

Extensible Plug-ins for Aspect-Oriented Programming

Macneil Shonle * Ankit Shah

Adaptive Programming

- Aspects were Specialized
  - DJ concentrated mostly on Traversals
  - AspectJ was dominated by the JoinPoint Model
  - COOL dealt with Co-ordination and Synchronization of Threads
  - RIDL concerned with Remote Invocation

AspectJ

- AspectJ attempted to include COOL
- JoinPoint Model did not provide for synchronization support

- Result: COOL misses AspectJ bus

Basis for XAspects

- Explore Possibility for uniting various Cross-Cutting Concerns under one roof
  - DAJ does this
    However,
    - Different File Names identify domain specific aspects
    - No Scope for expansion without rewrite
- Uniform Syntax for various domain specific aspects

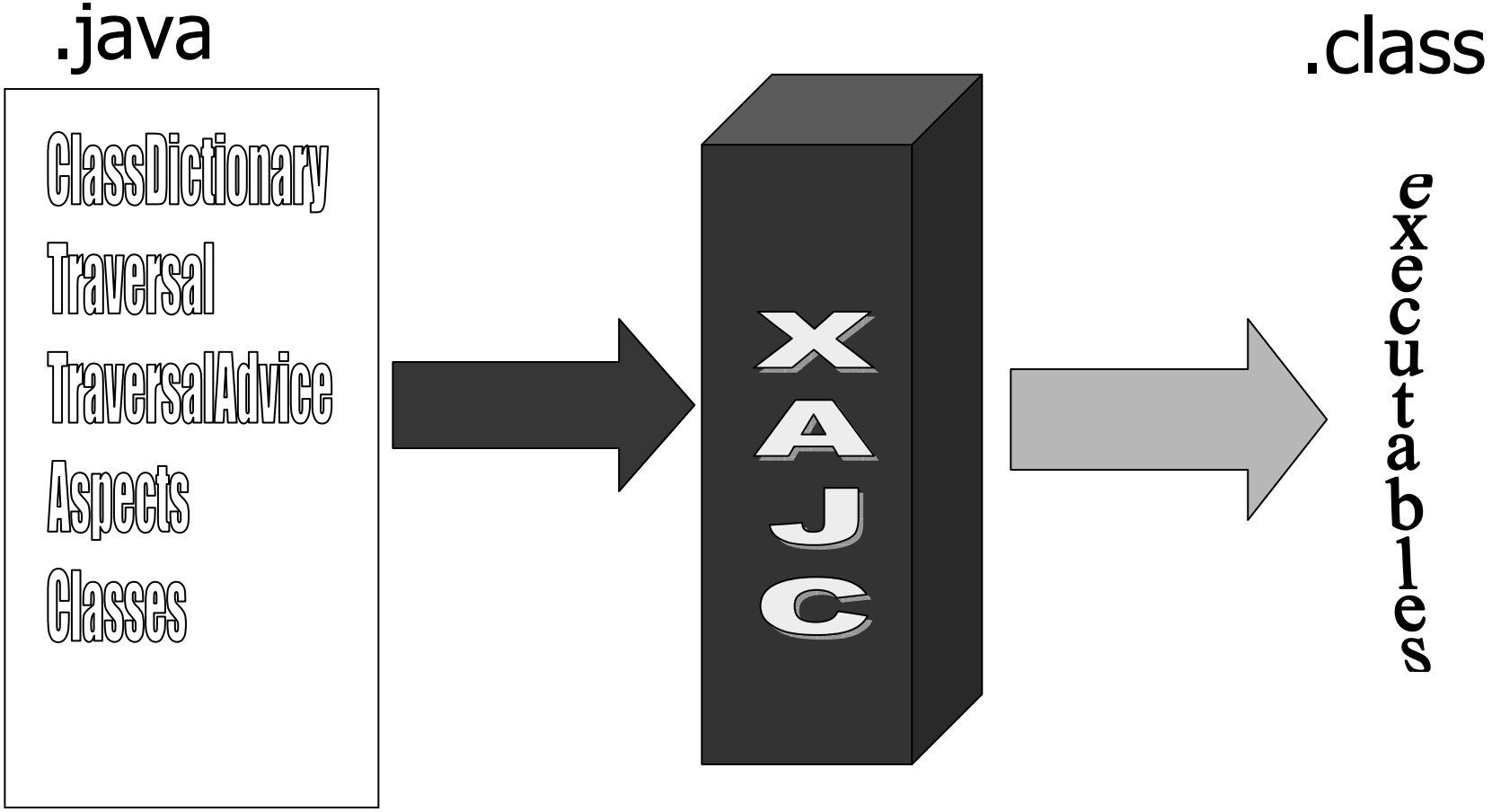
Goal for XAspects

- Integrate Domain Specific Aspects into General Purpose Aspect Oriented Programming Solutions
- Plugin Architecture to compile Domain Specific Aspects
- Plugins to generate Aspect J Compatible Code from User Programs
- AJC Compiler as Back end to generate final JVM compatible byte codes

Journey Begins ?

- YES !!!
- Target:
  - Develop a Prototype for XAspects that meets the above Goals
  - Provide support for existing Domain Specific Aspects
  - Support Uniform Syntax for all Aspects
  - Extensible Design

