Vrije
Universiteit
Brussel

Software variations by means of first-class change objects

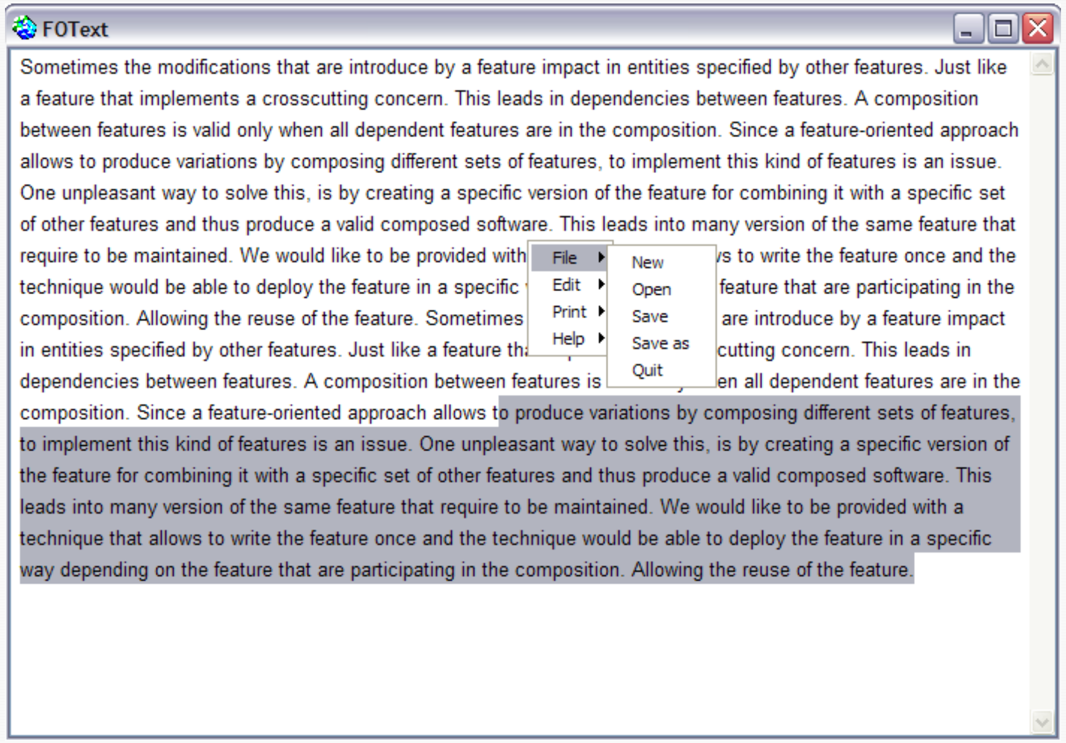
Peter Ebraert	pebraert@vub.ac.be
Leonel Merino	**leonelmerino@gmail.com**
Theo D'Hondt	tjdhondt@vub.ac.be

Agenda

- Software Variability
- Requirements
- Related work
- Discussion
- Our FOP model
- Demo
- Future work
- Conclusions

Software Variability

- FOText is a word processor that requires to be improved with a functionality to compress the files it produce.
- FOText has two variations:
  - FOText viewer
  - FOText full



Sometimes the modifications that are introduce by a feature impact in entities specified by other features. Just like a feature that implements a crosscutting concern. This leads in dependencies between features. A composition between features is valid only when all dependent features are in the composition. Since a feature-oriented approach allows to produce variations by composing different sets of features, to implement this kind of features is an issue. One unpleasant way to solve this, is by creating a specific version of the feature for combining it with a specific set of other features and thus produce a valid composed software. This leads into many version of the same feature that require to be maintained. We would like to be provided with
s to write the feature once and the
technique would be able to deploy the feature in a specific
feature that are participating in the
composition. Allowing the reuse of the feature. Sometimes
are introduce by a feature impact
in entities specified by other features. Just like a feature th
cutting concern. This leads in
dependencies between features. A composition between features is
en all dependent features are in the
composition. Since a feature-oriented approach allows to produce variations by composing different sets of features, to implement this kind of features is an issue. One unpleasant way to solve this, is by creating a specific version of the feature for combining it with a specific set of other features and thus produce a valid composed software. This leads into many version of the same feature that require to be maintained. We would like to be provided with a technique that allows to write the feature once and the technique would be able to deploy the feature in a specific way depending on the feature that are participating in the composition. Allowing the reuse of the feature.

