

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

Transformation-based Library Adaptation for Embedded Systems

Victor Winter, Azamat Mametjanov, Steven E.
Morrison, James A. McCoy, and Gregory L. Wickstrom

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

Tenth IEEE High Assurance Systems Engineering
Symposium
Dallas, Texas
November 14 – 16, 2007

Outline

Overview

Considerations

A Transformation-based Perspective

Example

Summary of Results

Conclusion

Library Adaptation

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

Problem and Solution

The Problem: Enable embedded systems to be programmed at a higher level of abstraction.

The Solution: Make libraries defining high-level abstractions accessible within an embedded programming environment.

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

[Overview](#)

[Considerations](#)

[A Transformation-
based
Perspective](#)

[Example](#)

[Summary of
Results](#)

[Conclusion](#)

An Instance of the Problem

- ▶ Computational Environment:
 - ▶ The **SCORE Processor** – A JVM-based platform designed for use in embedded systems.
- ▶ Specific Considerations:
 - ▶ **Restricted computing capabilities** – For example, floating point operations are not supported.
- ▶ Target Libraries:
 - ▶ To the extent possible, we want to make **general purpose libraries** such as `java.lang` and `java.util` available to the embedded systems programmer in this environment.

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

Overview

Considerations

A Transformation-
based
Perspective

Example

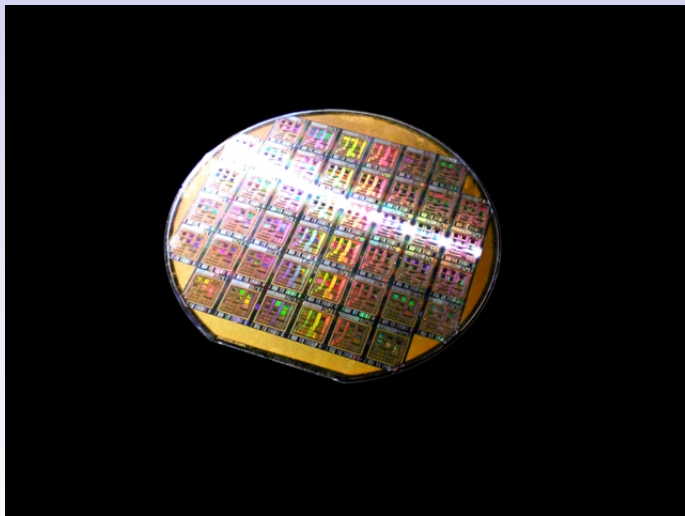
Summary of
Results

Conclusion

The SCORE Processor

Library Adaptation

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom



Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

A Quick Summary

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

- ▶ The software that we want to adapt is a **Java library**.
- ▶ The computational restriction we are considering is a **floating point operation**.
- ▶ The target platform is the **Score processor**.

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

Our Approach to Library Adaptation

- ▶ Is **based on removing** (i.e., filtering out) components of a library that depend on unsupported computations.
- ▶ Assumes that library components are **loosely coupled**.
- ▶ **DISCLAIMER:** We do not attempt adaptation that is based on re-implementing components.

Adaptation is Similar to Program Slicing

The approach presented is conceptually **similar to program slicing** where:

- ▶ One is given program p and an initial set of constructs R .
- ▶ One produces a output program p' that is obtained by **removing**, from p , all code depending either directly or indirectly on r when $r \in R$.

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

A General Purpose Approach

Library Adaptation

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

Our approach is **general-purpose** in the sense that:

- ▶ The techniques can be **adapted** to any language.
- ▶ For a given language, the set **R can be arbitrary.**

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

Theoretical Considerations

An understanding of **removal**. That is, the **UNIT** of removal.

- ▶ Well-formed
 - ▶ syntactically
 - ▶ semantically
- ▶ **Conservative approximations** – remove too much rather than too little (to assure semantic consistency)
- ▶ **Safeguards**
 - ▶ the Java compiler
 - ▶ code inspection

Syntactic Considerations

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

`int x = (int) 1.0;` → `int x = (int) _____;`

`double x;` → `_____ x;`

`float foo() {...}` → `_____ foo() {...}`

`if (y < 1.0) x = 0;` → `if (y < _____) x = 0;`

[Overview](#)

[Considerations](#)

[A Transformation-
based
Perspective](#)

[Example](#)

[Summary of
Results](#)

[Conclusion](#)

Semantic Considerations

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

```
int x = 0;  
int f(){ double x = 1.0; return (int) x;}  
→  
int x = 0;  
int f(){ return (int) x;}
```

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

Theoretical Considerations

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

An understanding of **dependency**.