XAspects Functioning



XAspects Architecture - I



X A J C

Use: Input Code

Generate: Stubs and Interfaces

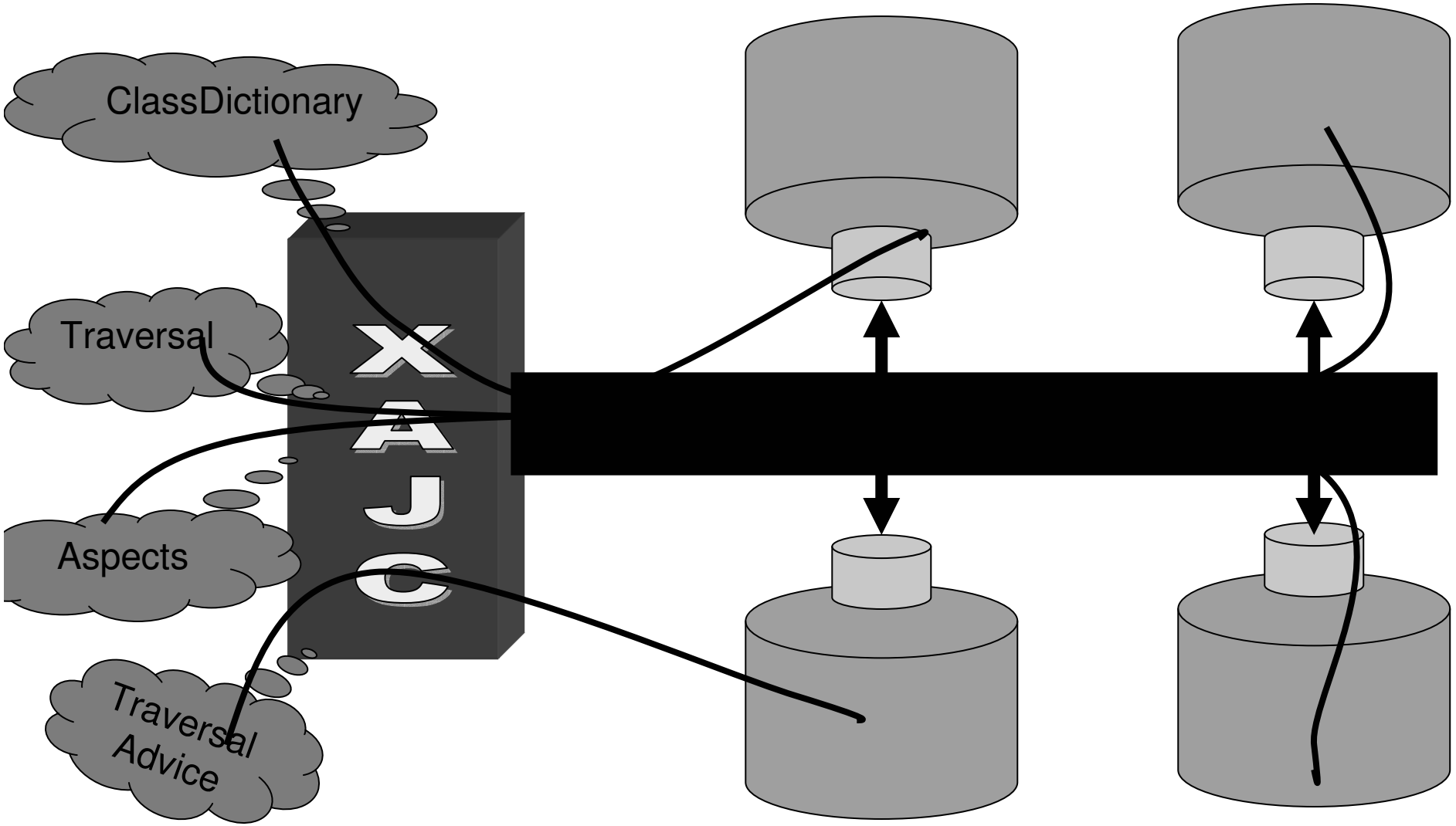
Compile Stubs and Interfaces

Use: Input Code, Compiled Stubs and Interfaces

Generate: Final Code

Compile Generated Code

XAspects Architecture - II



Main Components of XAspects

- XAJC
  - A Wrapper around ajc Compiler
  - Parse in Input Files and pass information to appropriate Plugins
- Plugins
  - Read in Domain Specific aspect and produce corresponding AspectJ code
- Interface
  - Interface for data Transfer between XAJC and Plugins

XAspects: Design Issues

- XAJC Design
- Plugin Design
- Interface Design

XAspects: Interface Design

- Start with Interface Design?
  - It demanded the most attention !!!
  - Crucial to successful Design
- Components of Interface
  - CompilationEnvironment
  - AspectPlugin
  - AspectInfo

XAspects
Interface: CompilationEnvironment

- Record all aspects in use, File Names and File Numbers

- Inventory for Error and Warnings
  Re-adjust line numbers to reflect actual position in file

XAspects
Interface: AspectPlugin

- Super Class of all Plugins
- Sets up framework of Plugin Functioning
- Steps (for all Plugins):
  - Receive Code
  - Generate External Interfaces
  - Generate Code
    <Interface Byte Code is available>

XAspects
Interface: AspectPlugin

- Receive Code
  - Initial Design: Pass aspectID and body of aspect
  - DrawBack:
