Virtual Separation of Concerns

Toward preprocessors 2.0

Christian Kästner
Software Product Lines & Features

• Set of related software systems (variants) in a domain
• Generated from common code base
• Variants distinguished in terms of features
Conditional Compilation with Preprocessors

- Annotated code fragments
- Variants with and without features
- Common for product-line implementation in industry

Preprocessors in C, C++, Java ME, Pascal, Erlang, C#, Visual Basic, ... GPP, GNU M4, SPLET, Frames/XVCL, Gears, pure::variants

Excerpt from Oracle's Berkeley DB
Objections / Criticism

Designed in the 70th and hardly evolved since

“#ifdef considered harmful”
“#ifdef hell”

“maintenance becomes a ‘hit or miss’ process”

“is difficult to determine if the code being viewed is actually compiled into the system”

“incomprehensible source texts”

“programming errors are easy to make and difficult to detect”

“CPP makes maintenance difficult”

“source code rapidly becomes a maze”

“preprocessor diagnostics are poor”
Academia: Compositional Approaches

Base / Platform

```java
class Stack {
    void push(Object o) {
        elementData[size++] = o;
    }
    ...
}
```

Feature: Queue

```java
refines class Stack {
    void push(Object o) {
        Lock l = lock(o);
        Super.push(o);
        l.unlock();
    }
    ...
}
```

Feature: Diagnostic

```java
aspect Diagnostics {
    ...
}
```

Composition

```java
class Stack {
    void push(Object o) {
        Lock l = lock(o);
        elementData[size++] = o;
        l.unlock();
    }
    ...
}
```

Module
Components
Frameworks, Plug-ins
Feature-Modules / Mixin Layers / ...
Aspects / Subjects, Hyper/J
Agenda

• Problems and Advantages of Preprocessors
• 4 Improvements
  • Views
  • Visual Representation
  • Disciplined Annotations
  • Product-Line-Aware Type System
• Summary and Perspective
Problem 1: Lack of Separation of Concerns

- Scattered and tangled source code
- Global search to understand a feature
- Deleting obsolete feature as challenge
- Difficult distributed development
- Difficult reuse
Problem 2: Obfuscation

- Mixture of two languages (C und #ifdef, ...)
- Control flow hard to follow
- Additional line breaks destroy code layout
- Long annotations difficult to recognize

```java
class Stack {
    void push(Object o
    #ifdef SYNC
    , Transaction txn
    #endif
    ) {
        if (o==null
        #ifdef SYNC
        || txn==null
        #endif
        ) return;
        #ifdef SYNC
        Lock l=txn.lock(o);
        #endif
        elementData[size++] = o;
        #ifdef SYNC
        l.unlock();
        #endif
        fireStackChanged();
    }
}
```
# Preprocessor in Femto OS

```c
void privTaskInit(TunitOS *pt) {
    uTaskInit(pt);  
    TaskControlBlock * task tcb = privGet_tcb(pt); 
    #if (defRqUseTaskInit == cfgTrue) 
        if (uiInitControl & defInitLockMux) 
            #if (cfgUseSynchronization == cfgTrue) 
                if (uiTaskNumber < defNumberOfTasks) 
                    privCleanSlotStack((TaskEx) tcb); 
                    #if (defUseMutexes == cfgTrue) 
                        privReleaseSyncBlocking1();
                        #endif
                    
                endif
            endif
        endif
    endif
}
```
Problem 3: Error-Prone

• Syntax Errors

```c
static int _rep_queue_filedone(...) {
    DB_ENV *dbenv;
    REP *rep;
    __rep_fileinfo_args *rfp; {
        #ifndef HAVE_QUEUE
            COMPQUIET(rep, NULL);
            COMPQUIET(rfp, NULL);
            return (__db_no_queue_am(dbenv));
        #else
            db_pgno_t first, last;
            u_int32_t flags;
            int empty, ret, t_ret;
            #ifndef DIAGNOSTIC
                DB_MSGBUF mb;
            #endif
            // over 100 lines of additional code
        }
        #endif
    }
}
```

