

A Case For Binary Component Adaptation

Modifying Components On The Fly

Urs Hölzle and Ralph Keller
 Department of Computer Science
 University of California, Santa Barbara
<http://www.cs.ucsb.edu/oocsb/bca>

Motivation

- OOP vision:
 - Pervasive component reuse
 - Large-scale component market
 - Programs are built mostly of existing components
- Unfortunately: Combining independently developed components is hard
- Binary Component Adaptation makes integration easier!

Binary Component Adaptation

2

Talk Outline

- Problems with component-based programming
- Binary Component Adaptation
- Binary Compatibility
- Performance
- Conclusions

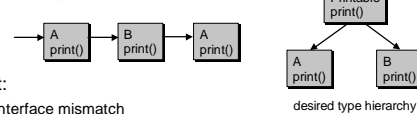
Binary Component Adaptation

3

The Integration Problem



- Both components provide functionality for printing
- Goal: Integrate A and B in list:

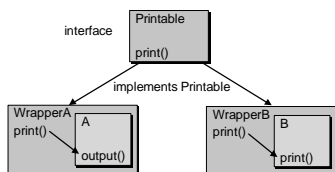


- But:
 - Interface mismatch
 - No common supertype

Binary Component Adaptation

4

Using Wrapper Classes

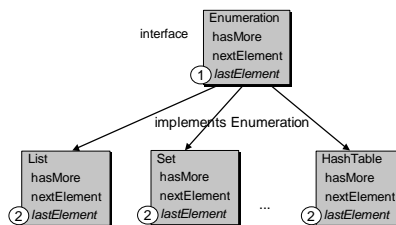


- Disadvantages:
 - Additional programming effort
 - Overhead (wrapping and unwrapping)
 - Redundant type hierarchy
 - Difficult subtyping problems [Hölzle, ECOOP '93]

Binary Component Adaptation

5

The Interface Evolution Problem



Binary Component Adaptation

6

Interface Evolution Problem

- Interface change must be reflected immediately by all classes
- Default implementation for each class:


```
object lastElement() {
    object o = null;
    while (hasMore()) o = nextElement();
    return o;
}
```
- But:
 - Generally: no source code
 - Open set of classes (dynamic linking)

Binary Component Adaptation

7

Problem Summary

- Minor incompatibilities prevent pervasive reuse
- Each re-user may have different requirements
- Source code modifications are not practical
 - not available
 - needs recompile before executing
 - need to keep track of different compiled versions
 - tracking new releases is tedious

Binary Component Adaptation

8

What We Want

- Easy-to-use, flexible adaptation mechanism
- No source code required
- “In place” changes of type (no wrappers)
- Automatic integration with future releases

Binary Component Adaptation

9

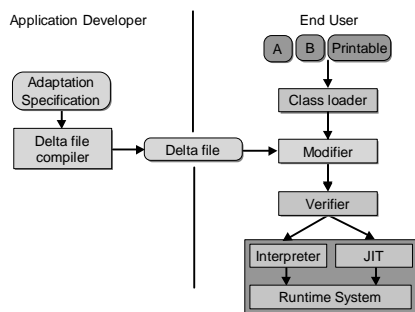
Talk Outline

- Motivation: Problems with component-based programming
- **Binary Component Adaptation**
- Binary Compatibility
- Performance
- Conclusions

Binary Component Adaptation

10

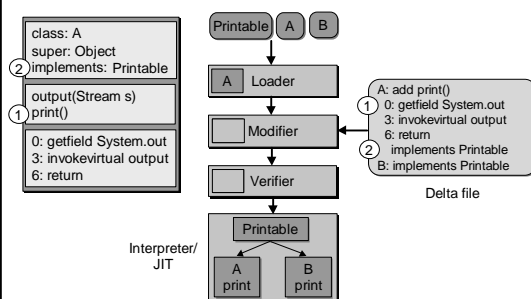
BCA Overview



Binary Component Adaptation

11

Modifying Components On The Fly



Binary Component Adaptation

12

Binary Component Adaptation

- Directly modifies component
- Uses type information in compiled component
 - Requires no source code
 - Works even on third-party components
- Keeps changes separately from components
 - No changed components delivered to client
- Performs component modification during class loading

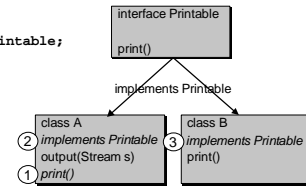
Binary Component Adaptation

13

Adaptation for Integration Problem

```

delta class A {
  ① add method void print() { output(System.out); }
  ② add implements Printable;
}
delta class B {
  ③ add implements Printable;
}
    
```



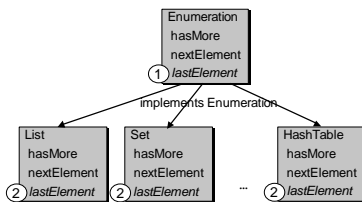
Binary Component Adaptation

14

Adaptation for Evolution Problem

```

delta interface Enumeration {
  ① add method Object lastElement();
}
delta implements Enumeration {
  ② add method Object lastElement() { ... while(hasMore())... }
}
    
```



Binary Component Adaptation

15

Talk Outline

- Problems with component-based programming
- Binary Component Adaptation
- **Binary Compatibility**
- Performance
- Conclusions

Binary Component Adaptation

16

What Modifications Are Possible?

