JavaUI: Effects for Controlling UI Object Access

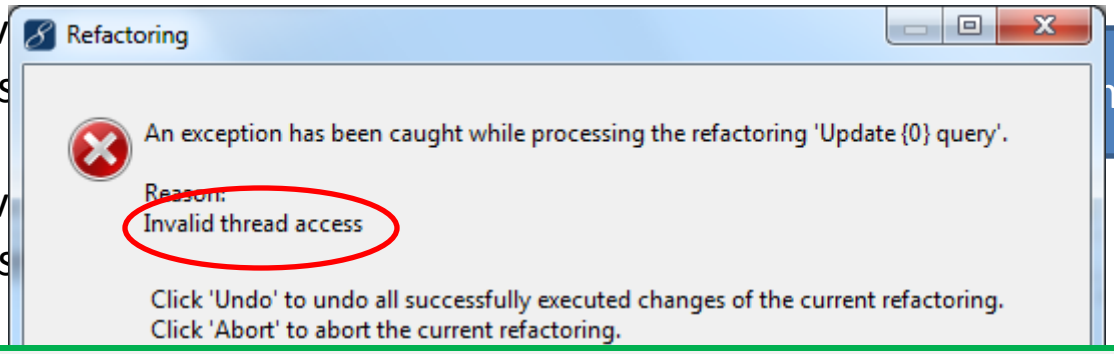
Colin S. Gordon, Werner M. Dietl,
Michael D. Ernst, Dan Grossman

University of Washington

https://github.com/csgordon/javaui

Invalid UI Updates

```
…
public static v
      mylabel.s
}
public static v
      mylabel.s
}
```



n method1()

```
// Corrected method2
public static void method2() {
      Display.syncExec(new Runnable {
            void run() { mylabel.setText("Hello from method2()"); }
      });
}
```

http://www.myeclipseide.com/PNphpBB2-printview-t-26285-start-0.html

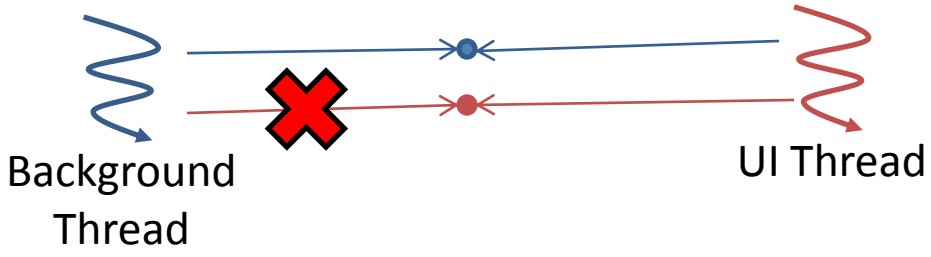
GUI Threading Errors are Common*

- In Top 3 SWT Exceptions
- Tens of thousands of search results
- >2700 Eclipse bug reports
- Possibly long-lived
  - One >10 years old in Eclipse!

*Zhang et al. (ISSTA'12)

GUI Thread Discipline

- GUI frameworks use a distinguished *UI thread*
- Only UI thread may access UI objects directly
  - Most GUI methods must run only on UI thread
  - E.g. `JLabel.setText()`
- Calling methods on wrong threads results in
  - Bugs
  - Crashes
  - Assertion failures



Appeal of the UI Thread Discipline

Use syncExec(new Runnable{ run(){ … } }):

- Simple specification
- Allows for responsive UIs
- Forced atomicity specs
- Simple (no) synchronization on UI objects
- Simple dynamic enforcement (assertions)

Need to track calling contexts well

Calling Context Confusion

```
public void updateView() {
    setupTitle(this.title);
}
```

- Can I directly touch the UI from this method?
- When is it safe to call this method?
  - UI libraries* don't document this behavior well
    - *Mainly SWT & JFace
- These calling context questions are easy to get wrong.
  - Results in bugs