• Type Errors

```c
#ifdef TABLES
class Table {
    void insert(Object data, Txn txn) {
        storage.set(data, txn.getLock());
    }
}
#endif

class Storage {
    #ifdef WRITE
        boolean set(...) { ... }
    #endif
}
```
Problem 3: Error-Prone

- Syntax Errors

```c
static int _rep_queue_filedone(...) {
    DB_ENV *dbenv;
    REP *rep;
    __rep_fileinfo_args *rfp; {

    #ifndef HAVE_QUEUE
        COMPQUIET(rep, NULL);
        COMPOQUIET(rfp, NULL);
        return (__db_no_queue_am(dbenv));
    #else
        db_pgno_t first, last;
        u_int32_t flags;
        int empty, ret, t_ret;
        #ifdef DIAGNOSTIC
            DB_MSGBUF mb;
        #endif
        // over 100 lines of additional code
    #endif
    }
    #endif
}
```

- Type Errors

```c
#endif TABLES
class Table {
    void insert(Object data, Txn txn) {
        storage.set(data, txn.getLock());
    }
}
#endif
class Storage {
    #ifdef WRITE
        boolean set(...) { ... }
    #endif
}
```
Problem 3: Error-Prone

• Syntax Errors

```c
static int _rep_queue_filedone(...) {
    DB_ENV *dbenv;
    REP *rep;
    __rep_fileinfo_args *rfp; {
#ifndef HAVE_QUEUE
    COMPQUIET(rep, NULL);
    COMPQUIET(rfp, NULL);
    return (__db_no_queue_am(dbenv));
#else
    db_pgno_t first, last;
    u_int32_t flags;
    int empty, ret, t_ret;
    #ifdef DIAGNOSTIC
    DB_MSGBUF mb;
    #endif
    // over 100 lines of additional code
#endif}
#endif
```

• Type Errors

```c
#define TABLES
class Table {
    void insert(Object data, Txn txn) {
        storage.set(data, txn.getLock());
    }
}
#endif
class Storage {
    #ifdef WRITE
    boolean set(...) { ... }
    #endif
}```
Advantages

• Easy to use
  • “annotate and remove”
  • Already part of many languages / simple tools
  • Most developers already familiar

• Flexible / Expressive
• Language-independent / Uniform

• Low Adoption Barrier
  • esp. with legacy code
Agenda

• Problems and Advantages of Preprocessors
• 4 Improvements
  • Views
  • Visual Representation
  • Disciplined Annotations
  • Product-Line-Aware Type System
• Summary and Perspective
Views

- Emulate modularity
- Hide irrelevant source code
  - Hides entire files
  - Hides code inside files

Related work:
- C-CLR
- Version Editor
- FeatureMapper
- pure::variants
- Effective Views
- on-demand remodularization

[ICSE’08, ViSPLE’08]
Expression Problem

```java
#ifdef ADD
class Add extends Expr {
    Expr left, right;
    Add(Expr l, Expr r)
    { left=l; right=r; }
    #ifdef EVAL
    double eval() {
        return left.eval() +
        right.eval();
    }
    #endif
    #ifdef PRINT
    void print() {
        left.print();
        System.out.print("+");
        right.print();
    }
    #endif
    #endif
}
#endif

#define POWER
class Pow extends Expr {
    Expr base, exp;
    Pow(Expr b, Expr e)
    { base=b; exp=e; }
    #ifdef EVAL
    int eval() {
        return MathLib.pow(
        base.eval(),
        right.eval());
    }
    #endif
    #ifdef PRINT
    void print() {
        left.print();
        System.out.print("^");
        right.print();
    }
    #endif
    #endif
}
#endif

class MathLib {
    static int pow
    (int b, int e)
    {
        if (e<=0)
            return 1;
        return b *
        pow(b, e-1);
    }
    //...
}
```
View on Feature Eval

- Shows all files containing Eval code
- Shows context information (kursiv)
- Hides code without annotations
View on the Variant “Add and Eval”

```c
#include ADD

class Add extends Expr {
  Expr left, right;
  Add(Expr l, Expr r)
      {left=l; right=r;}
  #ifdef EVAL
  int eval() {
    return left.eval() +
            right.eval();
  }
  #endif
  [...]}
  #endif
```