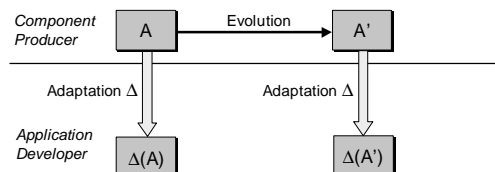
- Limited by amount of symbolic information
- For Java: can change almost everything
 - Add fields, methods
 - Add implemented interfaces
 - Delete fields, methods
 - Rewrite byte codes
- But: Binary compatibility must be preserved

Binary Component Adaptation

17

Binary Compatibility

- Goal: Adaptation is valid on any future release of A



Binary Component Adaptation

18

Binary Compatible Modifications

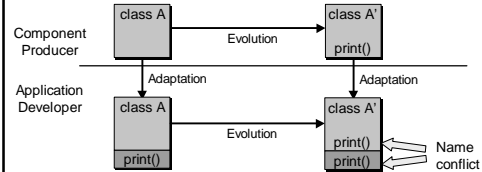
- Very flexible even with binary compatibility restriction:
 - Add fields & methods
 - Rename classes, fields & methods
 - Reimplement (wrap) methods
 - Add interface
- Plus: handles open systems
 - Rename classes or class members
 - Extend interfaces

Binary Component Adaptation

19

One Little Complication...

- Problem: Changes between Evolution and Adaptation may lead to name conflicts

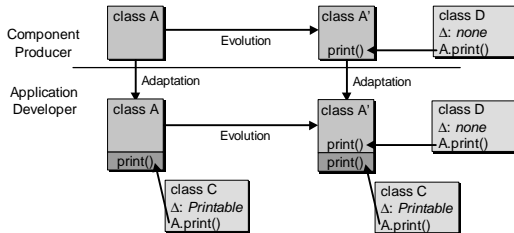


Binary Component Adaptation

20

Solution: Mark Class Files

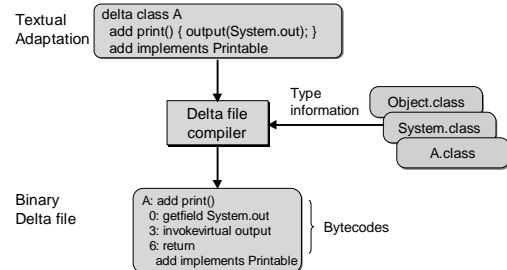
- Mark class files compiled against adapted class
- Resolve conflicts by renaming



Binary Component Adaptation

21

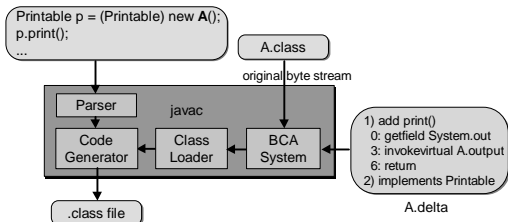
Delta File Compilation



Binary Component Adaptation

22

Compiling Against Adapted Classes



Binary Component Adaptation

23

Talk Outline

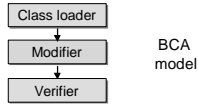
- Problems with component-based programming
- Binary Component Adaptation
- Binary Compatibility
- Performance
- Conclusions

Binary Component Adaptation

24

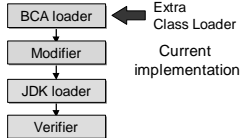
Overhead

- Introduced by modifier at load-time
- But code executes at full speed



- Our implementation:

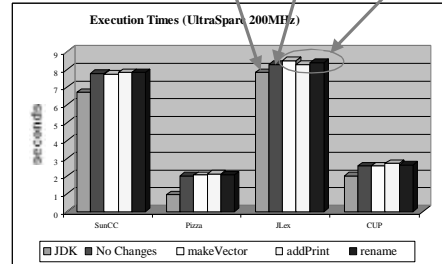
- Easy to integrate into JDK
- Independent of VM data structures



Binary Component Adaptation

25

Performance



Binary Component Adaptation

26

Status and Future Work

- Implemented:
 - BCA-aware virtual machine
 - Delta file compiler
 - BCA-modified javac
- Freely available for SPARC/Solaris:
 - <http://www.cs.ucsb.edu/oocsb/bca>
- Future:
 - Type checker for delta files
 - More modifications (such as refactorings)

Binary Component Adaptation

27

BCA Vs. Source Code Modification

- Just as many kinds of possible changes
- Guarantees binary compatibility
- Handles open systems and unknown, dynamically loaded classes
 - Rename classes or class members
 - Extend interfaces
 - Etc.
- More flexible than source code!

Binary Component Adaptation

28

Conclusions

Binary Component Adaptation

- Modifies components "in place" (type identity)
- Uses type information in compiled component
 - Requires no source code
 - Works on any component
- Guarantees release-to-release compatibility
- Is efficient enough to be performed at load-time

BCA can significantly improve reusability of components

Binary Component Adaptation

29

A Case For Binary Component Adaptation

Modifying Components On The Fly

Urs Hölzle and Ralph Keller
 Department of Computer Science
 University of California, Santa Barbara
<http://www.cs.ucsb.edu/oocsb/bca>