JavaUI's Contributions

- Static effect system to prevent bad UI access
  - Sound
  - Simple approach to effect polymorphism
  - Annotations document threading assumptions
- Implementation
  - Uses Checker Framework
  - https://github.com/csgordon/javaui
- Evaluation
  - Effective on >140,000 LOC
- Lessons for other effect systems

Outline

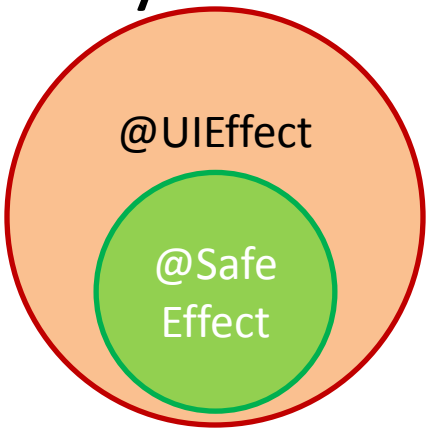
- Preventing GUI Threading Errors
- Evaluating JavaUI
- Lessons & Conclusions

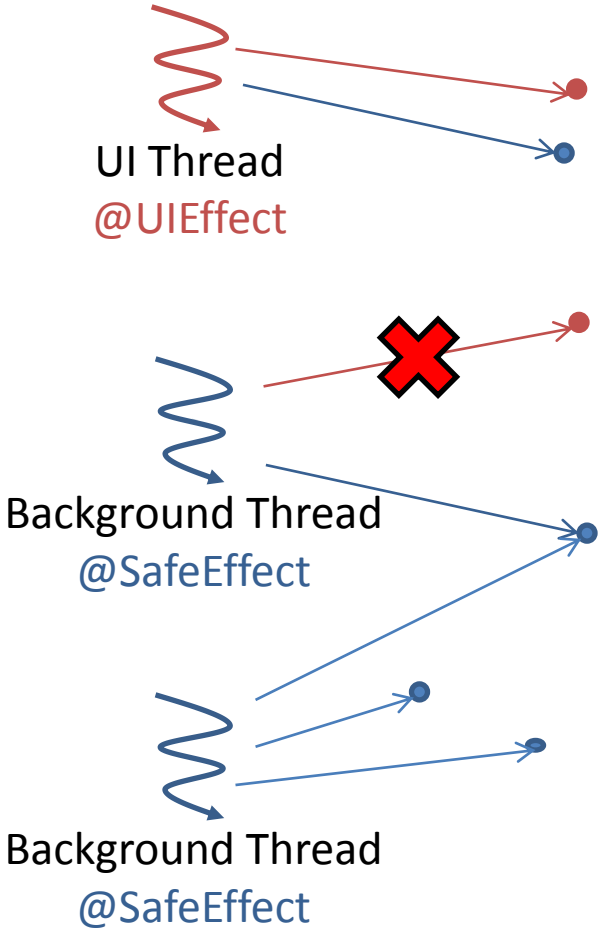
Alternatives: Detecting Bad UI Access

- Standard concurrency analyses don't apply
- Runtime checks & runtime exceptions
  - Cause of current crashes (still useful!)
- Make UI access safe from any thread
  - Locking
    - Deadlock-prone
    - Slower?
  - Retry operations on UI thread
    - Weird UI update interleavings

Effects for UI Methods

- Effects: constrain process of computation
- @UIEffect: code must run on UI thread
- @SafeEffect: code may run on any thread





UI Effects Detect Errors

```
class JLabel { …
  @UIEffect public void setText(String s);
… }
…
class Client { …
  @SafeEffect public void updateView() {
    myJlabel.setText("New Title"); // ERROR
  }
  @UIEffect public void refreshView() {
    myJlabel.setText("New Title"); // OK
  }
}
```

Polymorphic Effects

```
interface Runnable {
    @??Effect public void run();
}
```

- @UIEffect?
  - Breaks background thread uses…
- @SafeEffect?
  - Breaks sending code to UI thread…
- Must preserve API compatibility!
  - Can’t split the interface

