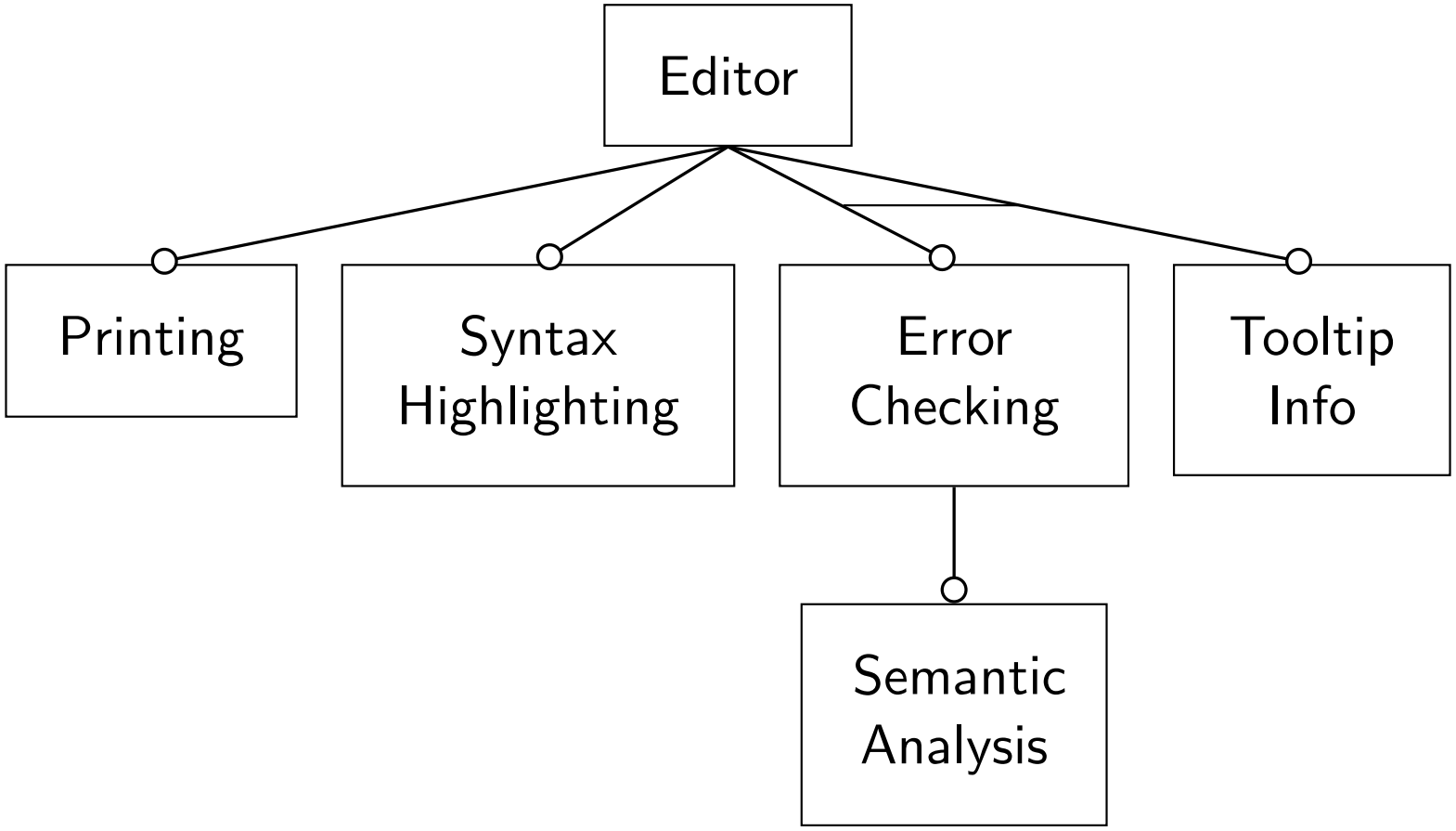
The Workflow



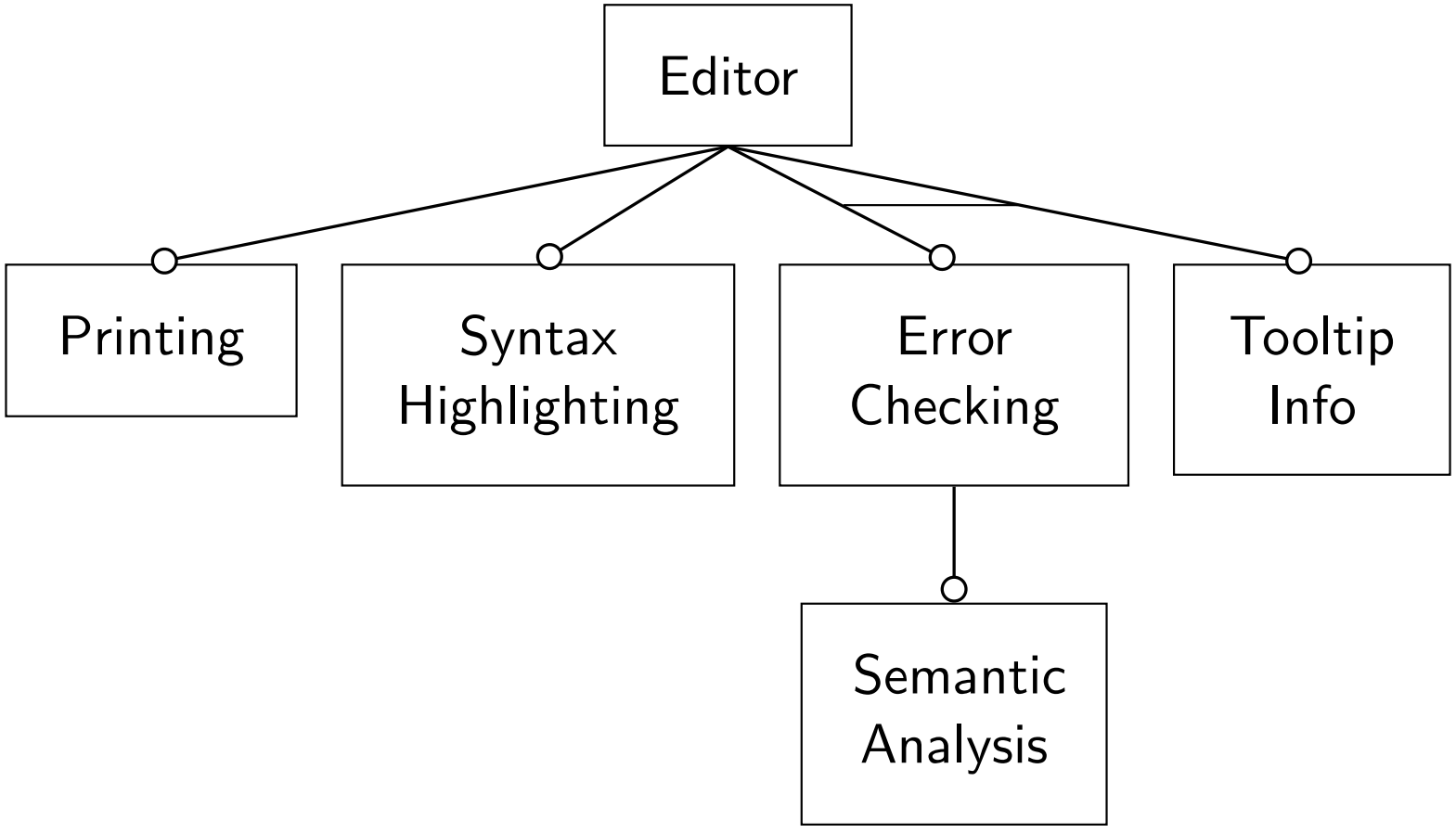
Workflow Properties

- The resulting product line is globally unambiguous (no conflicts).
- Every product will satisfy the specifications of its features.
- Features can be implemented concurrently and in isolation.
- Code duplication and overspecification will be minimized.