- ▶ A **direct dependency** is recognized at the syntactic level.
- ▶ A **transitive dependency** is recognized at the semantic level and involves analysis that is based on scope and type.

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

Semantic Considerations

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

```
int ask() {return (int)answer();}
```

```
double answer(){return 3.33;}
```

→

```
int ask() {return (int)answer();}
```

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

Practical Considerations

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

- ▶ Infrastructure supporting **automatic activities**:
 - ▶ parser
- ▶ Infrastructure supporting **manual activities** (e.g., code inspection):
 - ▶ preservation of comments
 - ▶ pretty-printer (i.e., un-parser)

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

A Transformation-based Perspective

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

- ▶ A **Java library** is modelled as a **set of classes**:

$$\mathcal{Lib} = \{Class_1, \dots, Class_n\}$$

- ▶ The **adaptation** of \mathcal{Lib} is viewed primarily in terms of the adaptation of individual classes, with analysis results exchanged between classes.

Overview

Considerations

A Transformation-based
Perspective

Example

Summary of
Results

Conclusion

Class-level Transformation

- ▶ Let C , with or without subscripts, denote an arbitrary Java class. The adaptation of a class is viewed from a transformational perspective as follows:

$$C_{initial} \xrightarrow{T_1} C_1 \xrightarrow{T_2} \dots \xrightarrow{T_{final}} C_{final}$$

where T_i denotes a transformation step that **removes** a portion of the class.

[Overview](#)

[Considerations](#)

[A Transformation-based Perspective](#)

[Example](#)

[Summary of Results](#)

[Conclusion](#)

Constraints on Transformation

1. Transformational steps should yield results that are syntactically well-formed.
2. The result of transforming $C_{initial}$ should yield a class C_{final} , with respect to which $C_{initial}$ is consistent.

$$C_{final} \sqsubseteq C_{initial}$$

3. C_{final} must be compatible with the SCORE platform.
4. Removal must be **minimal**; otherwise the empty class would be an acceptable adaptation.

Removal-oriented Abstractions for Java

- ▶ A **Java library** is modelled as a **set of classes (C)**.
- ▶ A **Java class** is modelled as a set of **members**.

```
class name {  
    inner classes  
    fields  
    methods  
    constructors  
    instance initializers  
    static initializers  
}
```

- ▶ The **unit of removal** is defined to be a class member.

The Syntactic Categories for Class Members

Library Adaptation

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

$M ::= x \mid mb$

$x ::= \text{inner-classes}$

$mb ::= m \mid b$

$m ::= \text{field-declaration}$
| $\text{method-declaration}$
| $\text{constructor-declaration}$

$b ::= \text{instance-initializer}$
| $\text{static-initializer}$

This notation enables us to describe terms in an abstract fashion.

[Overview](#)

[Considerations](#)

[A Transformation-based Perspective](#)

[Example](#)

[Summary of Results](#)

[Conclusion](#)

An Abstract Transformation

- ▶ Henceforth, we will view classes as **terms**.
- ▶ We will write $C[\dots t \dots]$ to denote an occurrence of the term t within the class C .
- ▶ The removal of the subterm t from the term C as follows:

$$C[\dots t \dots] \xrightarrow{\mathcal{T}} C[\dots \epsilon \dots]$$

The Axioms and Rule Defining R

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

$$\overline{FpLiteral} \in R_C$$

$$\overline{Modifier[\text{strictfp}]} \in R_C$$

$$\overline{BasicType[\text{float}]} \in R_C$$

$$\overline{BasicType[\text{double}]} \in R_C$$

$$\frac{C[\dots m \dots] \quad t \in R_C \quad m[\dots \hat{t} \dots]}{ref\text{-to}(m) \in R_C}$$

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

Example: ref-to

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

```
class A {  
  double x = 1.0;  
  int y;  
  int z;  
  { int x = (int)this.x; y = x; }  
  { z = (int) x; }  
}
```

[Overview](#)

[Considerations](#)

[A Transformation-
based
Perspective](#)

[Example](#)

[Summary of
Results](#)

[Conclusion](#)

Removal Rules

$$\frac{C[\dots mb \dots] \quad t \in R_C \quad mb[\dots \hat{t} \dots]}{C[\dots mb \dots] \rightarrow C[\dots \epsilon \dots]} \quad (\text{T-adapt1})$$

$$\frac{C[\dots x \dots] \quad t \in R_C \quad x[\dots \hat{t} \dots]}{x[\dots \hat{t} \dots] \rightarrow x[\dots \epsilon \dots]} \quad (\text{T-adapt2})$$