An Effect-Polymorphic Type

- Intuitively:

```
interface Runnable<E implements Effect> {
    @Effect(E) public void run();
}
```

- Actual syntax:

```
@PolyUIType interface Runnable {
    @PolyUIEffect public void run();
}
```

Instantiating Effect-Polymorphism

- Instantiate implicit effect parameter via *type qualifiers*:

```
@Safe Runnable s = ...;
s.run(); // has the safe effect (background!)
@UI Runnable u = ...;
u.run(); // has the UI effect (UI code!)
```

Safely Running Code on the UI Thread

- Background threads must send code to the UI thread:

```
class org.eclipse.swt.widgets.Display {
  @SafeEffect
  static void syncExec(@UI Runnable runnable);
  @SafeEffect
  static void asyncExec(@UI Runnable runnable);
}
```

Safe Calls to UI Methods

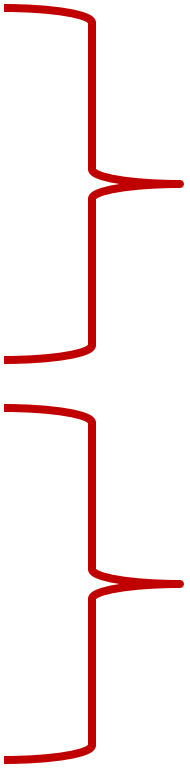
```
@SafeEffect public void updateView() {

   myJlabel.setText("New Title"); // ERROR

   Display.syncExec(new @UI Runnable {
     @UIEffect void run() {
       myJlabel.setText("New Title"); // OK
     });
 }
```

Outline

- Preventing GUI Threading Errors
- Evaluating JavaUI
- Lessons & Conclusions

Evaluating JavaUI

- Annotated 8 programs, >140,000 LOC:
  - EclipseRunner
  - HudsonEclipse
  - S3dropbox
  - SudokuSolver
  - Eclipse Color Theme
  - LogViewer
  - JVMMonitor
  - Subclipse

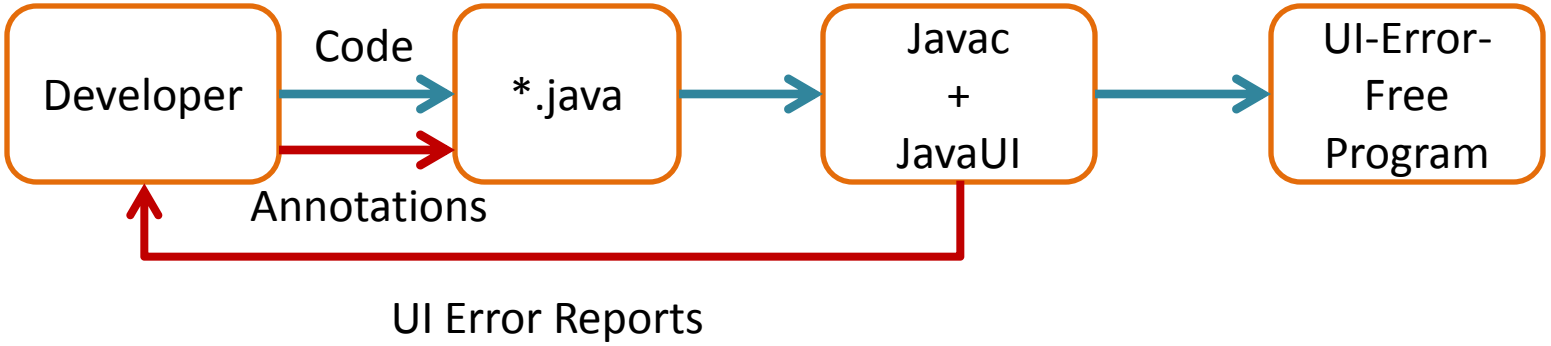


Compare to prior work

Large, popular Eclipse plugins

Applying JavaUI to Legacy Code

- Run javac with JavaUI annotation processor
- Examine error reports
- Add annotations to clarify program
- Repeat until you can't fix more errors



One-time process! Lower incremental cost.