Example: Feature Model

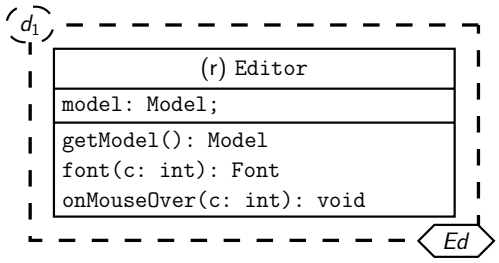


Example: Feature Model

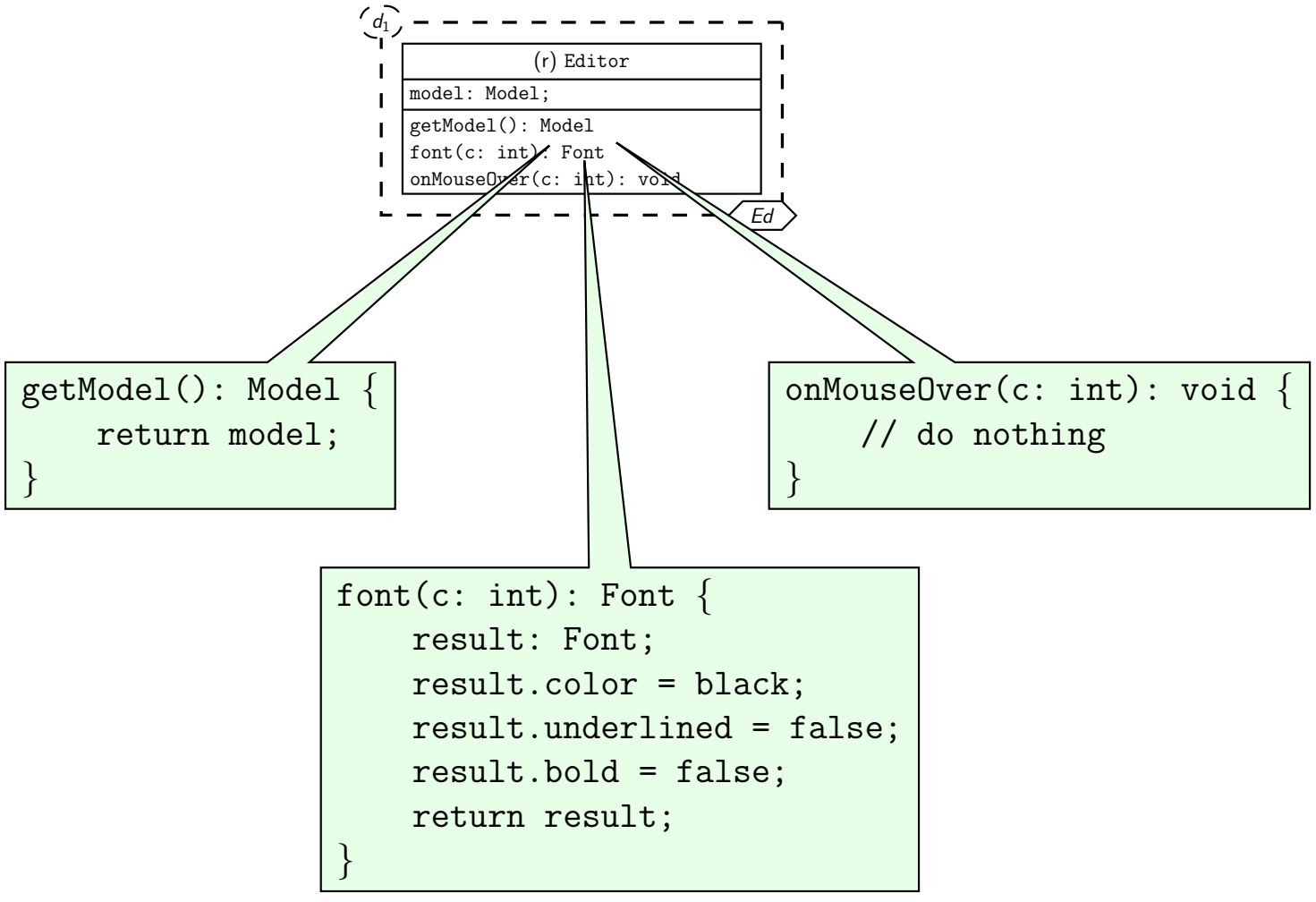


$2 \times 2 \times 4 = 16$ products

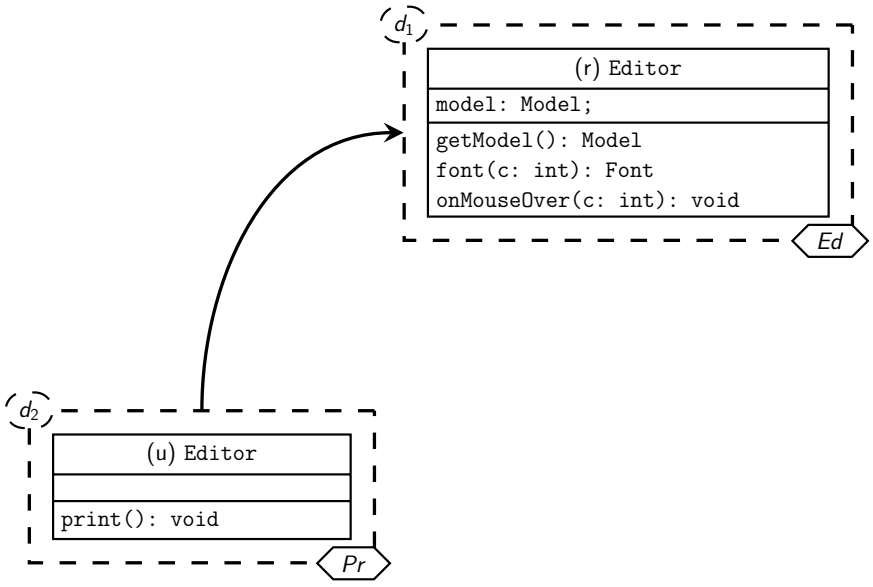
Example: Applying the Workflow



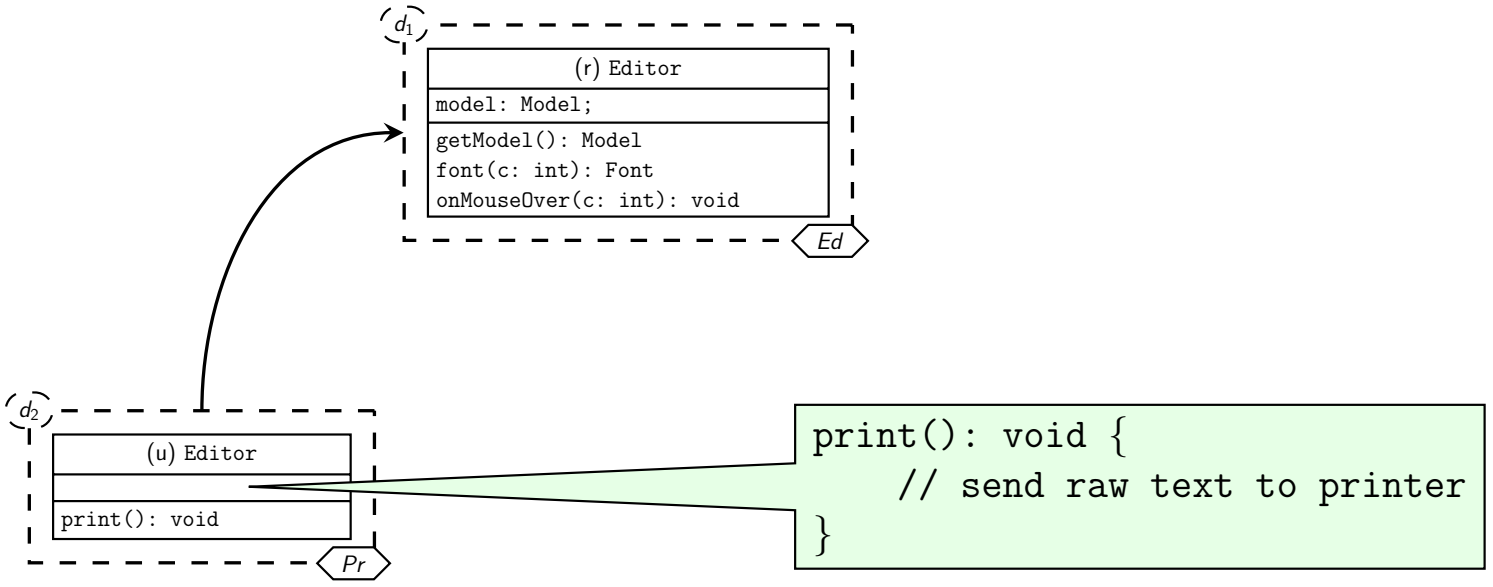
