Delta Modeling Workflow

Michiel Helvensteijn

VaMoS, 26-01-2012

This is the presentation of the paper Delta Modeling Workflow as presented at VaMoS 2012. The paper itself has more information and can be found in the VaMoS 2012 proceedings.

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`

This talk and the talk about the "Delta Modeling in Practice" paper are related. This talk explains the basic theory and uses an example to illustrate the workflow. The other talk shows an industrial case study based on the workflow and sums up our observations on the process. The slides of both talks may be found at the URLs shown on this slide.

1 Features and Product Lines

Features and Product Lines

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

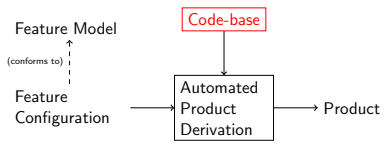
- Linux Kernel:
  - Loadable Module Support
  - Power Saving Support
  - Support for various hardware
- Payment System:
  - Chipknip Support
  - Creditcard Support
  - NFC Support
  - Cash Support
- Code Editor:
  - Syntax Highlighting
  - Error Checking
  - Printing

A software product line is basically a set of software products with well-defined commonality and variability. Each product is uniquely identified by the set of *features* it supports, called its *feature configuration*. This slide lists what may be considered three examples of software product lines and some of the features that their products can support.

Features and Product Lines

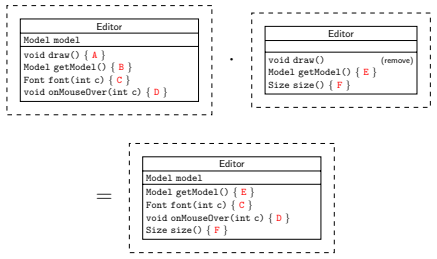
The code-base for an SPL should be organized in some way to reflect which features correspond to which code. This gives us:

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



The set of supported feature configurations is represented by a *feature model* (see section 3 for an example). *Automated product derivation* is the process of taking a code-base implementing a software product line and a valid feature configuration, and using an automated process to generate the corresponding software product. The challenge here is organizing the code-base with a clear link between features and behavior so that this can be done by a trivial composition of code while avoiding code duplication.

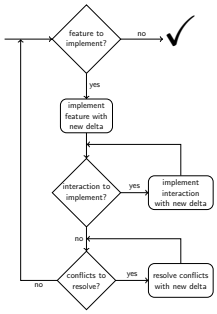
Deltas



In the concrete domain of object oriented programming, the basic building blocks of delta modeling (deltas) can add and overwrite methods and fields, as well as remove them. In this slide you can see an example delta being applied (with sequential composition operator ·) to an example product. The result is another product. This is how the different products in the product line are produced. Deltas are selected using application condition and ordered by a partial order. This is illustrated in the following example.

2 The Workflow

The Workflow



The workflow takes a feature model as input and produces a product line implementation as output. It a process illustrated by the flow-graph on this slide. Basically, features are implemented in a topological order from the feature diagram. So base features are implemented first, subfeatures later.

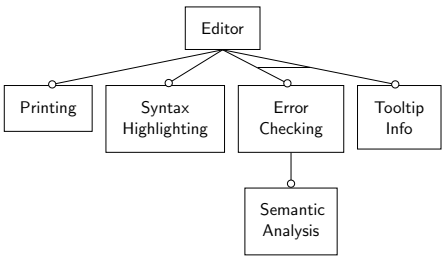
Workflow Properties

- 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.

This slide shows some of the important properties of the workflow. If it is properly followed, the resulting product line will be globally unambiguous. That is, every feature configuration yields a uniquely defined product. Every product of the product line will satisfy the specifications of its features, based on the developers guaranteeing this property locally. Independent features may be implemented concurrently and in isolation. Any desired interaction between features and required resolution of implementation conflicts can be collaboratively implemented later. Lastly, by virtue of the partial order between deltas, code duplication and overspecification will be at a minimum.

3 Example

Example: Feature Model



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

This slide presents an example feature model. It is of the Editor product line of code editors (such as may be found in Eclipse or Visual Studio). It offers support for the following features:

Editor (Ed) The mandatory base feature of the product line, which implements basic editing functionality.

Printing (Pr) The ability to print the editor content.

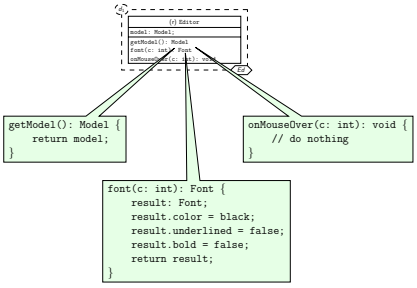
Syntax Highlighting (SH) Colors programming language constructs.

Error Checking (EC) Underlines syntactic errors and provides tooltips with error messages.

Semantic Analysis (SA) Underlines semantic errors and provides tooltips with error messages.