Approach

- Ad-hoc: add the code needed directly into the application. ✗
  - Solution tightly coupled
- Feature-oriented programming: create a feature that adds this functionality to the FOText base program. ✓
  - Features can be reused

FOP requirements

We identify the following requirements:

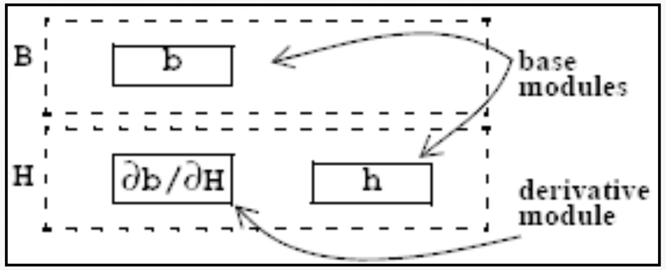
- The compress feature requires ***adding*** new statements and ***deleting*** existing ones in the *open* and *save* functionalities.

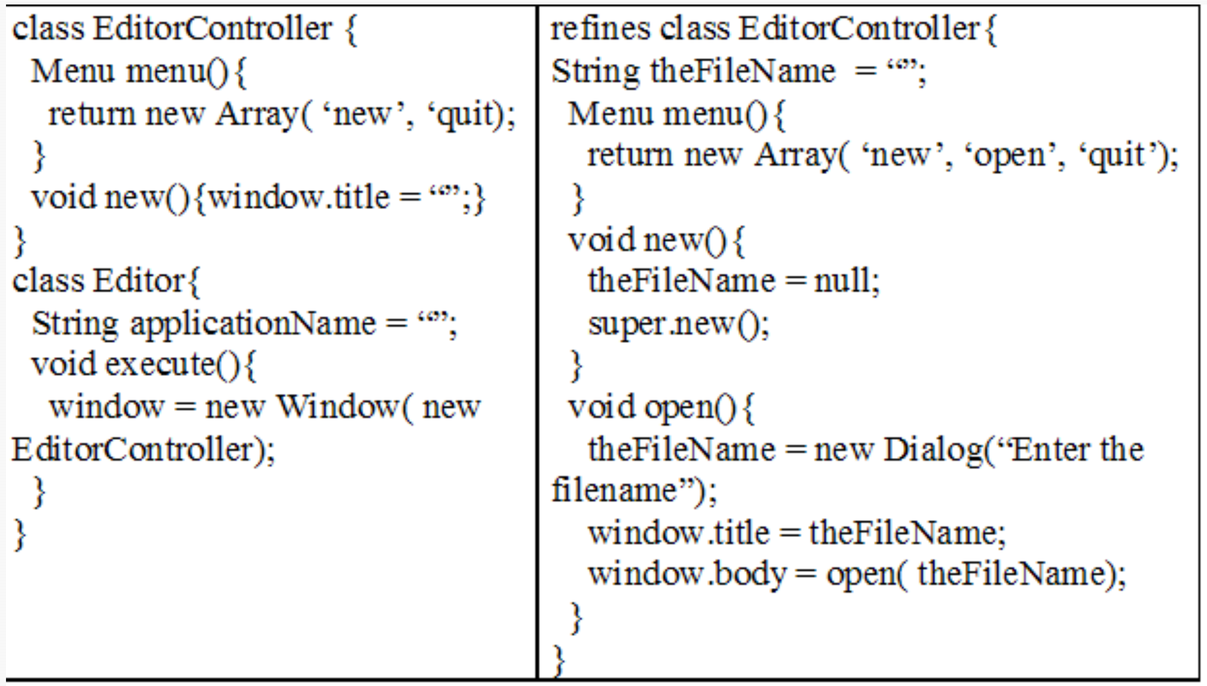
- Although we have two variations of FOText we would like to create the *compress feature* just once and **reuse** it.

