Do We Really Need to Extend Syntax for Advanced Modularity?

Shigeru Chiba¹, Michihiro Horie², Kei Kanazawa, Fuminobu Takeyama, Yuuki Teramoto

Tokyo Institute of Technology, Japan

¹Also, The University of Tokyo and JST CREST
²Currently, IBM Research Tokyo
Advanced modularity

Objectives

- Code reuse
  - Avoid code clones (duplicated code)
- Composability
  - Build software by combining modules as is (without modifications)
- Grouping related code
  - for spatial locality per concern
- Information hiding
  - Give higher-level abstraction
Observer pattern in Java

```java
class Line extends Shape {
    int angle, len;

    void setPos(int nx, int ny) {
        x = nx; y = ny;
        DisplayUpdate.after(this);
    }

    void setLength(int nlen) {
        len = nlen;
        DisplayUpdate.after(this);
    }

    int getLength() { return len; }
    /* (snip) */
}

class Rectangle {
    void setWidth(int nw) {
        width = nw;
        DisplayUpdate.after(this);
    }
}
```

Crosscutting concern
The display-update concern is in an aspect.

- Good composability \(i.e.\) easily removable
A code snippet belongs to multiple concerns.

Original

```java
class Line extends Shape {
    void setPos(int nx, int ny) {
        x = nx; y = ny;
        DisplayUpdate.after(this);
    }

    void setLength(int nlen) {
        len = nlen;
        DisplayUpdate.after(this);
    }

    // (snip)
}
```

Class + aspect

```java
class Line extends Shape {
    void setPos(int nx, int ny) {
        x = nx; y = ny;
        // (snip)
    }

    void setLength(int nlen) {
        len = nlen;
        // (snip)
    }

    // (snip)
}

aspect DisplayUpdate {
    // (snip)
    after(Shape s): change(s) {
        s.display().repaint(s);
    }
}
```

Cannot see the entire Line concern!!
Another Limitation of AspectJ: Code reuse

- Cannot avoid code duplication

```java
void around(Line line, int x, int y):
  execution(void Shape+.setPos(int, int)) && this(line) && args(x, y) {
    if (line.x != x || line.y != y) {
      proceed(line, x, y);
      line.display().repaint(line);
    }
  }

void around(Line line, int len):
  execution(void Line.setLength(int)) && this(line) && args(len) {
    if (line.len != len) {
      proceed(line, len);
      line.display.repaint(line);
    }
  }
```

// (snip)
Meta-helical contextual predicate dispatch

An extension to contextual predicate dispatch
[Chiba, et al, OOPSLA’10]

```java
void around anySetter(Line line, Method setter, Object[] args):
    execution(void Shape+.setPoint(int,int)
        || void Shape+.setPoint.setLength(int))
    && this(line) => $setter($args)

boolean around posSetter(int x, int y): within anySetter
    execution(void Shape+.setPos(int,int)) && args(x, y)
{ line.x != x || line.y != y } => $conditionExpr

boolean around lenSetter(int len): within anySetter
    execution(void Shape+.setLength(int)) && args(len)
{ line.len != len } => $conditionExpr

void setter(args) within anySetter && (posSetter || lenSetter) {
    super.setter(args);
    if (conditionExpr)
        line.display().repaint(line);
}
```
End of Skit

Developing a complex language.

This is our typical research, but is this a right approach?

(The last slide is fake)
Do We Really Need to Extend Syntax for Advanced Modularity?

Shigeru Chiba¹, Michihiro Horie², Kei Kanazawa, Fuminobu Takeyama, Yuuki Teramoto

Tokyo Institute of Technology, Japan

¹Also, The University of Tokyo and JST CREST
²Currently, IBM Research Tokyo
Programs expressed in dynamic text

- Dynamic text
  - changes its appearance while editing and browsing
    - like web page written in HTML + JavaScript
  - Traditional languages uses only static text.

- Two preliminary systems for implementing this idea.
Programs expressed in dynamic text

- Live text
  - changes its appearance while editing and browsing
    - like web page written in HTML + JavaScript
  - Traditional languages uses only static text.

- Two preliminary systems for implementing this idea.
Copy & paste

- The easiest approach to code reuse
  - actually provides good modularity!
- Dynamic text resolves its maintainability problem.

We love copy & paste when we were beginners
Synchronous copy & paste

- Synchronizes code clones
- Automatically updates synchronized clones when one of them is updated
The text in the holes is not synchronized

- Can synchronize somewhat different code clones
It’s like a procedure or an advice!

- Synchronized clones look like a procedure
  - Holes are parameters

- Better reusability
  - resolves the problem of AspectJ

```java
public class Line extends Shape {
    int angle, len;
    void setPos(int x, int y) {
        if (this.x != x || this.y != y) {
            this.x = x; this.y = y;
            this.display().repaint(this);
        }
    }
    void setLength(int len) {
        if (this.len != len) {
            this.len = len;
            this.display().repaint(this);
        }
    }
    void setAngle(int angle) {
        if (this.angle != angle) {
            this.angle = angle;
            this.display().repaint(this);
        }
    }
    int getLength() { return len; }
}
```

```java
void updateDisplay(cond, do) {
    if (cond) {
        do;
        this.display().repaint(this);
    }
}
```
Shows a group of code synchronized with each other.

