Context-oriented Programming for Mobile Devices: JCop on Android

Christopher Schuster\textsuperscript{1}, Malte Appeltauer\textsuperscript{2}, Robert Hirschfeld\textsuperscript{2}

\textsuperscript{1} University of California, Davis, USA
\textsuperscript{2} Software Architecture Group, Hasso-Plattner-Institut, Germany

Workshop on Context-oriented Programming (COP) 2011, Lancaster, UK

July 25, 2011
Outline

1. Introduction
2. Background
   - JCop
   - Android
3. COP for Android
   - Example Android Application
   - Challenges for a JCop-Android Integration
   - Example application with JCop
4. Evaluation
5. Future Work
6. Conclusion
Context of mobile applications

- Location
- Battery status
- Network availability and bandwidth
- Airplane mode, Silent mode
Context of mobile applications

- Location
- Battery status
- Network availability and bandwidth
- Airplane mode, Silent mode
- Date and Time
- User language and preferences
- Mobile device (screen resolution, processor speed, ...)
- etc.
Context-oriented programming on mobile devices

Context-oriented programming as solution?
Context-oriented programming on mobile devices

Context-oriented programming as solution?

Lack of ready-to-use implementation on current mobile platforms
Context-oriented programming as solution?

Lack of ready-to-use implementation on current mobile platforms

Android is a good target

- open source
- free development tools available
- uses popular Java programming language
Context-oriented programming on mobile devices

Context-oriented programming as solution?

Lack of ready-to-use implementation on current mobile platforms

Android is a good target

- open source
- free development tools available
- uses popular Java programming language

⇒ Let’s explore how to provide COP for Android!
JCop\textsuperscript{1}

- COP extension to Java

JCop\textsuperscript{1}

- COP extension to Java
- Behavioral variations as partial methods in layers

**JCop**

1. COP extension to Java
2. Behavioral variations as partial methods in layers
3. Layer activation/deactivation
   - either explicit (using `with`)

---

JCop$^1$

- COP extension to Java
- Behavioral variations as partial methods in layers
- Layer activation/deactivation
  - either explicit (using `with`)
  - or declarative (using a context object and pointcuts)

---

Android overview

- Linux-based open source software stack for mobile devices
- Java as intended language for application developers
  - besides Scala, JRuby and native C libraries
- Application code needs to subclass and implement certain Android classes to fit in the framework
Example application: Astronomy Picture of the Day

```java
class Main extends Activity {
    void onCreate(..) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        new DownloadEntryTask().execute(this);
    }
}

class DownloadEntryTask extends AsyncTask {
    Entry doInBackground(..) {
        client = AndroidHttpClient.newInstance();
        Entry e = loadEntry();
        loadPicture(e);
    }
}
```
Basic Android concepts

**Activity**
- include graphical user interface
- usually starting point of an application
- cannot execute blocking calls
Basic Android concepts

**Activity**
- include graphical user interface
- usually starting point of an application
- cannot execute blocking calls

**AsyncTasks**
- CPU-intensive tasks
- blocking calls like webservice invocations
- cannot access user interface
Basic Android concepts

**Activity**
- include graphical user interface
- usually starting point of an application
- cannot execute blocking calls

**AsyncTasks**
- CPU-intensive tasks
- blocking calls like webservice invocations
- cannot access user interface

**No network available**
- Current implementation would fail silently
- Traditional implementation would have to use conditional statements
Challenges for a JCop-Android Integration

- Thread control done by Android and separated into GUI and blocking threads
- Framework approach based on callbacks
- Technical limitations
  - No custom classloaders
  - No dynamic code generation
  - No bytecode manipulation
- dx tool makes certain assumptions
  - private method calls and constructors will never use the virtual method table
  - certain class flags are not used
Static contexts

Problem:

- Callbacks make control flow-based with context activation difficult
- Explicit context activation hard due to lack of control over thread creation
- Most context-dependent behavior in mobile applications driven by external events and information from the environment like sensor data
Static contexts

Problem:

- Callbacks make control flow-based with context activation difficult
- Explicit context activation hard due to lack of control over thread creation
- Most context-dependent behavior in mobile applications driven by external events and information from the environment like sensor data

Our proposed solution:

- Language extension to change context activation
- static context is assumed to be always active
- Declarative layer activation by using when statements and pointcuts
Example application with JCop

Entry e = loadEntry(main);
loadPicture(e, main);

public layer OfflineEntry {
    public Entry DownloadEntryTask
        .loadEntry(Context ctx) {
            return new Entry("No network available");
        }
}

public static context NetworkContext {
    when (!Network.connected()) {
        with (OfflineEntry);
    }
}
Demo
Evaluation

