Aspect-oriented Software Development

An Introduction

Johan Brichau

VUB Programming Technology Lab

Talk Contents

• Separation of Concerns
  – Broader Context of AOSD
• Aspect-orientation
  – Definition, History
• Ready-to-use Technologies
  – An overview
• Research activities
• AspectJ demo
Talk Contents

- Separation of Concerns
- Aspect-orientation
- Ready-to-use Technologies
- Research activities
- AspectJ demo

Separation of Concerns

- E.W. Dijkstra, Parnas
- Organize code according to common functionality
  - Modular programming
  - Need for programming language mechanisms
    - enforce encapsulation of module internals
    - provide module composition mechanisms
- Benefits:
  - Manage complexity
  - Fewer defects, easier to localize defects
  - Reusability
  - Ability to respond to market changes
Structured Programming

- Tangling from explicit gotos
- Recognized common control structures
  - capture in more explicit form
- Resulting code
  - more clear, easier to write, maintain, debug etc.

But… still tangled

```c
main () {
    draw_label("Haida Art Browser");
    m = radio_menu("Whale", "Eagle", "Dogfish");
    q = button_menu("Quit");
    while ( ! check_buttons(q) ) {
        n = check_buttons(m);
        draw_image(n);
    }
}
```

```c
radio_button(n) {
    draw_circle(get_x(), get_y(), 3);
    draw_circle(x, y, r) {
        primitive_oval(x, y, 1, r);
    }
    button_menu(labels) {
        i = 0;
        while ( i < labels.size ) {
            draw_label(labels[i]);
            set_y(get_y() + BUTTON_H);
        }
    }

    draw_label(string) {
        w = calculate_width(string);
        print(string, WINDOW_PORT);
        set_x(get_x() + w);
    }

    radio_menu(labels) {
        i = 0;
        while ( i < labels.size ) {
            radio_button(i);
            draw_label(labels[i]);
            set_y(get_y() + RADIO_BUTTON_H);
            i++;
        }
    }

    draw_image (img) {
        w = img.width;
        h = img.height;
        do (c = 0; c < w; c++)
            do (r = 0; r < h; r++)
                WINDOW[r][c] = img[r][c];
    }
```
Group functionality...

```cpp
main() {
    draw_label("Haida Art Browser");
    m = radio_menu(
        "(Whale", "Eagle", "Dogfish")
    );
    q = button_menu("Quit");
    while (!check_buttons(q)) {
        n = check_buttons(m);
        draw_image(n);
    }
}
```

```cpp
draw_label(string) {
    w = calculate_width(string);
    print(string, WINDOW_PORT);
    set_x(get_x() + w);
}
```

```cpp
draw_image(img) {
    w = img.width;
    h = img.height;
    do (r = 0; r < h; r++)
        do (c = 0; c < w; c++)
            WINDOW[r][c] = img[r][c];
}
```

```cpp
radio_menu(labels) {
    i = 0;
    while (i < labels.size) {
        radio_button(i);
        draw_label(labels[i]);
        set_y(get_y() + RADIO_BUTTON_H);
        i++;
    }
}
```

```cpp
radio_button (n) {
    draw_circle(get_x(), get_y(), 3);
}
```

```cpp
button_menu(labels) {
    i = 0;
    while (i < labels.size) {
        draw_label(labels[i]);
        set_y(get_y() + BUTTON_H);
        i++;
    }
}
```

...into Modules...

```cpp
main() {
    draw_label("Haida Art Browser");
    m = radio_menu(
        "(Whale", "Eagle", "Dogfish")
    );
    q = button_menu("Quit");
    while (!check_buttons(q)) {
        n = check_buttons(m);
        draw_image(n);
    }
}
```

```cpp
draw_image(img) {
    w = img.width;
    h = img.height;
    do (r = 0; r < h; r++)
        do (c = 0; c < w; c++)
            WINDOW[r][c] = img[r][c];
}
```

```cpp
draw_label(string) {
    w = calculate_width(string);
    print(string, WINDOW_PORT);
    set_x(get_x() + w);
}
```

```cpp
draw_circle (x, y, r) {
    primitive_oval(x, y, 1, r);
}
```

But: variations on modules are incredibly complex
...Object-orientation

- **Menu**
  - display: <promise>
  - click: <promise>

- **ToggleButtonMenu**
  - display: boxes, labels
  - click: highlight
  - click: set/clear

- **CheckBoxMenu**
  - display: boxes, maybe checks, labels
  - click: set/clear

- **RadioButtonMenu**
  - display: circle, maybe dots, labels
  - click: set/clear
  - AND clear others

---

But...