Example: Applying the Workflow



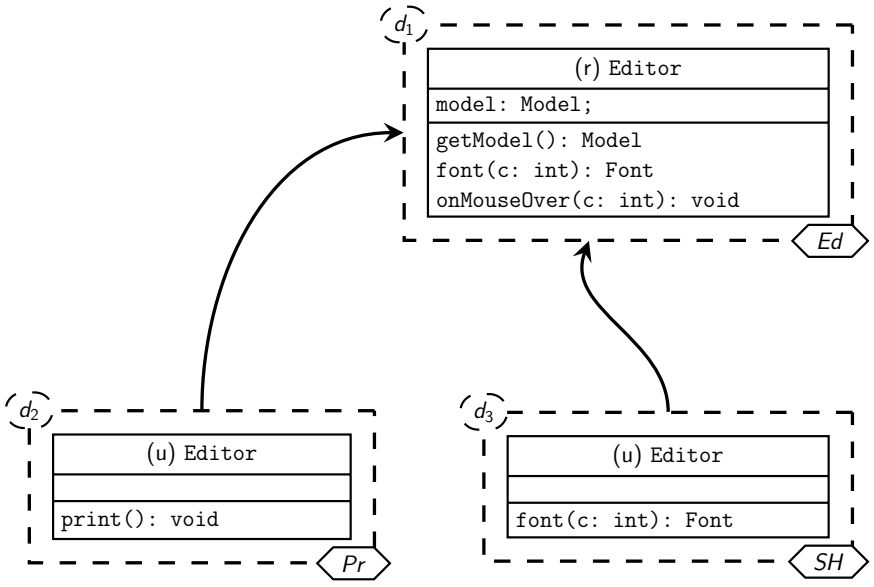
Example: Applying the Workflow



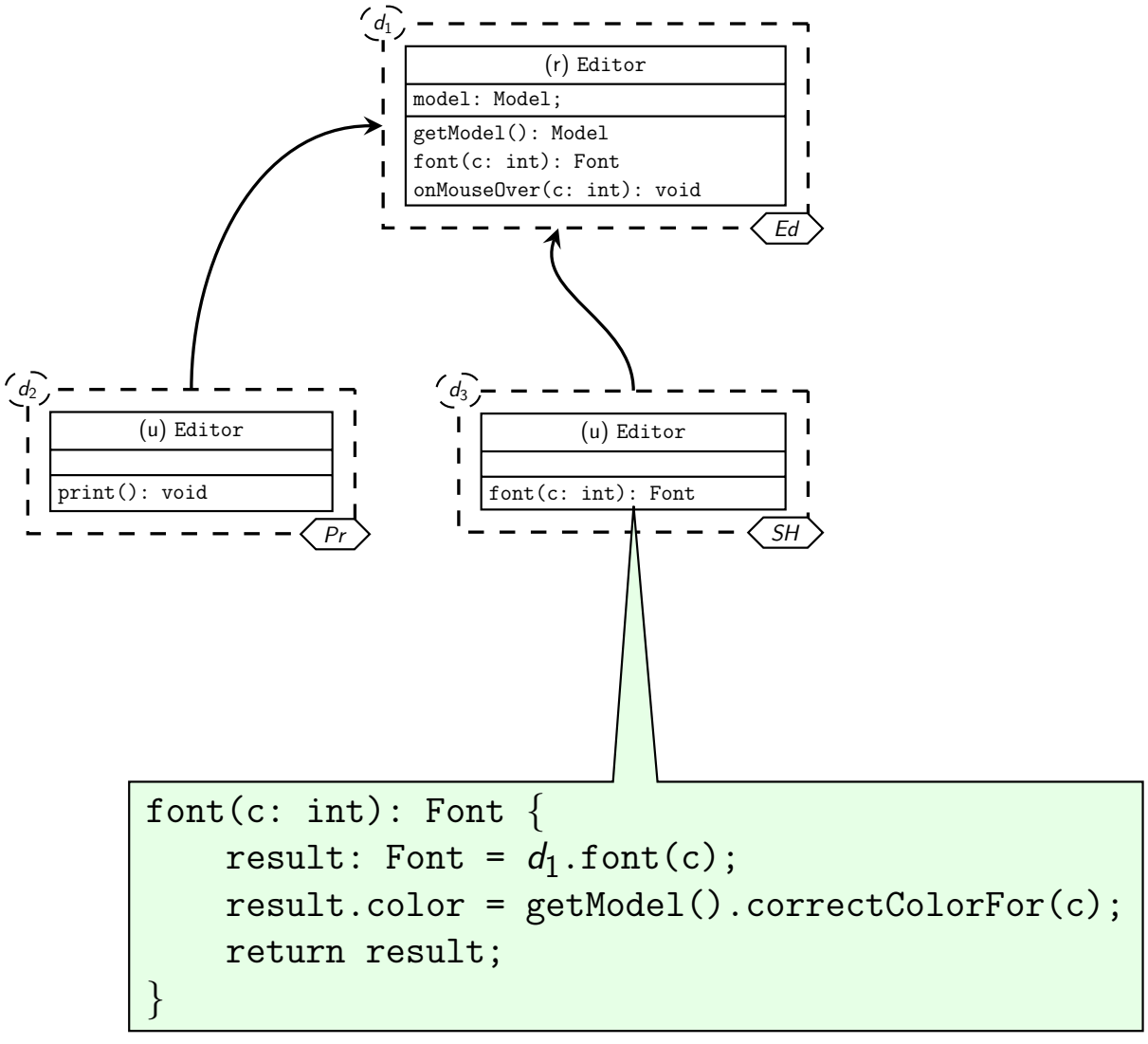
Example: Applying the Workflow



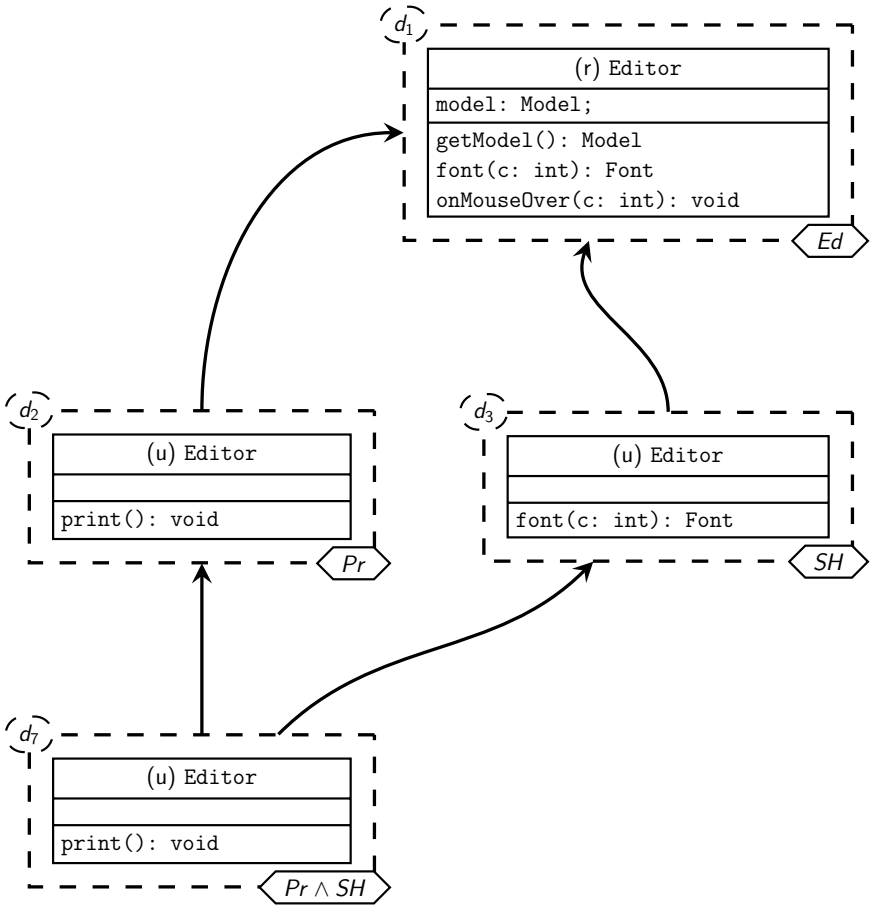
Example: Applying the Workflow



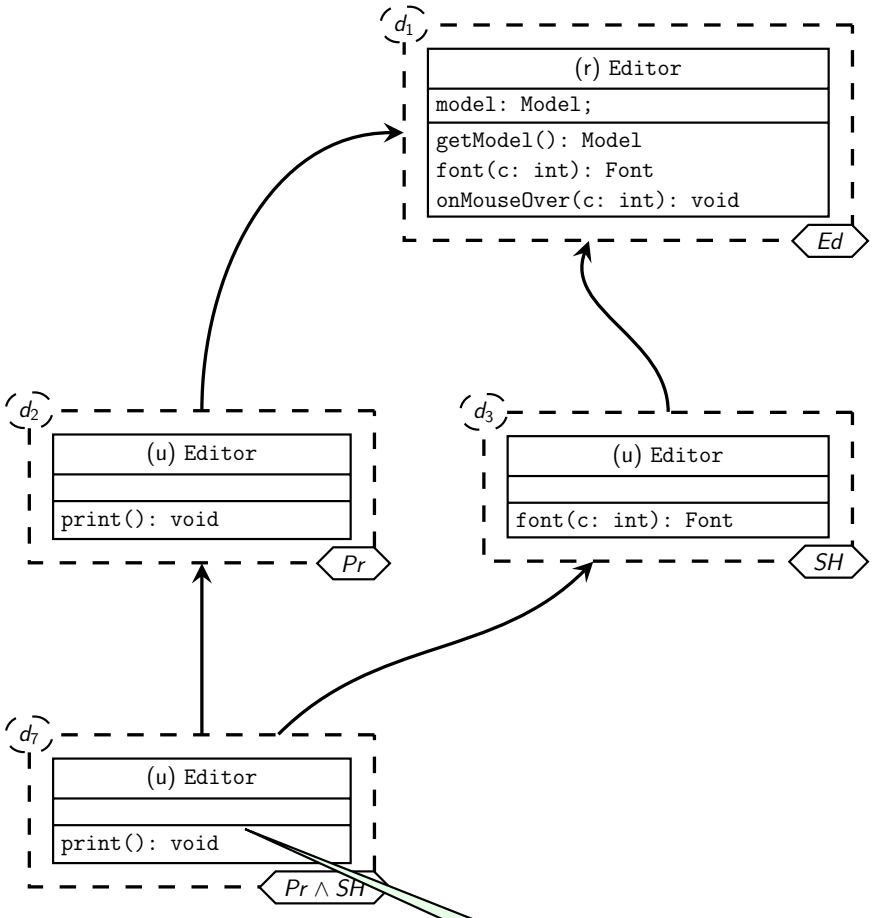
