Feature-Oriented Programming with Family Polymorphism

Fuminobu Takeyama
Shigeru Chiba

Tokyo Institute of Technology
Feature-oriented programming (FOP)

- modularizes code snippets related to the same feature
  - useful for software product lines (SPL)

- A feature = aspects + classes
  - To improve composability
    - *i.e.*, easy to add/remove code snippets related to a feature
  - In AHEAD, a typical FOP language, refinement + classes

- Selecting a feature
  - Aspects and classes related the feature is compiled
MobileMedia SPL [T. Young, et al., AOSD 2005 demo]

- Multimedia viewer for mobile devices

- Alternative feature
  - original: select 1 feature from alternative features
  - select a subset of features from alternative features
Code clones among features

- Code clones are found especially in
  - alternative features
  - derivatives

- These clones cannot be removed easily
  - Not only classes but also aspects
  - Class names in those clones are different
Because alternative feature have similar behavior and structures of classes

- Both Photo and Music have a list of media
  - show/play a selected medium
Clones among alternative features 2/2

Photo

```java
reviser PhotoTypeInitializer
    extends Application {
    private PhotoListScreen screen;
    private PhotoController cont;
    void startApp() {
        screen = new PhotoListScreen();
        cont = new PhotoController();
        super.startApp();
    }
}

class PhotoController ext... Cont... {
    boolean handleCommand(Command c) {
        if (c == OPEN) {
            String sel = getSelected();
            Display.setCurrent(
                new PhotoViewScreen(sel));
        } else if (...) { ... } 
    }
}

class PhotoViewScreen ext... Screen {...
```
Code clones among derivatives 1/2

- A derivative
  - a special feature needed only when a certain set of features are selected

- PhotoCopy derivative
  - needed only when Photo and Copy are selected
  - Adding “Copy” to PhotoListScreen

- Photo
  - only for Photo

- PhotoCopy

- Copy
  - only for Copy
Different derivative for every combination

- The number of derivative might explode

```
reviser AddCopyToPhoto extends PhotoListScreen {
    void initMenu() {
        addCommand(new CopyCommand());
    }
}
```

```
reviser AddRenameToMusic extends MusicListScreen {
    void initMenu() {
        addCommand(new RenameCommand());
    }
}
```
The cause of these clones

Our observation is there are two types of edges
- It is said that an edge represents a \textit{has-a} relation
- Some edges normally represent \textit{is-a}
Avoiding code clones with inheritance

- A feature can be implemented by extending another
  - Common codes can be removed into super feature

- A generic derivative
  - A super feature is used as interface for its sub features
Family polymorphism with aspects

- A family is regarded as a feature
  - consist of aspects and virtual classes

- A virtual class
  - can be overridden by classes of sub feature
    - The real class referred by a virtual class is depends on feature
  - reduce difference of class names
    - in classes and aspects
Feature modules in FeatureGluonJ

Feature

- classes belong to one feature
  - declare at the top of source code

```java
feature Photo {
}

feature Photo;
class PhotoController ... {
    boolean handleCommand(Command c) {
        /* (snip) */
    }
}

feature Photo;
class PhotoViewScreen ... {
    /* (snip) */
}
```
PhotoFeature implemented with inheritance

- Use **overrides**, not **extends**
  - X overrides Y: X is replaced with Y that extends original X

```
abstract feature MediaType {} /* (snip) */

abstract class MediaTypeController ... {
    boolean handleCommand(Command c) {
        ... new MediaViewScreen(); ...
    }
}

abstract class MediaViewScreen {...}

feature Photo extends MediaType {}

class PhotoController overrides MediaTypeController {...}

class PhotoViewScreen overrides MediaViewScreen {...}
```

MediaViewScreen is overridden by PhotoViewScreen
Family polymorphism with revisers 1/2


- around advices with an execution pointcut
- intertype declarations
- can belong to a feature

```java
reviser MediaTypeInitializer extends Application {
    ...
    private MediaController cont;
    ...
    void startApp() {
        cont = new MediaController();
        ...
        super.startApp();
    }
}
```

```
class Application {
    void startApp() {
        // executed when this application starts
    }
}
```
No need to define the revisers for each feature

- Virtual classes in a reviser are overridden as well
Copy semantics 1/2

- Classes and revisers of super feature are copied to sub-features
  - visible only in the features

- Selecting multiple feature from sub features
  - multiple copies of a reviser will be executed.
Copy semantics 2/2

Photo

```java
reviser MediaTypeInfoMediaController cont;
    ... new MediaController(sel) ...
}
class PhotoViewScreen

Music

```java
reviser MediaTypeInfoMediaController cont;
    ... new MediaController(sel) ...
}
class MusicPlayerScreen
Implementation of naive derivatives

- Derivatives represents connection between features

- 2 steps to access classes belonging other features
  - `import feature`
  - Feature qualified class access, ::

```java
feature PhotoCopy {
  import feature p: Photo;
  import feature c: Copy;
}
```

```java
reviser AddCopyToPhoto extends p::PhotoListScreen {
  void initMenu() {
    addCommand(new c::CopyCommand());
  }
}
```

Derivative clones are still remaining

- Copy's CopyCommand
A generic derivative

- used to implement other derivatives
  - A super feature is used as interface
    - What virtual classes the feature provides?
  - import feature will be overridden by sub-derivative

---

**MediaTypeFileOp**

```java
feature MediaTypeFileOp {
  abstract import feature mt: MediaType;
  abstract import feature fo: FileOperation;
}
```

**reviser AddCmdToMediaList extends mt::MediaListScreen**

```java
void initMenu() {
  addCommand(new fo::FileOpCommand());
}
```
Derivatives extending the generic derivative

- Code clones are removed
  - Redundant feature definitions are still remaining

```java
feature MediaTypeFileOp {  
    abstract import feature mt: MediaType;  
    abstract import feature fo: FileOperation;  
}  

reviser AddCmdToList extends mt::MediaListScreen {  
    void initMenu() {  
        addCommand(new fo::FileOpCommand());  
    }  
}
```

```java
feature PhotoCopy extends MediaTypeFileOp {  
    import feature mt: Photo;  
    import feature fo: Copy;  
}
```

```java
feature PhotoRename extends MediaTypeFileOp {  
    import feature mt: Photo;  
    import feature fo: Rename;  
}
```

Overrides alias with the same name

A reviser is derived
Auto generation of the derivatives

- derivatives extending the generic one
  - no need to create new derivatives when a new feature is added

```java
feature MediaTypeFileOp defines forevery(mt, fo) {
    abstract import feature mt: MediaType;
    abstract import feature fo: FileOperation;
}

reviser AddCmdToMediaList extends mt::MediaListScreen {
    void initMenu() {
        addCommand(new fo::FileOpCommand());
    }
}
```
Related work 1/2

Lightweight family polymorphism

- Similar semantics from the view point of virtual class
  - Virtual classes (and revisers) can be described only in top level
  - No subtyping between classes from different family

CaesarJ

- language supporting virtual classes and aspects
- Aspects in CaesarJ are limited
  - No intertype declarations
  - (?) cannot handle join points in other feature (cclass)
    - Does anyone know the detail of CaesarJ’s semantics?

[I. Aracic, et al., TAOSD I 2006]
Related work 2/2

- FOP only with virtual classes
  - needs private inheritance to prevent the code clones

[V. Gasiunas, et al., AO Product Line Eng.]
Conclusions

FeatureGluonJ

- family polymorphism / virtual classes for FOP
  - supports aspects
- removes code clones
  - among alternative features
  - among derivatives

Future work

- Formal definition of type system
- Other types of clones
  - code for resolving conflict of revisers