![Diagram of java.awt-components](image-url)
But... (example)

Where is logging in Apache web server?
- Not in one place
- Not even in a small number of places

Crosscutting Concerns

• Concern
  – ‘Something the programmer should care about’
  – Ideally implemented in one single module
• Crosscutting
  – Implementation is spread across other modules
  – Difficult to understand, change, maintain, etc...
• Tyranny of the Dominant Decomposition
  – Given one of many decompositions of the problem...
    • Mostly core functionality concerns
  – ...some subproblems (concerns) cannot be modularized
    • Non-functional, functional, added later on,...
Tangling -> Structure

structured control constructs
modules with narrow interfaces
classification & specialization of objects

Talk Contents

• Separation of Concerns
• Aspect-orientation
• Ready-to-use Technologies
• Research activities
• AspectJ demo
Aspect-orientation

• Modularize crosscutting concerns
  – Code scattering (one concern in many modules)
  – Code Tangling (one module, many concerns)

Examples?

• What is an Aspect? vs. What is an Object?
• Non-functional aspects
  – Synchronisation
  – Logging
  – Error Handling
  – Persistence
  – Real-time issues
• Functional aspects
  – Coloring of figures in a drawing application
  – User interface
  – Personalisation of E-commerce application
AO History

- **Reflection**
  - Meta-object protocol (MOP)
    - Control over method invocation, instance creation, etc…
  - Often used to implement crosscutting concerns
  - Considered too powerful and too difficult

- **AOSD**
  - Provide necessary abstractions to implement crosscutting concerns
    - Instead of all reflection possibilities
  - Is often implemented through MP and reflection

AO History

- **Domain-specific Aspect Languages**
  - Language targeted towards one kind of aspect
    - COOL & RIDL (Synchronisation)
    - RG (Loop fusion optimization)
  - describe a concern
  - Easy to understand
  - But
    - Limited in expressiveness
    - New Language for each kind of aspect??

- **General-purpose Aspect Languages**
  - More-general aspects possible
  - describe ‘crosscutting’
COOL example

```java
class Stack {
    push(...);
    pop(...);
    ...
}

coopordinator BoundedStackCoord {
    selfexclusive {pop,push};
    mutexclusive {pop,push};
    ...
}
```

Synchronisation code is separated using an aspect language with synchronisation primitives

AO Technologies

- Well known general AO technologies
  - AspectJ (Aspect-oriented Programming)
  - HyperJ (Multidimensional SoC)
  - DemeterJ (Tree traversals)
  - Composition Filters (Message adapters)
- Many more Academic Prototypes
  - JAC (Dynamic AOP in Java)
  - AspectC (AOP in C)
  - AspectS/Andrew/Soul (AOP in Smalltalk)
- Different crosscutting => different solution
AOP Terminology

- **Base program**
  - (Object-oriented) program

- **Aspect**
  - Modularization of a crosscutting concern

- **Weaver**
  - Composes (Compiles) the aspect into the base program

- **Joinpoint**
  - Particular point in the base program where the aspect can be woven

AOP Example

```java
class Buffer {
    char[] data;
    int nrElements;
    Semaphore sema;

    bool isEmpty() {
        bool returnVal;
        sema.lock();
        returnVal = nrElements == 0;
        sema.unlock();
        return returnVal;
    }
}
```

- **Synchronization concern**
- **Functionality concern**

- **Code tangling**
AOP Example

When a Buffer object receives the message isEmpty, first make sure the object is not being accessed by another thread through the get or put methods

AOP Example

When a Buffer object receives the message isEmpty, first make sure the object is not being accessed by another thread through the get or put methods

When to execute the aspect (joinpoints)
Composition of when and what (weaver directive)
What todo at the joinpoint (aspect functionality)
### AOP Example

```java
class Buffer {
    char[] data;
    int nrElements;

    bool isEmpty() {
        bool returnVal;
        returnVal = nrElements == 0;
        return returnVal;
    }

    before : reception(Buffer.isEmpty)
    { sema.lock();}

    after : reception(Buffer.isEmpty)
    { sema.unlock();}
}
```

### Talk Contents

- **Separation of Concerns**
- **Aspect-orientation**
- **Ready-to-use Technologies**
  - AspectJ
    - Example
    - Development-time Aspects
  - HyperJ
  - DemeterJ
  - Sina (Composition Filters)
- **Research activities**
- **AspectJ demo**
AspectJ

- Extension of Java with support for aspects
- Aspects can introduce extra behaviour around joinpoints.
  - Method calls
  - Assignments
  - Method Executions
  - Exception throws
  - …
- Integration in Emacs, Jbuilder, Forte4J, Eclipse