Example: Applying the Workflow



Example: Applying the Workflow

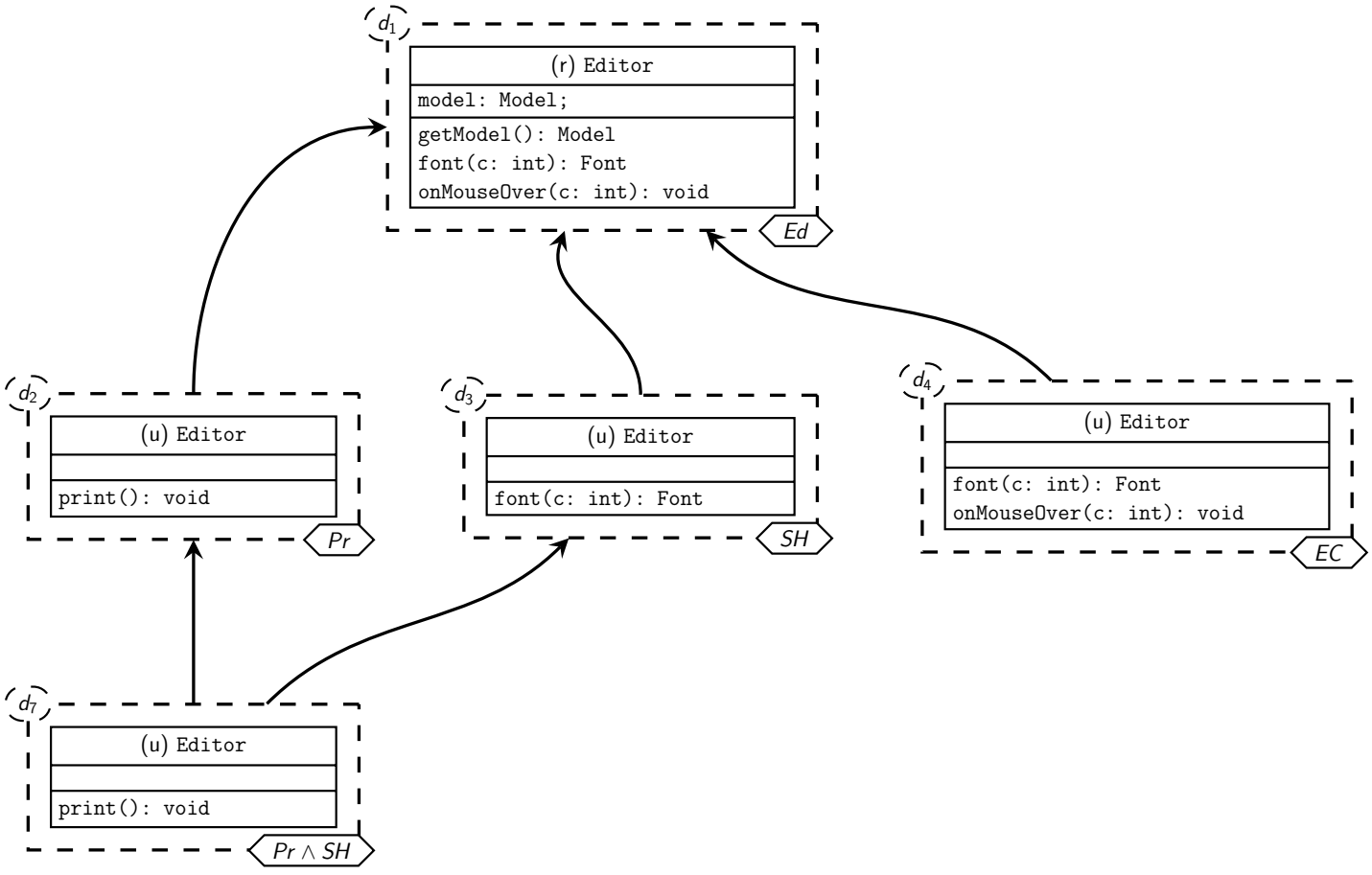


Example: Applying the Workflow

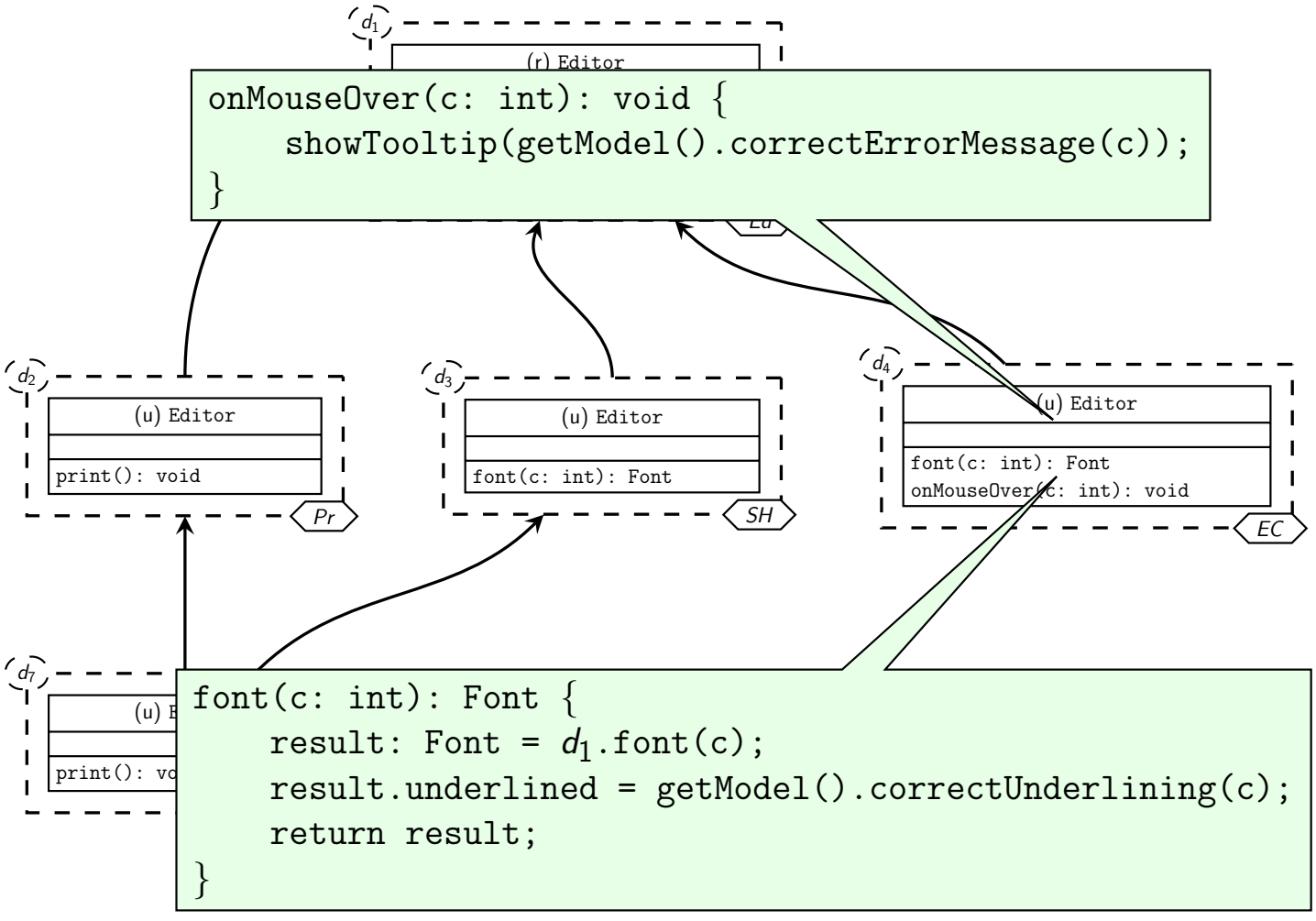



```
print(): void {
    // send text to printer, colored using d3.font(c).color
}
```