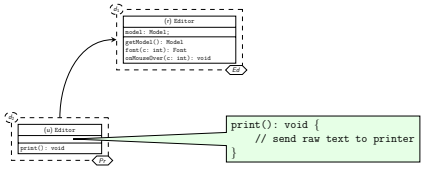
Tooltip Info (TA) Generic tooltip information.

Example: Applying the Workflow



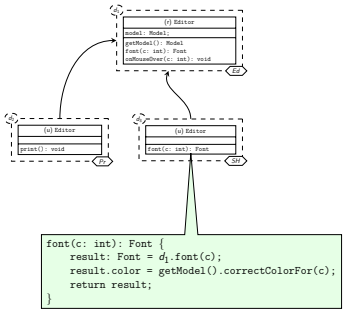
The mandatory Editor feature is implemented by the class shown in this slide. The `model` field and `getModel` method will not be changed by any deltas in this product line. The `font` method now specifies a default font with no decoration for every character of content. The `onMouseOver` method now does nothing when the mouse hovers over any character. These two methods will be modified later to implement the features of the product line.

Example: Applying the Workflow



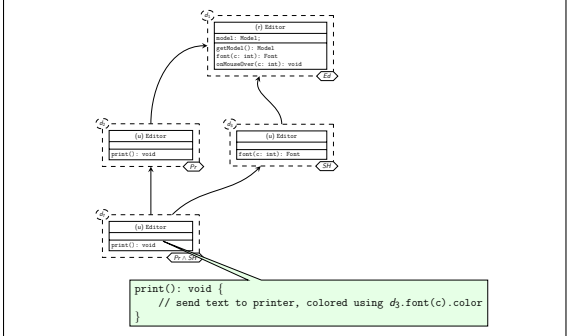
This delta implements the Printing feature. It adds the `print` method to the `Editor` class, which sends the content of the editor to a printer. It is applied to the core product whenever the Printing feature is selected. It is applied after the mandatory Editor delta.

Example: Applying the Workflow



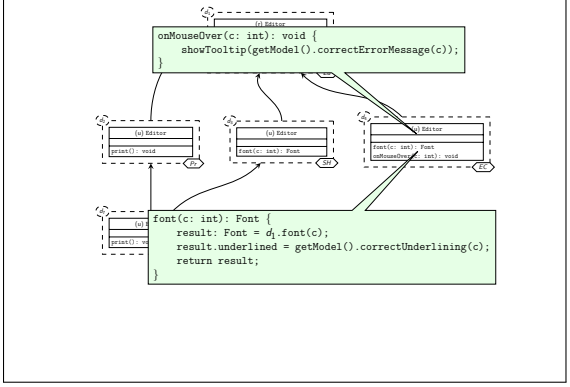
This delta implements the Syntax Highlighting feature. It overwrites the `font` method in order to properly set the color for every character of the content. It is applied whenever the Syntax Highlighting feature is selected, after the mandatory Editor delta.

Example: Applying the Workflow



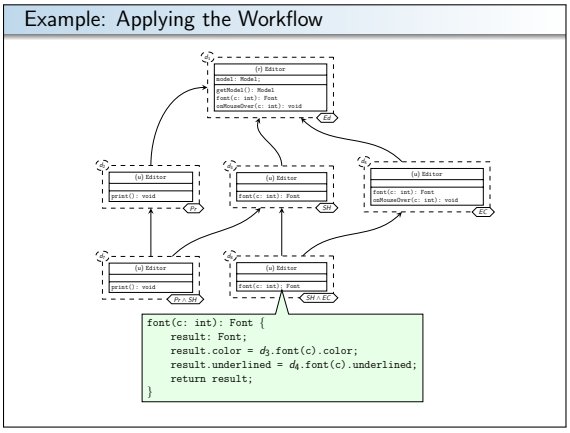
The Printing and Syntax Highlighting features can work together. Ideally, if both features are selected, we'd want any printout to contain the syntax highlighting colors as well. It is a feature interaction that we still have to implement. This is done by the delta in this slide, which is applied when both relevant features are selected, after the two basic feature implementation deltas, as we need to overwrite the `print` method and use the SH version of the `font` method.

Example: Applying the Workflow



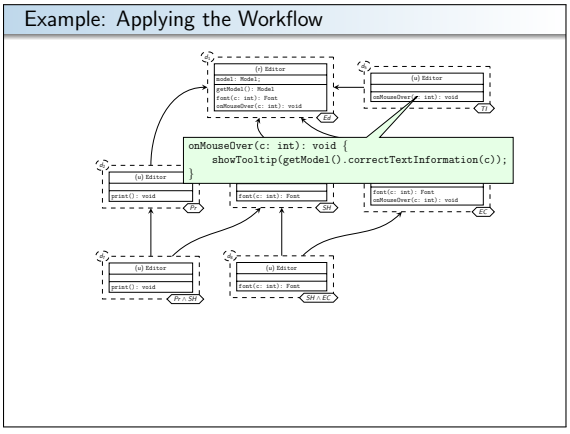
This delta implements the Error Checking feature. It modifies two methods. `onMouseOver` now shows a tooltip with an error message when you over erroneous code. So far so good. We also want to underline errors in the code. However, the base `Editor` class doesn't provide separate `color` and `underline` methods to overwrite. Only `font`. You might notice a conflict coming up. However, we disregard it and implement the `font` method as if the Syntax Highlighting feature doesn't exist (features can be implemented in isolation).