```c
class MathLib {
  static int pow
      (int b, int e)
      {
        if (e<=0)
          return 1;
        return b *
              pow(b, e-1);
      }
  //...
```

- Entirely hides file *Power*
- Hides method *print* in *Add* ([…])
- Code without annotation remains visible
Agenda

• Problems and Advantages of Preprocessors
• 4 Improvements
  • Views
  • Visual Representation
  • Disciplined Annotations
  • Product-Line-Aware Type System
• Summary and Perspective
**Visual Representation: Background Colors**

```java
class Stack {
    void push(Object o, Transaction txn) {
        if (o == null || txn == null)
            return;
        Lock l = txn.lock(o);
        elementData[size++] = o;
        l.unlock();
        fireStackChanged();
    }
}
```

**Features:**
- **Synchronization**

**Related Work:**
- Spotlight
- NetBeans
- fmp2rsm
- FeatureMapper
- AspectBrowser
"
Alternative Visual Representations

- NetBeans

```java
/** Read HTML and if it has links, redirect and parse */
protected String parseHTMLRedirect(String url, InputStream is) throws IOException, Exception {
    //ifdef D SHALL

    //##
    throw new IOException("Error HTML not supported");
    //else
    if (m_redirect) {
        //ifdef D Logging
          logger.severe("Error 2nd redirect wlan");
        //endif
    System.out.println("Error 2nd redirect wlan");
    //endif
    throw new IOException("Error url ",
    " to 2nd redirect wlan");
    }

    m_redirect = true;
    m_redirectUrl = url;
    com.shouldcode.rssreader.businessentities.
    HTMLLinkParser.parseSchemas(is);

    //##
    //##
    //##
}
```

- Spotlight
Scaling Visual Representations

• Focus on few features at a time
• Repeating colors / manual assignment sufficient

• Analysis of 4 Java ME and 40 C programs:
  • 96 % pages of source code with \( \leq 3 \) colors
  • 99 % pages of source code with \( \leq 7 \) colors

<table>
<thead>
<tr>
<th>Name</th>
<th>Pages</th>
<th>Annotations per page (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Product Line</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>Functional Graph Library</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>Mobile Media (Ver. 8)</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>Berkeley DB</td>
<td>2921</td>
<td></td>
</tr>
</tbody>
</table>
Program Comprehension: An Experiment

- `#ifdef` vs. colors
- 43 subjects in 2 groups
- S1-2: search tasks faster with colors (43% & 23%)
- M1-3: maintenance tasks same perform.
- M4: maintenance task with red background color -37%
- No influence on correctness
- Subjects prefer colors
Agenda

- Problems and Advantages of Preprocessors
- 4 Improvements
  - Views
  - Visual Representation
  - Disciplined Annotations
  - Product-Line-Aware Type System
- Summary and Perspective
Disciplined Annotations

- Syntax errors caused by annotations on plain text
- Solution: Use underlying artifact structure
  - Enforce disciplined annotations on entire classes, methods, or statements
  - Annotations of isolated brackets or keywords regarded as undisciplined
Variant Generation by AST Transformation