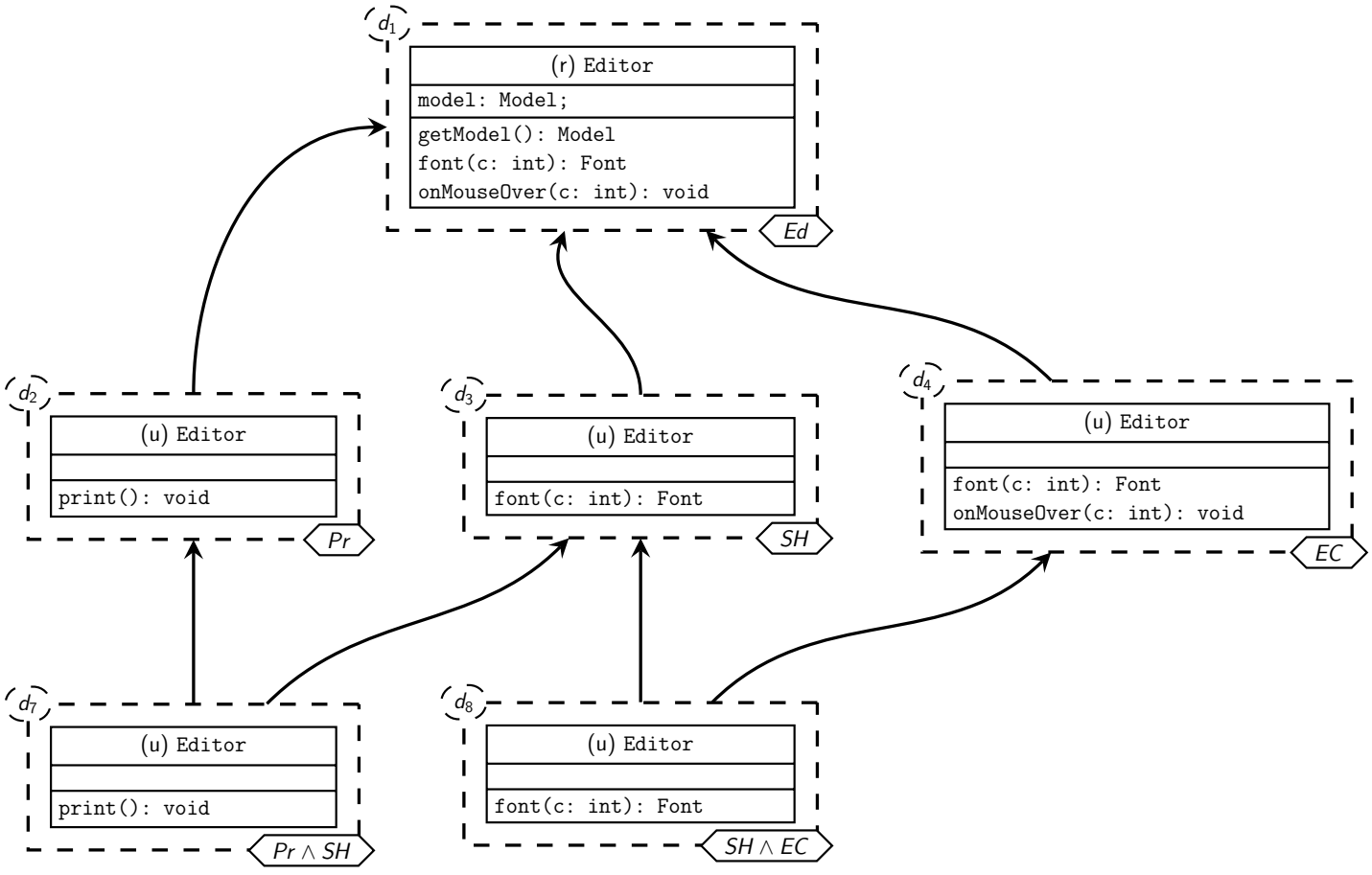
Example: Applying the Workflow



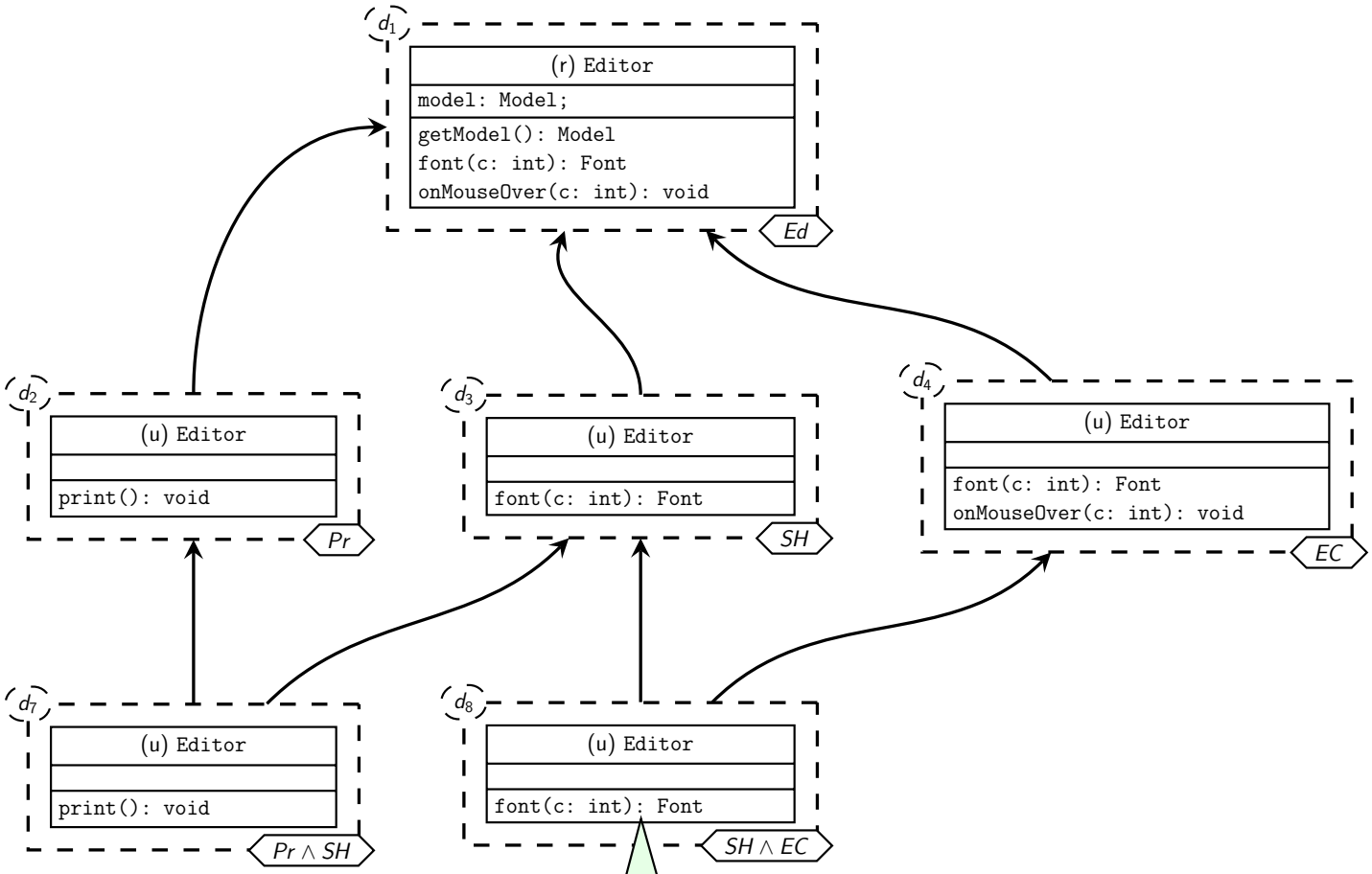
Example: Applying the Workflow



Example: Applying the Workflow

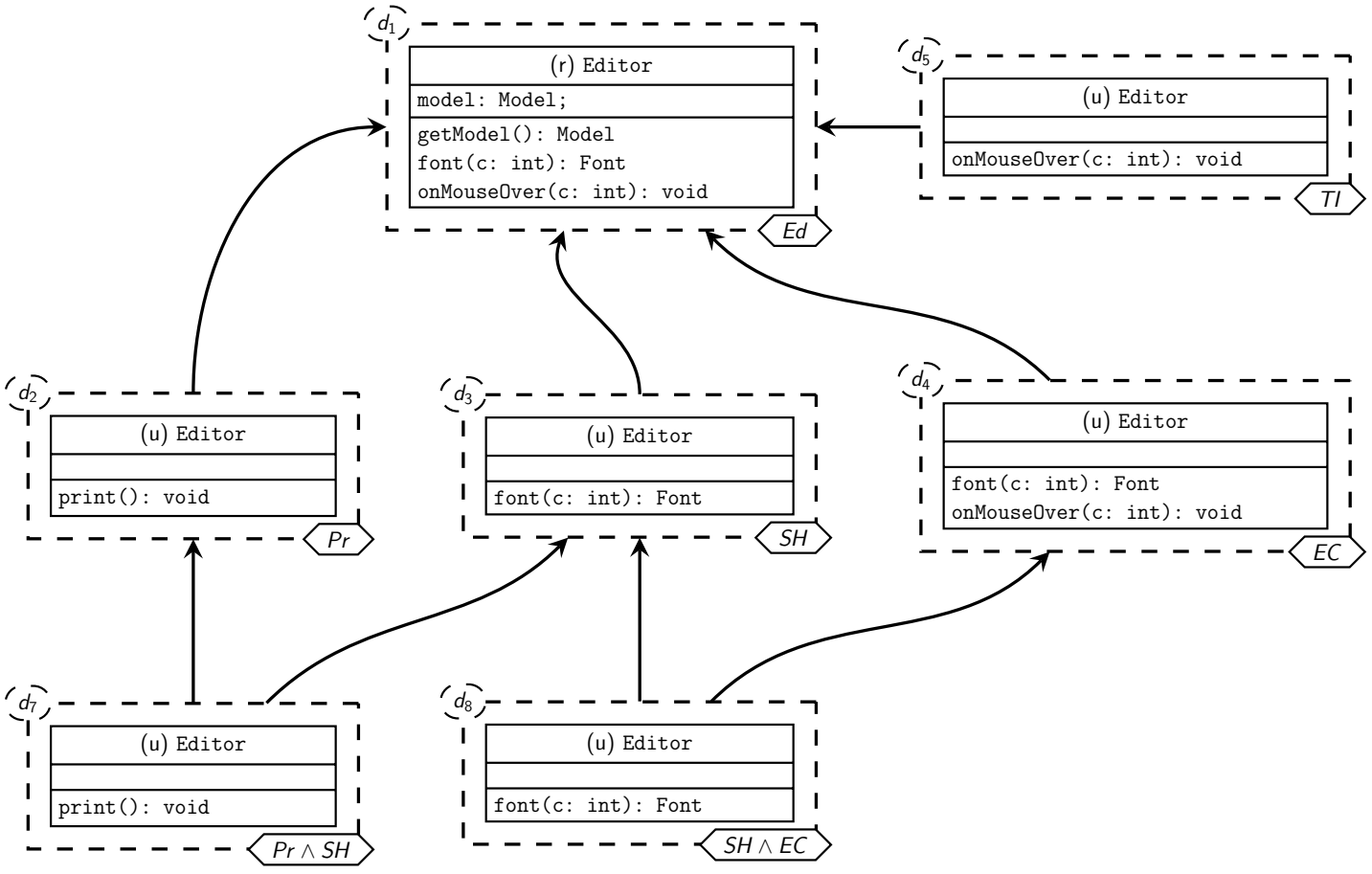


Example: Applying the Workflow

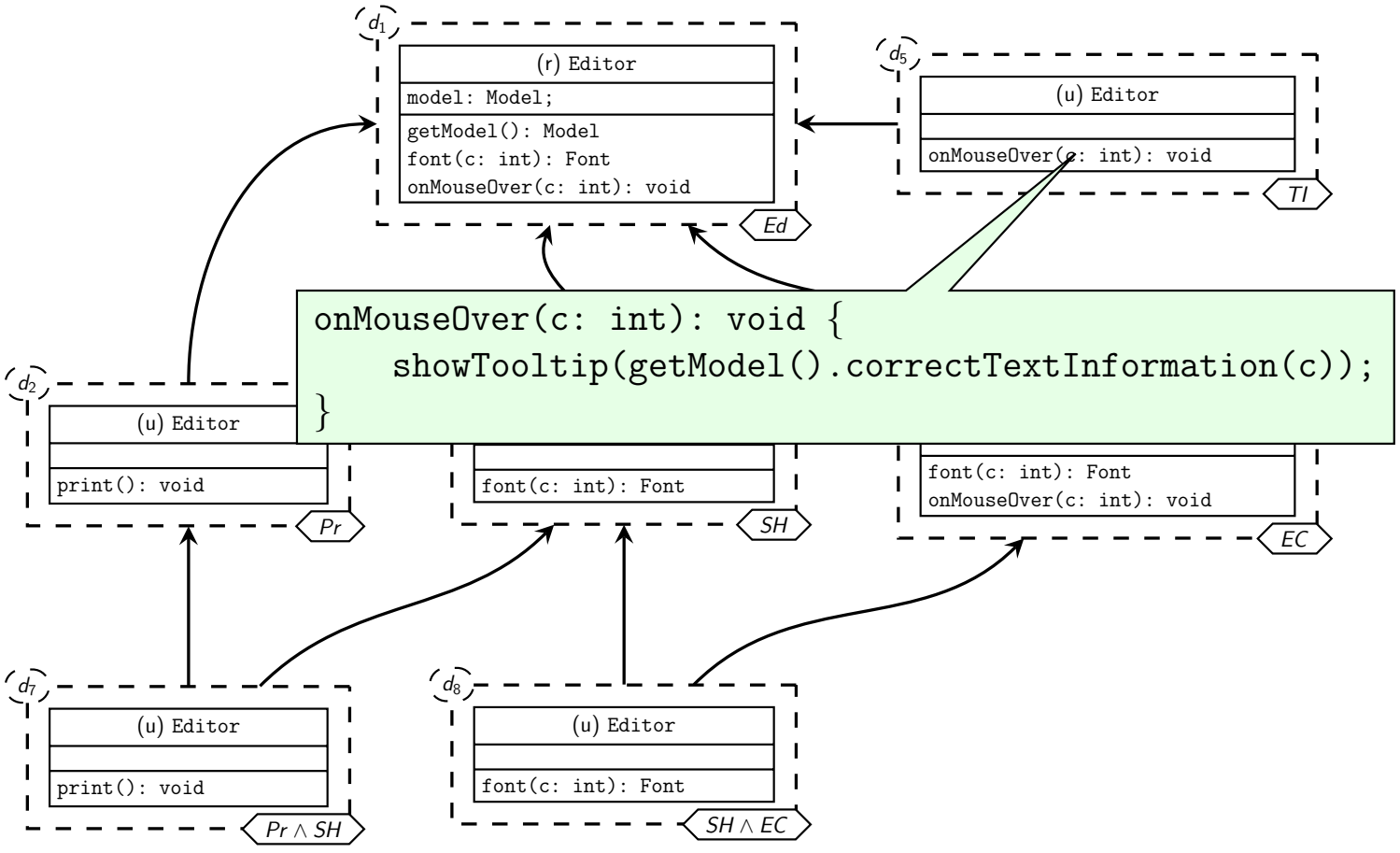



```
font(c: int): Font {
    result: Font;
    result.color = d3.font(c).color;
    result.underlined = d4.font(c).underlined;
    return result;
}
```