Example: Applying the Workflow



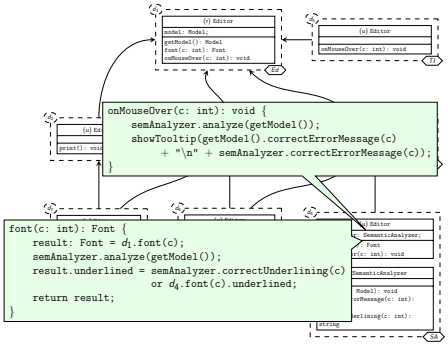
If both Syntax Highlighting and Error Checking are selected, there is a problem now. There are two possible linear orders in which to apply the relevant deltas, and depending on the order, the `font` method will be different. We need to write a delta to resolve this implementation conflict and properly combines both features. This is done by the delta on this slide. As you can see, we assume an underlying programming language in which we can target specific deltas and reuse their implementations. As such, we do not duplicate any code.

Example: Applying the Workflow



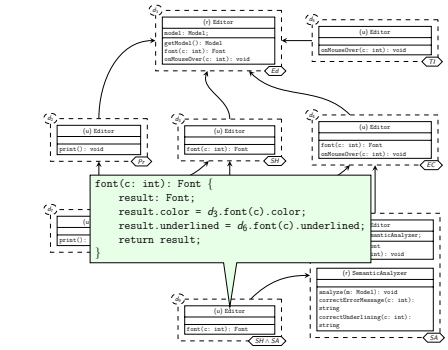
This delta implements the Tooltip Information feature. It shows some basic information (type, for instance) about the content that the mouse cursor is currently hovering over. At first glance it might appear that there is an implementation conflict between this delta and the delta implementing Error Checking. However, upon inspecting the feature model, one can see that those two features are mutually exclusive. They can never be selected together, so the two deltas are not in conflict.

Example: Applying the Workflow



This delta implements the subfeature of Error Checking: Semantic Analysis. It offers more sophisticated error reporting by overwriting both relevant methods. It is applied only when the Semantic Analysis feature is selected. By the feature model, this also implies inclusion of the base feature of Error Checking. It is applied after the Error Checking delta, so if it is selected, it overwrites the methods of that delta.

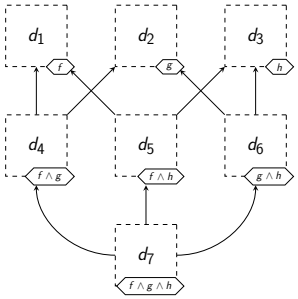
Example: Applying the Workflow



The last delta introduces an old conflict again. In order to allow all possible combinations from the feature model, we need to re-resolve this one. The delta on this slide does so, and completes our product line implementation.

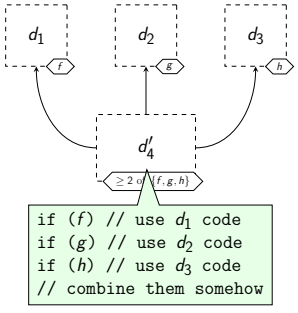
4 Parametrized Deltas

Parametrized Deltas



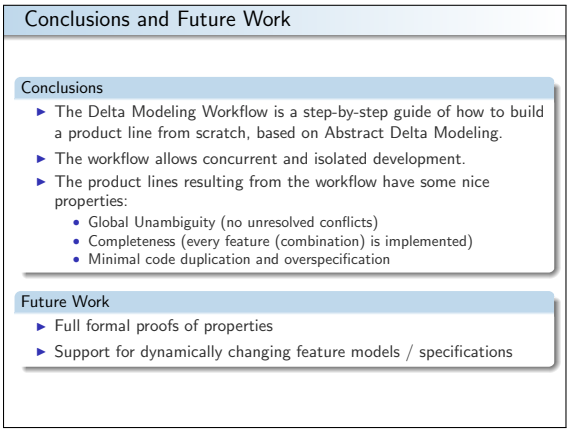
When following the workflow, we will often see conflicts being resolved, or interactions being implemented, layer upon layer, until all combinations are covered. This can lead to $2^n - 1$ deltas for n features. If all these combinations require a distinct solution, the structure shown in this slide is precisely what we need for full control.

Parametrized Deltas



By referencing old deltas, the model from the previous slide may not duplicate behavior, but it would duplicate boilerplate code, and would be tedious to write and maintain. Sometimes the variability can be much more conveniently expressed on the programming language level. For this, parametrized deltas may be used. The content of a delta would have access to the currently selected feature configuration, and use this, for instance, for conditional compilation. It is a tradeoff between the brevity of annotative approaches and the modularity of delta modeling. The combination of both is what make parametrized deltas powerful.

5 Conclusions and Future Work



This slide speaks for itself. For more information, we refer you to the paper.