Modular Reasoning for Software Product Lines with Emergent Feature Interfaces

Jean Melo
Advisor: Paulo Borba
Software Product Line
Preprocessors
(a.k.a. conditional compilation)

• Often used to implement the variability (features) of SPLs

```c
... char_u *fname_res = ...;
...
#ifdef UNIX || WIN3264
...
# endif
...
tail = make_percent_swname(fname_res);
...
#endif
...```
Preprocessors

• Often used to implement the variability (features) of SPLs
• But,

```c
... char u *fname_res = ...;
 ...
 ifndef UNIX || WIN3264 ...
 ...
 tail = make_perfect_swname(fname_res);
 ...
 #endif ...
```
Problem

• Code maintenance can **break** a variant of a SPL. For example, changes in one feature might **affect** another one.

Example #1: Bug in Vim editor

```cpp
char_u *fname_res = *fname;
...
#endif
...
```

```cpp
char_u *fname_res = null;
...
#endif
...
```

Example #1: Bug in Vim editor
Problem

• Code maintenance can **break** a variant of a SPL. For example, changes in one feature might **affect** another one.

```c
... char_u *fname_res = *fname;
...
#elifdef UNIX || WIN3264
tail = make_percent_swname(fname_res);
... #endif
...
```

```c
... char_u *fname_res = null;
...
#elifdef UNIX || WIN3264
tail = make_percent_swname(fname_res);
... #endif
...
```

Example #1: Bug in Vim editor

Compilation error
public void computeLevel() {
    ...
    totalScore = perfectCurvesCounter * PERFECT_CURVE_BONUS
    + perfectStraightCounter * PERFECT_STRAIGHT_BONUS
    + gc_levelManager.getCurrentCountryId()
    - totalLapTime * SRC_TIME_MULTIPLIER;
    ...

    NetworkFacade.setScore(totalScore);
    NetworkFacade.setLevel(this.gcGetCurrentLevel());

    public void setScore(int s) {
        score = (s < 0) ? 0 : s;
    }
}

Example #2: Maintenance in the best lap product line
Example #2: Maintenance in the *best lap* product line

```java
public void computeLevel() {
    ...
    totalScore = perfectCurvesCounter * PERFECT_CURVE_BONUS
                + perfectStraightCounter * PERFECT_STRAIGHT_BONUS
                + gc_levelManager.getCurrentCountryId();
    - totalLapTime * SRC_TIME_MULTIPLIER;
    ...
}
```

**Behavioral error**

NetworkFacade.setScore(totalScore);
NetworkFacade.setLevel(this.gc_getCurrentLevel());

```java
public void setScore(int s) {
    score = (s < 0) ? 0 : s;
}
```

Negative score appears correctly in the game! Let’s commit!

Zero instead of -90!
To minimize the feature modularization problem...
Emergent Interfaces

- The developer selects the **maintenance point**

```
char_u *fname_res = ...;
...
#ifdef UNIX || WIN3264
...
tail = make_percent_swname(fname_res);
...
#endif
...
```

Be careful! You provide *fname_res for the UNIX and WIN3264 features.

Code analysis is performed...
Let’s maintain the feature A!