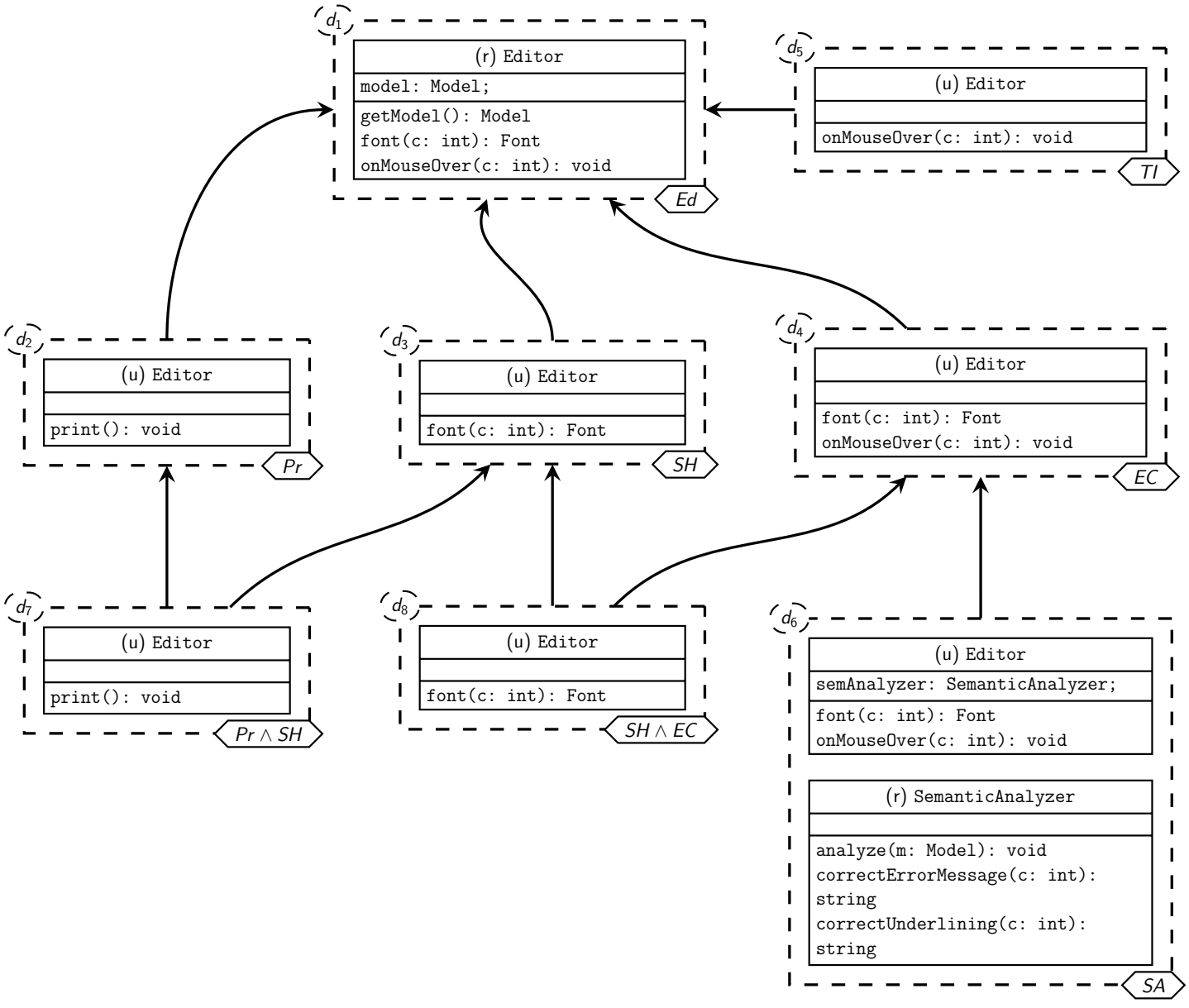
Example: Applying the Workflow



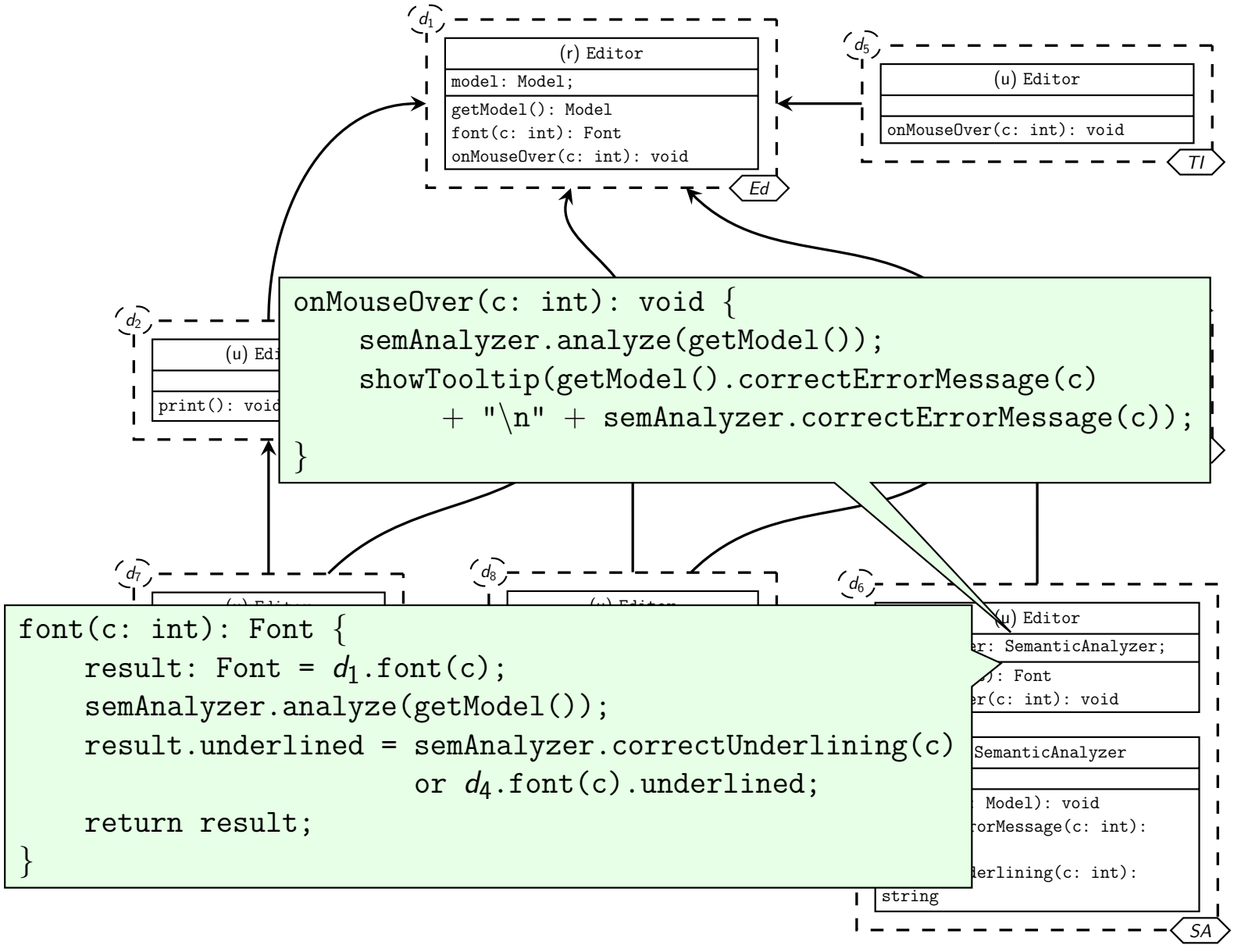
Example: Applying the Workflow



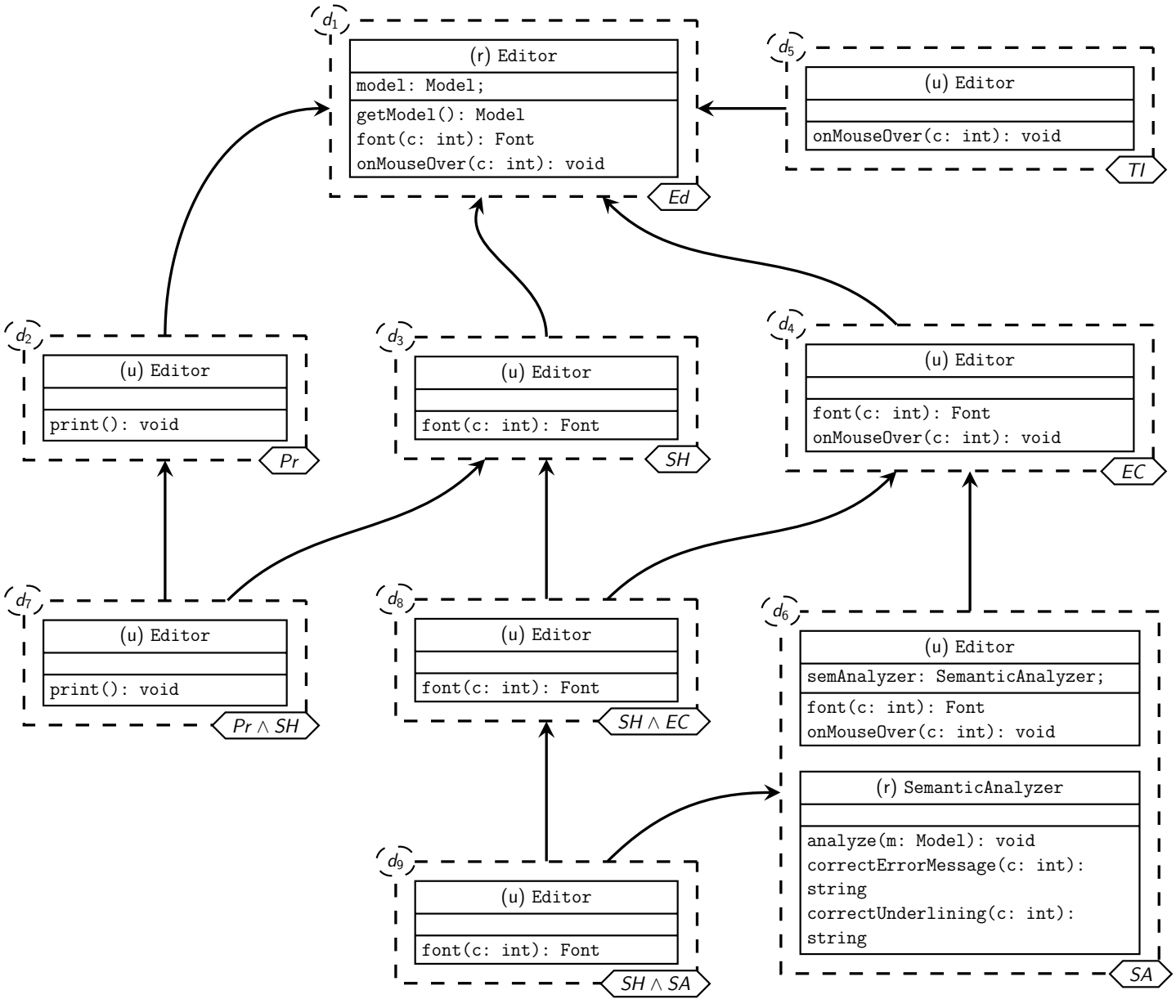
Example: Applying the Workflow



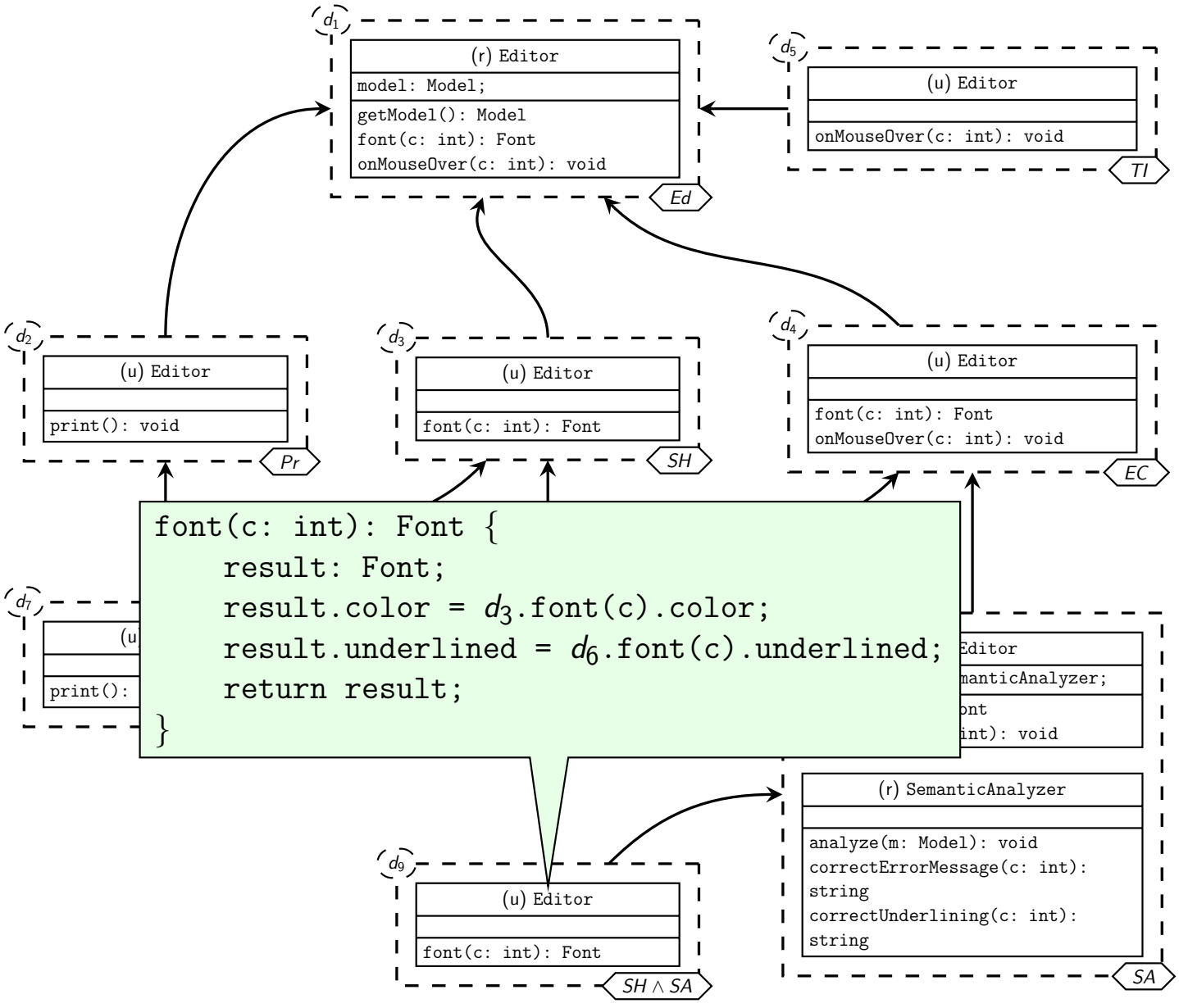
Example: Applying the Workflow



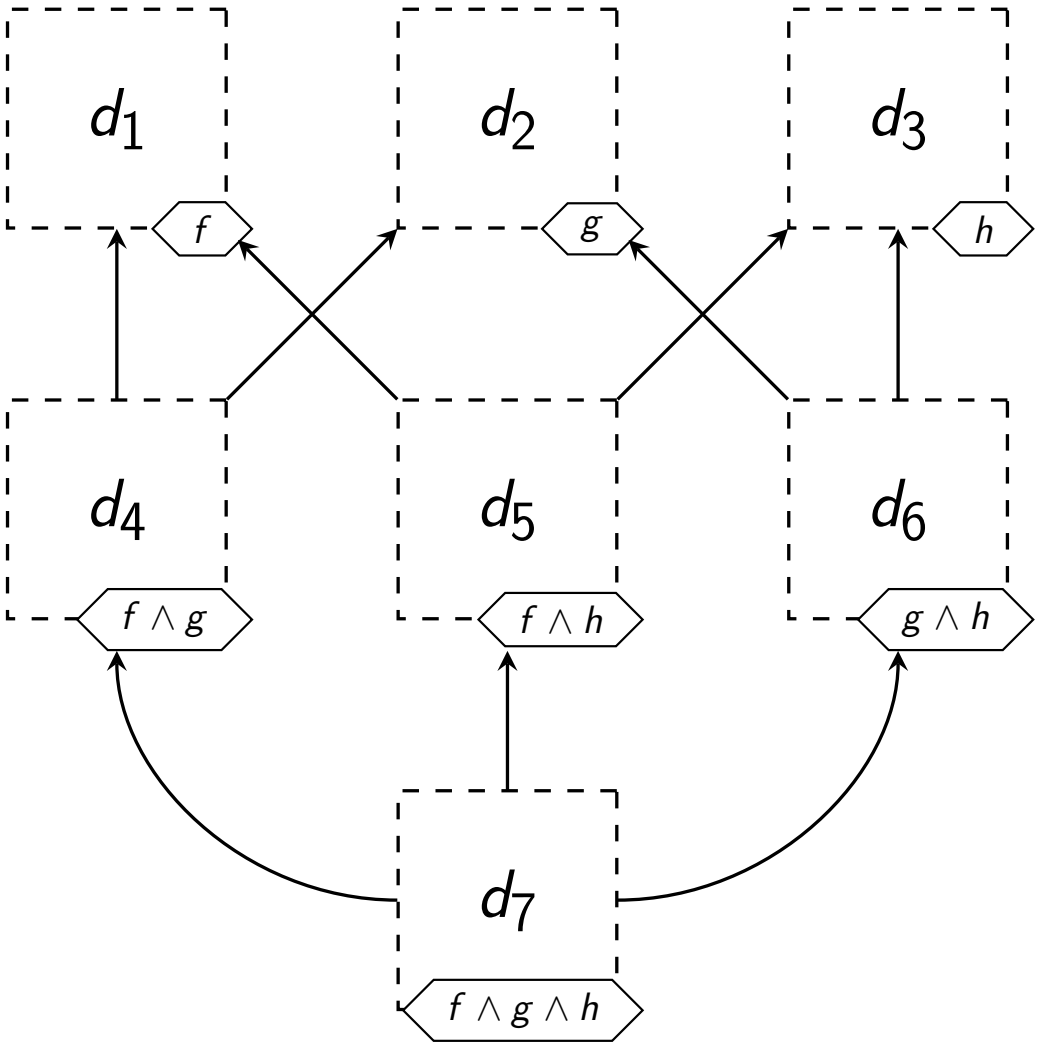
Example: Applying the Workflow