Example: figure editor

```
Display

Figure
  makePoint(..)
  makeLine(..)

FigureElement
  move(int, int)

Point
  getX()
  getY()
  setX(int)
  setY(int)
  move(int, int)

Line
  getP1()
  getP2()
  setP1(Point)
  setP2(Point)
  move(int, int)
```

factory methods

operations that move elements
**Joinpoints in AspectJ**

Imagine `l.move(2, 2)` — key points in dynamic call graph

- **a Line**
  - dispatch
  - a method execution returning or throwing

- **a Point**
  - dispatch
  - a method call returning or throwing

Joinpoint terminology

- **several kinds of join points**
  - method & constructor call
  - method & constructor execution
  - field get & set
  - exception handler execution
  - static & dynamic initialization
Pointcuts

each time there is a `<void Line.setP1(Point)>` or `<void Line.setP2(Point)>` method call

name and parameters

a “void Line.setP1(Point)” call

pointcut move():
  call(void Line.setP1(Point)) ||
  call(void Line.setP2(Point));

a “void Line.setP2(Point)” call

After advice

an aspect defines a special class that can crosscut other classes

aspect History Updating {
    pointcut move():
        call(void Line.setP1(Point)) ||
        call(void Line.setP2(Point));

    after() returning: move() {
        <code here runs after each move>
    }
}

after advice runs “on the way back out”
Development-time aspects

Debugging and tracing

• Turn debugging/tracing on/off without editing classes
• Debugging/tracing disabled with no runtime cost
• Can save debugging/tracing code between uses
• Easy to be sure it is off

Object / Macro vs Aspect

• using an object captures tracing support, but does not capture its consistent usage by other objects
• using an aspect captures the consistent usage of the tracing support by the objects
HyperJ

- **No base program**
  - Different views on one program
  - Compose different views
  - One concern per view

![Diagram of HyperJ's structure]

HyperJ vs AspectJ

- **Weaving**
  - Both at compile-time

- **Joinpoints**
  - HyperJ: static locations in source (static joinpoint model)
  - AspectJ: static and dynamic locations (dynamic joinpoint model)

- **Aspects?**
  - HyperJ: different views on OO program
  - AspectJ: OO program + aspects
DemeterJ: traversals

find all persons waiting at any bus stop on a bus route

![Diagram showing the relationship between classes: BusRoute, BusStopList, BusList, BusStop, PersonList, Person, and their association through busStops and passengers.]

OO solution: one method for each red class

DemeterJ: traversals

- **Describe traversal**
  - Language *primitives* to describe complex Visitor Design Patterns
  - In terms of graph of classes

- **Program robust towards changes in class hierarchy/relationships**

- **Special case of AspectJ aspects**
  - Maybe even domain-specific aspect-language for traversals…
**Composition Filters (Sina)**

- Extend Object with Filters (Modular Units)
- Filters intercept Messages
  - Conditions on messages
  - Extra behaviour on messages
- Also possible in AspectJ
  - Different Model

**Aspect Visualisation**

- **Aspect Browser**
  - Aspect classifier
- **FEAT**
  - Code ‘Surfer’ to classify crosscutting concerns
Talk Contents

- Separation of Concerns
- Aspect-orientation
- Ready-to-use Technologies
- Research activities
- AspectJ demo
Research in AO

- **Active Research Community**
  - Over 50 research groups participated in proposal for European network on AOSD
  - ±200 participants at first two AOSD Conferences
  - AOSD workshops at conferences
  - Many academic prototypes
  - Many open problems and questions

- **Research Topics**
  - Many research in alternative implementations
  - AO design
  - Composition and interference of Aspects
  - Dynamic aspects
  - …

Belgian Research

- **Separation of Business Rules from core application using AspectJ**
  - *Augustina Cibran, Maja D’Hondt (VUB)*
  - Models business rules using standard OO techniques
  - Coupling of business rules and base program with aspectJ
Belgian Research

- Composable Aspect-specific Languages
  - Johan Brichau (VUB)
  - Combine advantages of domain-specific and general-purpose aspect-languages.
  - Tackle issues of composition

- Intentional crosscut expressions
  - Kris Gybels (VUB)
  - Use a logic programming language (Prolog) to describe how an aspect crosscuts a base program

Belgian Research

- Runtime Aspect Composition for Distributed Systems
  - Eddy Truyen (KUL)
  - Composition of services required from a system. Each service is an aspect

- Security Aspects
  - Bart De Win (KUL)
  - Use aspects to modularize security

- Arriba Research Project
  - UIA / VUB / RUG
  - Use aspect technology for code instrumentation
Talk Contents

- Separation of Concerns
- Aspect-orientation
- Ready-to-use Technologies
- Research activities
- AspectJ demo

The End

Know more?

http://www.aosd.net

October 2001 issue of Communications of the ACM

Questions?

Johan Brichau
Johan.brichau@vub.ac.be
Programming Technology Lab
Vrije Universiteit Brussel