```java
public void method_X() {
    ...
    B
    A
    A
    ...
    C
    A
    A
    B
    B
    ...
    A
    C
    ...
    B
    A
    ...
}
```
Let’s maintain the feature A!

What we need to do using EL?
Let’s maintain the feature A!

What we need to do using EI?

Select all fragments of the feature A one-by-one! 😞
Emergent Interfaces...

... capture dependencies between a feature maintenance point and parts of other feature implementation, but they do not provide an overall feature interface considering all parts in an integrated way.
public static void main(String args[]) {
    String title;

    //#ifdef PT_BR
    title = "JCalc - Calculadora Padrão e Científica";
    ...
    //#endif

    //#ifdef GUI
    JFrame frame = new JFrame(title);
    ...
    //#endif

    //#ifdef LOOKANDFEEL
    initLookAndFeel(frame);
    //#endif

    //#ifdef GUI
    ...
    JCalcStandardFrame myFrame = new JCalcStandardFrame();
    JPanel myPane = myFrame.getPane();

    frame.getContentPane().add(myPane, ...);
    ...
    //#endif
}
public static void main(String args[]) {
    String title;

   .isDefined PT_BR
    title = “JCalc - Calculadora Padrão e Científica”;
    ...
    #endif

   .isDefined GUI
    JFrame frame = new JFrame(title);
    ...
    #endif

   .isDefined LOOKANDFEEL
    initLookAndFeel(frame);
    #endif

   .isDefined GUI
    ...
    JCalcStandardFrame myFrame = new JCalcStandardFrame();
    JPanel myPane = myFrame.getPane();

    frame.getContentPane().add(myPane, ...);
    ...
    #endif
}
public static void main(String args[]) {
    String title;

    //=ifdef PT_BR
    title = “JCalc - Calculadora Padrão e Científica”;

    //=endif

    //=ifdef GUI
    JFrame frame = new JFrame(title);

    //=endif

    //=ifdef LOOKANDFEEL
    initLookAndFeel(frame);

    //=endif

    //=ifdef GUI

    JCalcStandardFrame myFrame = new JCalcStandardFrame();
    JPanel myPane = myFrame.getPane();
    frame.getContentPane().add(myPane, ...);

    //=endif
}
Another problem is...
Dependencies among heterogeneous artifacts!

```groovy
//if($ImportBibtex)
def generateBib()
    def publication = Publication.get(params.id)
    ...
}
//end
PublicationController.groovy
```

```
<!-- #if($Bibtex) -->
...
<g:link controller="Publication" action="generateBib">Bibtex</g:link>
<!-- #end -->
show.gsp
```
Emergent Interfaces...

...do not capture dependencies from different kinds of artifacts (.java, .groovy, .jsp, .gsp, etc.)
Dependencies between GSP-Groovy

```groovy
class User {
    String username
    ...
}
```

```
<g:form id="log" controller="auth" action="signIn">
    <!-- ... -->
    <g:textField name="username" required="true"/>
    <!-- ... -->
</g:form>
```

login.gsp
Dependencies between GSP-Groovy

```groovy
User.groovy

class User {
    String username
    ...
}
```

```groovy
User.groovy
```

```groovy
User.groovy
```
To address these problems...
Emergent Feature Interfaces!
EFI for a single language settings

```java
public static void main(String args[]) {
    //
    // ifdef GUI
    JFrame frame = new JFrame(title);
    initResolution(frame);
    //#endif

    // ifdef LOOKANDFEEL
    initLookAndFeel(frame);
    //#endif
    ...
}
```

**Requires** `title` **from** PT_BR

**Provides** `frame` **to** LOOKANDFEEL
EI vs. EFI

```java
public static void main(String args[]) {
    //
    //#ifdef GUI
    JFrame frame = new JFrame(title);
    initResolution(frame);
    //#endif

    // //#ifdef LOOKANDFEEL
    initLookAndFeel(frame);
    // #endif

    ...
}
```

**Requires** 
*title from PT_BR*

**Provides** 
*frame to LOOKANDFEEL*

**Provides** 
*frame to LOOKANDFEEL*

No dependencies found!
Emergo

A tool for improving the maintainability of SPLs
Single language evaluation
(EIF versus EFI)
Evaluation main goals

- Is there any difference between EI and EFI in terms of size and precision?
- How do EFI dependency detection compare to EI?
Study settings: experimental objects

<table>
<thead>
<tr>
<th>System</th>
<th>Version</th>
<th># methods</th>
<th>MDi</th>
<th>MDe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best lap</td>
<td>1.0</td>
<td>343</td>
<td>20.7%</td>
<td>11.95%</td>
</tr>
<tr>
<td>Juggling</td>
<td>1.0</td>
<td>413</td>
<td>16.71%</td>
<td>11.14%</td>
</tr>
<tr>
<td>Lampiro</td>
<td>10.4.1</td>
<td>1538</td>
<td>2.6%</td>
<td>0.33%</td>
</tr>
<tr>
<td>MobileMedia</td>
<td>0.9</td>
<td>276</td>
<td>7.97%</td>
<td>5.8%</td>
</tr>
<tr>
<td>Mobile-rss</td>
<td>1.11.1</td>
<td>902</td>
<td>27.05%</td>
<td>23.84%</td>
</tr>
</tbody>
</table>

# methods: Number of Methods; MDi: Methods with Directives; MDe: Methods with Dependencies
Study settings: method and maintenance point selection

• Random selection of methods
  – Only methods with dependencies
  – No dependency: empty interfaces
  – We use MDe to select ten methods

• Random selection of maintenance points
  – Only suitable maintenance points
  – No comment or whitespace as well as method/class declaration
  – Feature-sensitive dataflow analysis is intra-procedural
Evaluation results

• EI return ‘No dependencies found!’ in all cases that the maintenance point is not an assignment (i.e. do not have required interfaces)

• Methods with many dependencies favor EFI

• EI do not provide support for feature selection as a maintenance point
Threats to validity

• Only 10 methods were chosen (five SPLs)
• Unavailable feature models
• Manually computing EI and EFI
• More studies are required to draw more general conclusions
Regarding the cross-language dependencies...
Cross-Language Analysis!