We pursue an approach that fulfills these criteria

Related work AHEAD *

- AHEAD addresses FOP by providing: step-wise development, generative programming and algebras.





```
class EditorController {
  Menu menu(){
    return new Array( 'new', 'quit);
  }
  void new(){window.title = "";}
}
class Editor{
  String applicationName = "";
  void execute(){
    window = new Window( new
EditorController);
  }
}
```

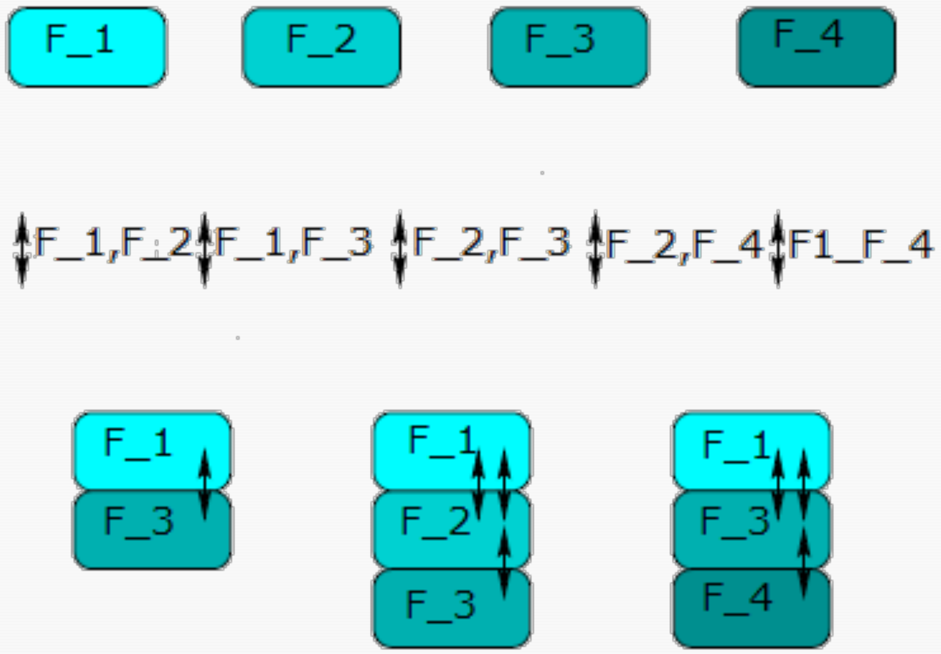
```
refines class EditorController{
String theFileName  = "";
  Menu menu(){
    return new Array( 'new', 'open', 'quit');
  }
  void new(){
    theFileName = null;
    super.new();
  }
  void open(){
    theFileName = new Dialog("Enter the
filename");
    window.title = theFileName;
    window.body = open( theFileName);
  }
}
```

(a) Base (c)Open

* D. Batory, J. Sarvela, and A. Rauschmayer. Scaling stepwise refinement, 2003.

Related work Lifting functions *

- This model allows flexible composition of objects from a set of features.
- Lifting functions resolve the interactions between features.



* Christian Prehofer. Feature-Oriented Programming: A Fresh Look at Objects. *Lecture Notes in Computer Science*, 1241:419{434, 1997.