Results

Program	Warnings	UI Defects	Other Defects	False Positives	Unknown	Annos / KLOC
EclipseRunner	1	1	0	0	0	10
HudsonEclipse	13	3*	0	2	0	4
S3dropbox	2	2	0	0	0	21
SudokuSolver	2	2	0	0	0	8
Eclipse Color Theme	0	0	0	0	0	3
LogViewer	1	0	0	1	0	17
JVMMonitor	9	0	0	9	0	7
Subclipse	19	0	1**	13	5	8

* 3-4 Warnings / bug, compound expressions
** Dead code with wrong effect

JavaUI Effectiveness

- Modest annotation burden
  - 7 annotations / 1000 LOC
  - Careful choice of defaults, scoped default controls
- Usually not hard to annotate a project
  - 4300 LOC/hour for last four projects
- Few false positives
  - Usually code smells
  - Would be avoided if starting with JavaUI

False Positives in 140,000 LOC

- 12 code smells: global property store issues
  - Dictionary w/ callbacks of both effects
- 5 legacy issues: pre-generics code + subtyping
  - Implicitly, List<T extends @Safe Object>
  - Can't add @UI IChangeListener (not <: @Safe Object)
- 5 ambiguous design (unknown)
  - No documentation
- 1 subtyping abuse
- 7 require data-dependent effects

Subtyping Abuse

- Many methods are intended to have a specific effect
  - Modest docs, no checks: devs bend the rules
- E.g., @SafeEffect override of @UIEffect method, used safely but stored as supertype:

```
class Super { @UIEffect public void m() {…} }
class Sub extends Super {
    @Override @SafeEffect public void m() {…}
}
…
Super s = new Sub(); // field decl
…
s.m(); // ERROR: UI effect, but safe context
```

  - Creates errors; introduce explicitly safe subclass
- Or override @SafeEffect method with @UIEffect, but control data flow of bad subclass (no easy fix)

Effects Depending on Runtime Values

- Dynamic thread checks
  `if (currThread.isUIThread()) …`
  - Different branches have different effects
  - Sometimes boolean dataflow is also present
- Argument-qualifier-dependent effects

```
@EffectFromQual("runnable")
public static void
showWhile(Display display,
          @PolyUI Runnable runnable);
```

  - Effect of `showWhile()` depends on the effect of the Runnable
  - Also on whether `display` is null…

Outline

- Preventing GUI Threading Errors
- Evaluating JavaUI
- Lessons & Conclusions

Lessons for Other Effect Systems

- Effect system design:
  - Scoping defaults to lower annotation burden
  - One effect variable may be enough
- Poor framework design:
  - Should separate effects, but don't
- Effect system features:
  - Implicit effect polymorphism everywhere
  - If effects are associated with runtime values, programs will have data-dependent effects

Related Work

- CheckThread
  - Open source tool
  - Effect system?
  - Unsound for inheritance
- GUI Error Detector (Zhang, Lu, Ernst, ISSTA'12)
  - Unsound automatic control flow analysis
  - Similar false-positive rate
  - No guarantee (we found a new bug)
- Thread Coloring (PPoPP'10)
  - Supports arbitrary declared thread roles
  - Complicated design
  - Unclear annotation burden
- None support polymorphism (e.g. run())

JavaUI

- Polymorphic effect system for UI bugs
- Simple & effective on > 140,000 LOC
  - Found 9 bugs, modest annotation burden
  - Lessons for future effect systems

http://github.com/csgordon/javaui

http://types.cs.washington.edu

Backup Slides

Checked Exceptions for UI Threading

```
class JLabel{
      void setText(String) throws WrongThreadException;
}
…
void updateView() { myJLabel.setText("New Title"); } // ERROR
```

+ Easily marks code that must run on UI thread!
+ Potentially sound
  - If user catching were prohibited
  - It would be a sound static effect system for these errors
– Breaks legacy code