```
<!- #if($Bibtex) -->
...
<g:link controller="Publication" action="generateBib"/>
...
<!- #end -->
```

Show.gsp

Requires `generateBib` from PublicationController
Cross-Language Analysis!

show.gsp

```groovy
// #if($ImportBibtex)
def generateBib() {
  def publication = Publication.get(params.id)
  render(text: publication.generateBib(),
         contentType: "text/txt",
         encoding: "UTF-8")
}
// #end
```

PublicationController.groovy

```groovy
<-- #if($Bibtex) -->
...
<g:link controller="Publication" action="generateBib"/>
...
<-- #end -->
```

Requires `generateBib` from PublicationController
GSPAnalyzer
A tool for capturing cross-language feature dependencies in Grails SPLs
Multi-language evaluation
(A case study with RGMS)
Evaluation main goals

- Measure the number of occurrences of cross-language dependencies in the RGMS
- Identify different types of dependencies among heterogeneous artifacts
RGMS SPL

Research Group Management System
Developed during an SPL course
More than 50 students implemented it
Using the Grails framework

<table>
<thead>
<tr>
<th>RGMS Stats</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># artifacts</td>
<td>371</td>
</tr>
<tr>
<td># languages involved</td>
<td>7</td>
</tr>
<tr>
<td># features</td>
<td>17</td>
</tr>
<tr>
<td># LOC</td>
<td>40026</td>
</tr>
</tbody>
</table>
Satisfied dependency

```g:form id="log" controller="auth" action="signIn">
  <!-- ... -->
  <g:textField name="username" required="true"/>
  <!-- ... -->
</g:form>

class User {
  String username

  ...
}

User.groovy

login.gsp
Unsatisfied dependency

Periodico/show.gsp

```groovy
class PeriodicoController {
    //...
    //=if($Facebook)
    def share() { ... }
    //=end
    //=...
}
```

PeriodicoController.groovy
Evaluation results

Evaluation summary

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># pages analyzed</td>
<td>76</td>
</tr>
<tr>
<td># satisfied dependencies</td>
<td>304</td>
</tr>
<tr>
<td># unsatisfied dependencies</td>
<td>4</td>
</tr>
</tbody>
</table>

Set of pages
Threats to validity

• RGMS is an academic software product line
• Our tool has not recognize the Groovy constructs `hasMany` and `belongsTo` yet
• We need of other case studies with different sizes, purposes, architectures, granularity, and complexity
Concluding remarks

• Our work helps to achieve independent feature comprehensibility and, consequently, to maintain features in SPLs
  – Looking for feature dependencies
  – Providing a global interface considering all fragments in an integrated way, and,
  – Supporting different types of artifacts
Limitations

- We capture only data dependencies
- So far, we only make syntactic analysis of Grails programs and
- Capture dependencies among...
Future work

• Provide inter-procedural analysis to capture feature dependencies among methods, classes, and components

• More studies are required to draw more general conclusions
  – Benefits of supporting different types of artifacts
Modular Reasoning for Software Product Lines with Emergent Feature Interfaces

Jean Melo (jccm@cin.ufpe.br)
Advisor: Paulo Borba
BACK-UP SLIDES
Emergo’s Architecture

GUI → Core → Analysis

Resolve Classpath → getCompilationUnit

GenerateEL Handler

CompUnit Java → CompUnit Groovy

ASTNodeUnitBridge

Configure Classpath → Load Class

Soot

Instrumentor

LiftedReaching Definitions

DependencyGraph Builder

ASTNodeUnit BridgeJava

ASTNodeUnit BridgeGroovy

EmergoGraphView

int a = 0;
A
m(a);

Maintenance Point

-> Mark the AST from Text Selection

-> pruning the graph
Our proposal in a nutshell

(1) Preprocessing;
(2) Compiling;
(3) Instrumenting;
(4) Dataflow analysis;
(5) Transformation of the analysis’ results.