Example – Input

```
class Cafe {  
  
    float fabulous = 0xCAFEBABE; // catch me if you can  
    int innocent = (int) fabulous; // masquerade as int  
  
    Cafe(int innocent) {  
        this.fabulous = 0xDD; // explicit ref to FP field  
        this.innocent = innocent;  
    }  
  
    int environment(int innocent) { // re-decl in params  
        innocent = 0xFACADE;  
        return innocent;  
    }  
  
    int environment() {  
        int harmless = innocent; // use comes before decl  
        int innocent = harmless;  
        return innocent;  
    }  
  
    int order(int choice){  
        int waitress;  
        if (choice == 0xC0FFEE) {  
            int innocent = 0xBABE; // re-decl in local if-block  
            waitress = innocent;  
        } else {  
            waitress = innocent; // indirect ref to FP field  
        }  
        return waitress;  
    }  
}
```

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

[Overview](#)

[Considerations](#)

[A Transformation-
based
Perspective](#)

[Example](#)

[Summary of
Results](#)

[Conclusion](#)

Example – Output

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

```
class Cafe {  
    int environment(int innocent) {  
        // re-decl in params  
        innocent = 0xFACADE;  
        return innocent;  
    }  
}
```

[Overview](#)

[Considerations](#)

[A Transformation-
based
Perspective](#)

[Example](#)

[Summary of
Results](#)

[Conclusion](#)

SCORE: Unsupported Feature Profile

synchronized serialized access to data or execution of code; compiles to unsupported opcodes
`monitorenter, monitorexit`

volatile thread coherence; multi-threading is not supported

transient I/O persistence; persistence is not supported

assert assertion-checking with dependencies on reflection features of a JVM; reflection is not supported

native code implemented in target platform's native code; not supported unless implemented on the embedded platform

floating-point keywords `float, double, strictfp`

other unsupported classes: e.g. `ClassLoader, Compiler, etc.`

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

[Overview](#)

[Considerations](#)

[A Transformation-based
Perspective](#)

[Example](#)

[Summary of
Results](#)

[Conclusion](#)

Highlights: `java.util` stats

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

The util library consists of 96 source files with a number of files containing more than 1000 (SLOC).

	sloc	orig	auto	man	end	sloc'
Arrays	4173	161	43	1	117	2788
Date	1315	53	45	0	8	216
Total	42343	2177	442	40	1695	29199

[Overview](#)

[Considerations](#)

[A Transformation-based Perspective](#)

[Example](#)

[Summary of Results](#)

[Conclusion](#)

An Example Requiring Manual Intervention

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

The class `Throwable` required manual adjustment.

- ▶ The removal of 4 constructors in `Throwable` triggered compile errors in all `Error` and `Exception` classes.
- ▶ The constructors were put back; statements within the constructors referencing native code were manually removed (`fillInStackTrace()`)
- ▶ Root cause: cohesion
- ▶ Solution: minor re-implementation

[Overview](#)

[Considerations](#)

[A Transformation-based Perspective](#)

[Example](#)

[Summary of Results](#)

[Conclusion](#)

Effort

Initial investment of 3 man/months:

- ▶ 132 SLOC of Java lexer
- ▶ 277 SLOC of Java BNF
- ▶ 2655 SLOC of Java pretty-printer
- ▶ 1133 SLOC of transformation code

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

[Overview](#)

[Considerations](#)

[A Transformation-
based
Perspective](#)

[Example](#)

[Summary of
Results](#)

[Conclusion](#)

Payoff

- ▶ 1 day automated vs. 45 days manual transformation of the `java.util` library.
- ▶ We have adapted:
 - ▶ `java.util` with a $\approx 91\%$ migration rate
 - ▶ `java.lang` with a $\approx 98\%$ migration rate
- ▶ **Reuse**: Keep in mind that new versions of `java.lang` and `java.util` are released at regular intervals (approximately every 12 -18 months).

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

Industrial rule-of-thumb [Akers et al.]: investment into automation is well-justified if 75% of legacy code is migrated using automation.

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion

Conclusion

- ▶ Embedded Java programming needs library support.
- ▶ Manual library adaptation is resource-intensive.
- ▶ Program transformations are well suited to this kind of activity.
- ▶ Re-targeting of the libraries to heterogeneous Java-based embedded platforms with different instruction sets is easily handled.

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

[Overview](#)

[Considerations](#)

[A Transformation-
based
Perspective](#)

[Example](#)

[Summary of
Results](#)

[Conclusion](#)

Victor Winter,
Azamat
Mametjanov,
Steven E.
Morrison, James
A. McCoy, and
Gregory L.
Wickstrom

The End

Overview

Considerations

A Transformation-
based
Perspective

Example

Summary of
Results

Conclusion