- provides the same information that AspectJ pointcuts do.
- actual parameters are displayed.
Browsing related code at a glance

- Developers have to open each file
Browsing related code at a glance

- Developers have to open each file

My office mate loves dual 30inch displays
5120 x 1600px
A virtual source file

- collects related snippets from many source files

DisplayUpdate.kide

```java
void setPos(int nx, int ny) {
    x = nx; y = ny;
    DisplayUpdate.advice(this);
}

void setLength(int nlen) {
    len = nlen;
    DisplayUpdate.advice(this);
}

void setWidth(int nlen) {
    len = nlen;
    DisplayUpdate.advice(this);
}
```

Physical source files

A virtual source file
A virtual source file

- Developers can edit as a normal source file

```java
/* /demo/src/figureEditor/Rectangle.javasetHeight */
void setHeight(int nh) {
    height = nh; DisplayUpdate.advice(this);
}

/* /demo/src/figureEditor/Rectangle.javasetWidth */
void setWidth(int nw) {
    width = nw; DisplayUpdate.advice(this);
}

/* /demo/src/figureEditor/Shape.javasetPos */
void setPos(int nx, int ny) {
    x = nx; y = ny;
    DisplayUpdate.advice(this);
}
```

A comment indicating physical source file

From Rectangle.java

From Shape.java
Selecting code snippets for a virtual source

- by Wizard-like UI
  - no pointcut language
  - supports to select all methods calling a specific method
    - like AspectJ’s call pointcut

- A virtual source files is saved as part of a project
  - listed in concerns view
Better grouping than AspectJ

- From the viewpoint of the Line concern
  - the same presentation as in pure Java

```java
class Line extends Shape {
    void setPos(int nx, int ny) {
        x = nx; y = ny;
        DisplayUpdate.after(this);
    }

    void setLength(int nlen) {
        len = nlen;
        DisplayUpdate.after(this);
    }

    int getLength() {
        return len;
    }

    // (snip)
}
```
Better grouping than AspectJ (cont.)

- From other viewpoints

```java
class Rectangle extends Shape {
    void setWidth(int nw) {
        width = nw;
        DisplayUpdate.after(this);
    }
    // (snip)
}
```

A physical source file

```java
DisplayUpdate.kide
/* src/Line.java */
void setLength(int nlen) {
    len = nlen;
    DisplayUpdate.after(this);
}
/* src/Rectangle.java */
void setWidth(int nlen) {
    len = nlen;
    DisplayUpdate.after(this);
}
```

Virtual source files

```java
SetWidthHeight.kide
/* src/Circle.java */
void setWidth(int nw) {
    this.width = nw; /*(snip)*/
    DisplayUpdate.after(this);
}
/* src/Rectangle.java */
void setWidth(int nlen) {
    len = nlen;
    DisplayUpdate.after(this);
}
```
Crosscutting over documentation

- A virtual source can contain plain text from any text file
  - Literate programming

- Bug report
  - Kide helps to quote method declarations from source files

```java
@sun.reflect.DelegatingMethodAccessorImpl
@java.lang.reflect.Method.invoke
@javassist.util.proxy.FactoryHelper.toClass...
... 46 more

The performance is also very bad in comparison

Solution:
The following are the methods I changed for fix

```
Related work: code clone

- Tools for managing code clones
  - Simultaneous editing [R. C. Miller, et al., USENIX 2002]
  - Linked Editing [M. Toomim, et al., VLHCC 2004]
    - supports parameters
  - CReN [D. Hou, et al., CASCON 2009]

- These tools do not provide the concerns view
  - Sync. copy & paste does since its focus is on modularity.
Related work: aspect orientation

- **Mylar** [M. Kersten, et al., AOSD 2005]
  - monitors developers’ action and detects a group of related code
    - Temporary concerns
  - does no change the expression of source code

- **Fluid AOP** [T. Hon, et al., OOPSLA 2006]
  - can show a program differently from the representation in the actual source file
  - requires AspectJ-like syntax
    - Sync. Copy & paste and Kide does not
Related work: other systems

- **CIDE [C. Kästner, et al., ICSE 2008]**
  - Developers can mark a code snippet with a color of its feature
    - They can include/exclude all the code snippets related to a given feature
    - by a single action
    - without syntactic directive `#ifdef`
  - Its navigation panel is similar to our concerns view
  - It does not provide a virtual source file

- **Code Bubbles [A. Bragon, et al. ICSE 2010]**
  - A small editor dedicated for every method can be freely placed on the screen
    - can be used as an editor of a virtual source file
  - does not provide a wizard-based support tool
Approaches to modularity

- A language based on ...

static text
What is it suitable for?

Dynamic text
Synchronous copy & paste
Kide

2D figures and diagrams
Executable UML