- COP successfully applied to the example application
- JCop processes the bytecode of the application \(\Rightarrow\) works on Android 2.3 and later
- Context data, like sensor values, are only accessed during execution of context-dependent code
Evaluation

- COP successfully applied to the example application
-JCop processes the bytecode of the application ⇒ works on Android 2.3 and later
- Context data, like sensor values, are only accessed during execution of context-dependent code
- Performance evaluation
  - No real benchmark yet
Evaluation

- COP successfully applied to the example application
- JCop processes the bytecode of the application ⇒ works on Android 2.3 and later
- Context data, like sensor values, are only accessed during execution of context-dependent code
- Performance evaluation
  - No real benchmark yet
  - But evaluated whether overhead is feasible
Evaluation

- COP successfully applied to the example application
- JCop processes the bytecode of the application $\Rightarrow$ works on Android 2.3 and later
- Context data, like sensor values, are only accessed during execution of context-dependent code
- Performance evaluation
  - No real benchmark yet
  - But evaluated whether overhead is feasible
  - Compared runtime performance for simple code snippit with three different implementation strategies
Results of the performance evaluation

<table>
<thead>
<tr>
<th>Approach</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>No context-dependency</td>
<td>2901ms</td>
</tr>
<tr>
<td>Conditional if branching</td>
<td>2959ms</td>
</tr>
<tr>
<td>JCop on Android</td>
<td>3450ms</td>
</tr>
</tbody>
</table>

Table: Measured runtime performance

- JCop adds additional overhead for layer management
- Implementation on Android not significantly different
- Measured runtime performance still within reasonable limits

---

Virtual Android 2.3.1 device inside the Android Emulator on an Intel Core Duo processor with 1.66 GHz running a Linux 2.6.35 kernel
Future Work

- Using a realistic benchmark for performance evaluation
Future Work

- Using a realistic benchmark for performance evaluation
- Evaluate effects of COP on code quality and modularization by using code metrics
  - e.g. lines of code, cyclomatic complexity or class cohesion
Future Work

- Using a realistic benchmark for performance evaluation
- Evaluate effects of COP on code quality and modularization by using code metrics
  - e.g. lines of code, cyclomatic complexity or class cohesion
- Adding additional, more complex, types of context variables like GPS location or user preferences
Conclusion

- Context important for mobile applications
Conclusion

- Context important for mobile applications
- Successfully used JCop to implement an Android application
Conclusion

- Context important for mobile applications
- Successfully used JCop to implement an Android application
- JCop’s pointcuts and **static** contexts useful for modifying behavior without relying on the application’s control flow
Conclusion

- Context important for mobile applications
- Successfully used JCop to implement an Android application
- JCop’s pointcuts and static contexts useful for modifying behavior without relying on the application’s control flow
- Initial results for performance evaluation look reasonable
Conclusion

- Context important for mobile applications
- Successfully used JCop to implement an Android application
- JCop’s pointcuts and static contexts useful for modifying behavior without relying on the application’s control flow
- Initial results for performance evaluation look reasonable
- Applying COP to mobile applications seems promising
Conclusion

- Context important for mobile applications
- Successfully used JCop to implement an Android application
- JCop’s pointcuts and \textit{static} contexts useful for modifying behavior without relying on the application’s control flow
- Initial results for performance evaluation look reasonable
- Applying COP to mobile applications seems promising
- Further research possible for location-based applications
Appendix: Static behavior adaptation

- Repeated execution of one method
- Statically changed after 1000 invocations

```java
for (i = 0; i < 1000; i++) {
    countFromZero();
}
for (i = 0; i < 1000; i++) {
    countFromOne();
}
```

- Average measured runtime: 2901ms
Appendix: Conditional behavior adaptation

- Repeated execution controlled by an `if` statement

```java
boolean state = false;
for (int i = 0; i < 2000; i++) {
    if (i == 1000) {
        state = true;
    }
    if (state) {
        countFromOne();
    } else {
        countFromZero();
    }
}

Average measured runtime: 2959ms
Appendix: Behavior adaptation by JCop on Android

- Variations in the repeated execution by using JCop

GlobalState.SetActive(false);
for (int i = 0; i < 2000; i++) {
    if (i == 1000) { GlobalState.SetActive(true); }
    countZero();
}

public layer CountLayer {
    public int Main.countZero() {
        Main.CountOne();
    }
}

public static context CountContext {
    when (GlobalState.isActive()) {
        with (CountLayer);
    }
}

- Average measured runtime: 3450ms