Diagram:
- Groovy annotated → Groovy
- CK → Groovy
- Groovy → Jimple
- Jimple annotated → Jimple
- Jimple → Results
- Results → Dependency Graph

Dependency Graph:
1. Groovy annotated to CK
2. Groovy to Jimple
3. Jimple annotated to CK
4. Jimple to Results
5. Results to Dependency Graph
static void main(String[] args) {
  // ifdef A
  def a = 0
  // endif

  ...

  // ifdef B
  m(a)
  // endif

}

Sample code
Preprocessing

1. #ifdef A
2. def a = 0
3. #endif
... 
20. #ifdef B
21. m(a)
22. #endif

Groovy annotated

Groovy

CK

Jimple annotated

Results

Dependency Graph

<metadata>
<config expression="A" line=[2]/>
<config expression="B" line=[21]/>
</metadata>
Compiling

groovy_annotated

CK

Groovy annotated

Groovy

Jimple

virtualinvoke\langle\texttt{void m(int)}\rangle(b0);
return;

Jimple annotated

Results

Dependency Graph

public static void main(java.lang.String[]) {
    r0 := @parameter0: java.lang.String[];
b0 = 0;
...
"}
public static void main(java.lang.String[]) {
    r0 := @parameter0: java.lang.String[];
    b0 = 0;
    ... 
    virtualinvoke<void m(int)>(b0);  // A
    return;
}
Dataflow analysis

Legend:
0 -> No feature
1 -> A
2 -> B
3 -> A and B

r0 := @parameter0: java.lang.String[] => {3={}, 2={}, 1={}, 0={}}
b0 = 0 => {3={}, 2={}, 1={}, 0={}}
... virtualinvoke r1.<void m(int)>(b0) => {3={b0 = 0}, 2={}, 1={}, 0={}}
...
Transformation of the analysis’ results

```groovy
def a = 0
m(a)
```

A and B
Emergent Feature Interface

```java
static void main(String[] args) {
    // ifdef A
    def a = 0
    // endif

    ...
    // ifdef B
    m(a)
    // endif
}
```

**Provides** $a = 0$ to B

[Configuration: A and B]
Dependency between GSP-Groovy

...<form id="log" action="auth/signIn" method="post">
  <td>Username: </td>
  <input name="username" required="true"/>
</form>...

```
groovy

class Member {
  String username
  ...
}
```

"Dependency between GSP-Groovy"

...
Dependency between GSP-Groovy

Unmatched *login* element (params) between page and code.
Dependency among GSP-Groovy-jrxml

```
... environments {
  development {
    grails.logging.jul.usebridge = true
    jasper.dir.reports = '../rgms/web-app/reports/report_Bundle'
  }
  ...
}
```

```
...<g:jasperReport jasper="publication"
  format="PDF,HTML,XML" name="export" />
...```

`publication` file not found at configured report folder
def user = Member.findByUsername(params.username)
...
log.info "Redirecting to '${targetUri}'." 
redirect(uri: "/member/list")
...

.groovy
def user = Member.findByUsername(params.username)
...
log.info "Redirecting to '${targetUri}'."
redirect(uri: "/member/list")
...

newPassword action not found on the current controller.
Dependency between GSP-Groovy

```gsp
...<form id="log" action="auth/signIn" method="post">
...
</form>
```

```groovy
...def signIn = {
   // Authentication code
}
...```

Dependency between GSP-Groovy

```groovy
... def signIn = {
// Authentication code
}
...
```

```gsp
...<form id="log" action="auth/signIn" method="post">...
</form>
```

```
groovy
<form id="log" action="auth/login" method="post">...
</form>
```

*login* action not found on the auth controller.
We have dependencies among...

Legend:
- M: Model
- V: View
- C: Controller