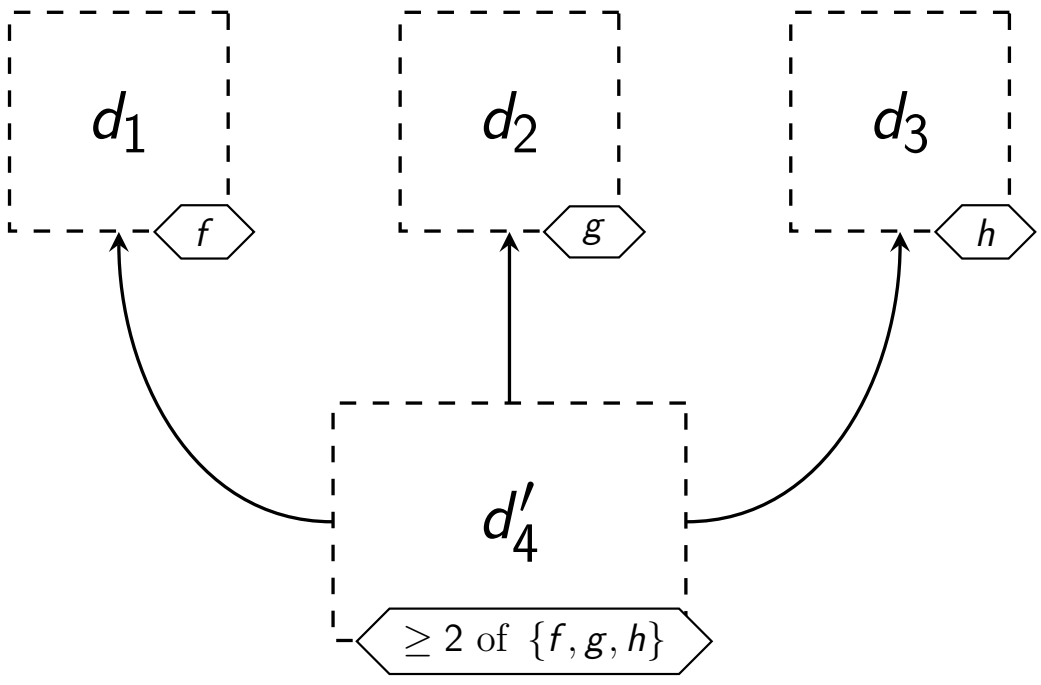
Example: Applying the Workflow



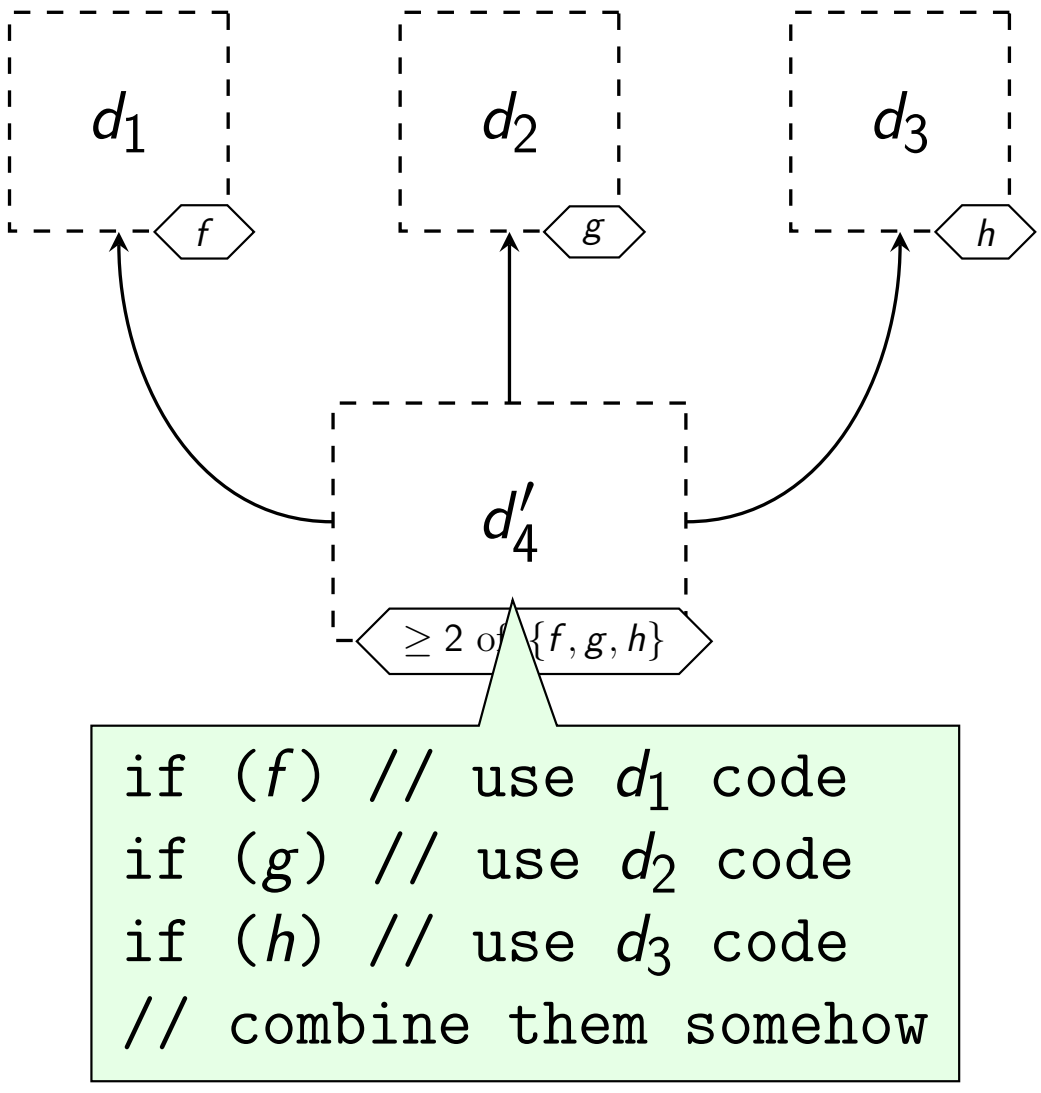
Parametrized Deltas



Parametrized Deltas



Parametrized Deltas



Conclusions and Future Work

Conclusions

- The Delta Modeling Workflow is a step-by-step guide of how to build a product line from scratch, based on Abstract Delta Modeling.
- The workflow allows concurrent and isolated development.
- The product lines resulting from the workflow have some nice properties:
  - Global Unambiguity (no unresolved conflicts)
  - Completeness (every feature (combination) is implemented)
  - Minimal code duplication and overspecification

Future Work

- Full formal proofs of properties
- Support for dynamically changing feature models / specifications