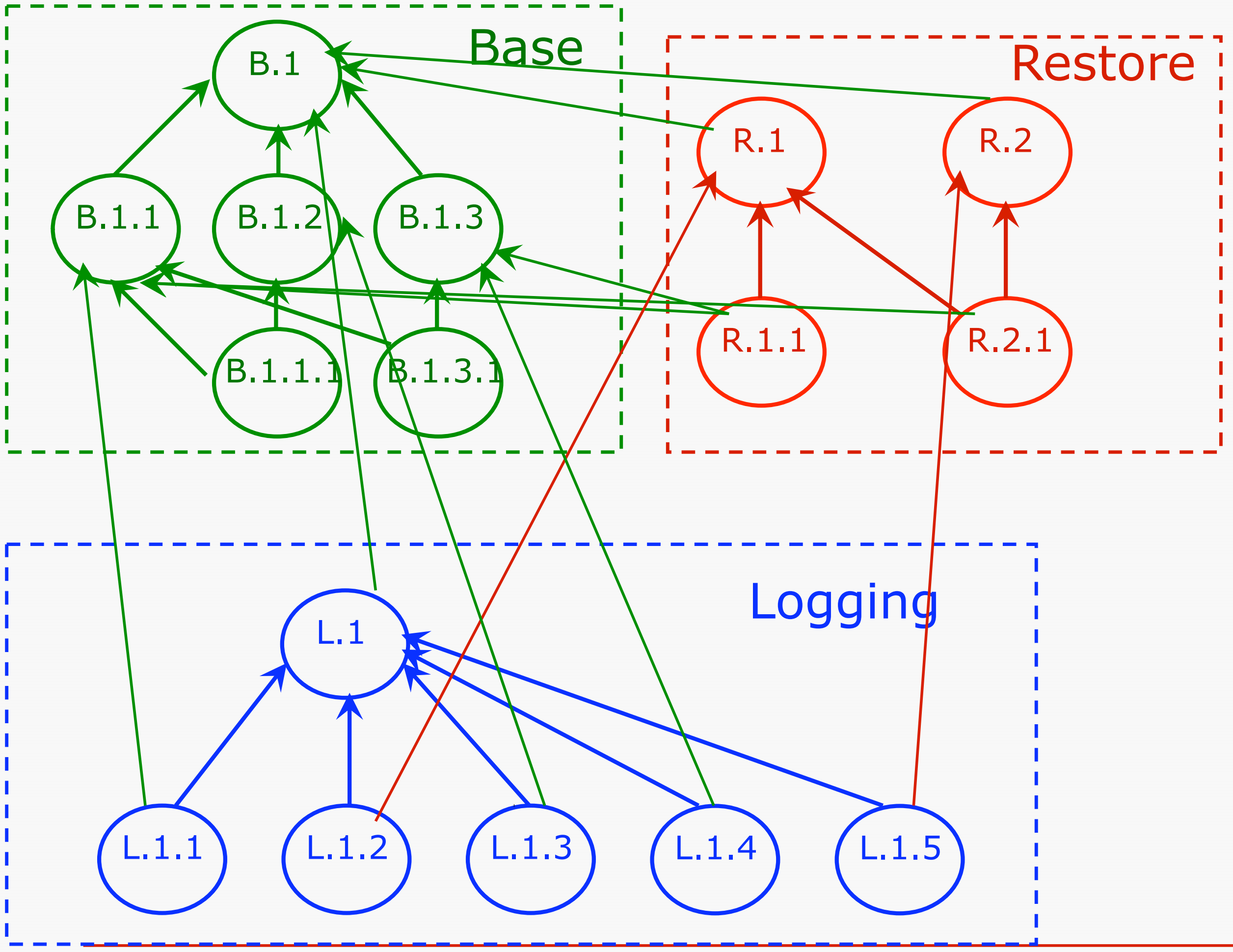
Discussion

	AHEAD	Lifting Functions	FeatureC++	Mixin-layers
Operations allowed	Addition and modification *	Addition	Additions and modifications	Additions and modifications
Granularity	Method, field and statement *	Statement	Method, field and statement *	Statement
Feature reuse	No	No	No	No
Dependency management	Implicit	Implicit	Implicit	Implicit

A change based solution for FOP

- ***Change:*** any operation produced by the programmer in the code.
- Feature as a set of first-class change objects.
- Changes are captured from the IDE using the ChEOPS tool.
- Explicitly stored dependencies between changes
- We aim at feature reuse

FOP model



```
B.1      class Buffer{
B.1.1      int buf = 0;
R.1        int back = 0;
B.1.2      void get(){
L.1.3            logit();
B.1.1.1          return buf;
           }
B.1.3      int set( int x ){
L.1.4            logit();
R.1.1            back = buf;
B.1.3.1          buf = x;
           }
R.2        void restore(){
L.1.5            logit();
R.1.2            buf = back;
           }
L.1        void logit(){
L.1.1            print( buf );
L.1.2            print( back );
           }
         }
```

Demo

Future work

- Develop a characterization for flexible features
- Not only allowing addition and deletion but modifications would decrease the number of changes.
- In ChEOPS the changes cannot be exported or applied*

Conclusion

- Programming languages do not provide enough tools to do FOP.
- We propose:
  - A conceptual model where features are described by changes and dependencies are explicit. Moreover, we introduce flexible features.
  - Tool support to compose features.

Thanks !

Software variations by means of first-class change objects

leonelmerino @ gmail.com