```
class C {
    void m(int p) {
        s1();
        s2(p, true);
    }
}
```

```
class C {
    void m(int p) {
        s1();
    }
}
```

Variant generation cannot cause syntax errors.
Expressiveness of Disciplined Annotations

- Not all annotations are possible
- Sometimes workarounds required
- Experience: not a significant restriction in practice
- ~90% of all annotations in 40 C programs is disciplined already

![Diagram showing high flexibility to no flexibility on a spectrum, with unrestricted annotations on any character and disciplined annotations (underlying structure) as the middle ground. Classes are only annotations at the no flexibility end.]
Agenda

- Problems and Advantages of Preprocessors
- 4 Improvements
  - Views
  - Visual Representation
  - Disciplined Annotations
  - Product-Line-Aware Type System
- Summary and Perspective
Product-Line-Aware Type System

- **Base:** Type correct without annotations (for tool support)
- **Detecting all pairs:**
  - Method invocation and method declaration
  - Reference and declaration of class, variable, etc.
  - ...
- **Comparison of annotations for each pair**

Declaration annotated + Reference not annotated \[\rightarrow\] Error in some variants

Related Work:
- SafeGen
- fmp2rsm
- Safe Composition
- FFJPL
- Conditional Methods
Type Checking in the Context of a Feature Model

- Handling dependencies between features
- Can all variants with reference reach a corresp. declaration?

- Implementation: Propositional logic and SAT solvers:

In all variants in which code fragment A is included also code fragment B is included:

\[ FM \Rightarrow (AT(a) \Rightarrow AT(b)) \]
Formalization: CFJ

- On top of Featherweight Java
- Theorem: Generation preserves typing
- Formal proof with Coq

\[
\begin{align*}
\text{Term typing} & \\
\Gamma \vdash t : C & \text{with } \forall \alpha \in \Gamma \quad \alpha \rightarrow \alpha' \\
\text{(T-VAR)} & \\
\text{fields}(C_0) = \overline{C} & \text{on top of Featherweight Java} \\
\text{(T-FIELD)} & \\
\text{fields}(C) = \overline{D} & \text{AT}(t); \Gamma \vdash \overline{C} : C < : D \\
\text{(T-INVK)} & \\
\text{fields}(C) = \overline{D} & \text{AT}(t); \Gamma \vdash \overline{C} : C < : D \\
\text{(T-NEW)} & \\
\text{Method typing} & \\
M & \text{OK in } C \\
\text{(M-OK)} & \\
\text{Class typing} & \\
\text{OK} & \\
\text{(C-OK)} & \\
\text{supid } & \text{class } C \text{ extends } D \{ \overset{\alpha}{C} ; K \overset{\overline{M}}{M}; \text{OK} \}
\end{align*}
\]
Implementation: CIDE

http://fosd.de/cide
Automated Refactorings

- Feature Module

```java
class Stack {
    int[] data;
    void push(int o) { /*...*/ }
    int pop() { /*...*/ }
}
```

- Annotationen

```java
class Stack {
    int[] data;
    #ifdef UNDO
    int backup;
    void undo() { /*...*/ }
    #endif
    #ifdef TOP
    int top() { /*...*/ }
    #endif
    void push(int o) {
        #ifdef UNDO
        backup = top();
        #endif
        original(v);
    }
    int pop() { /*...*/ }
}
```

Automated Transformation
Agenda

• Problems and Advantages of Preprocessors
• 4 Improvements
  • Views
  • Visual Representation
  • Disciplined Annotations
  • Product-Line-Aware Type System
• Summary and Perspective
Summary

• Problems:
  • Separation of Concerns
  • Obfuscation
  • Error-proneness

• Retained advantages:
  • Easy to use
  • Flexible
  • Language-independent
  • Low adoption barrier

• Improvements:
  • Views
  • Visual representation
  • Disciplined annotations
  • Product-line-aware type system

“Virtual Separation of Concerns”
Summary

- **Problems:**
  - Separation of Concerns
  - Obfuscation
  - Error-proneness

- **Retained advantages:**
  - Easy to use
  - Flexible
  - Language-independent
  - Low adoption barrier

- **Improvements:**
  - Views
  - Visual representation
  - Disciplined annotations
  - Product-line-aware type system

“Virtual Separation of Concerns”
Summary

• Problems:
  • Separation of Concerns
  • Obfuscation
  • Error-proneness

• Retained advantages:
  • Easy to use
  • Flexible
  • Language-independent
  • Low adoption barrier

• Improvements:
  • Views
  • Visual representation
  • Disciplined annotations
  • Product-line-aware type system

• Remaining Problems:
  • Separate Compilation
  • Black-box Reuse

“Virtual Separation of Concerns”
Perspective

• Preprocessors common in practice, despite strong criticism
• Neglected by researchers
• Tool support can address most problems
• Interim solution (with migration path) or serious alternative?
• Give preprocessors a second chance!
Questions?

http://fosd.de/cide