    - We need more than just body to generate classes
      imports
      modifiers
      inheritance information
  - Solution:
    - AspectInfo class is added

Anatomy of an Aspect

```
package i.like.this.course;

import thiscourse.techniques.*;

public abstract aspect (ClassDictionary) COM3360 extends fun
    implements goodCourse, nicePeople {

                    … body …

}
```

XAspects
Interface: AspectInfo

- Store all Information about an Aspect
  - FileName and line number
  - Package Info
  - Libraries Imported Info
  - Modifiers used
  - Plugin Info
  - Its ID (Aspect Name)
  - Inheritence Info
  - Body of code
- Populated by AJC
- Data read by Plugins

XAspects: AJC Design

- Wrapper around ajc just like DAJ
- 2 Modules:
  - Parse Input Source Code
  - Wrapper around ajc
    (Similar to DAJ)
    - Invokes compilation using plugins
    - Feed Generated Code to ajc for compilation

XAspects: AJC Design

- Parser
  - Read in .java source files and create an AspectInfo instance for EVERY aspect
  - Collect pure “Classes” in a separate file; since these don’t need any special processing

XAspects: Plugin Design

- Receive Code using ReceiveBody
- Generate Interfaces and Stubs when 1st pass is invoked
- Generate Final Complete Code when 2nd pass is invoked. Use Compiled Byte Codes if needed to get Structure Data
- Report all errors and warnings to CompilationEnvironment
  PRINT NOTHING TO STANDARD INPUT / ERROR

Plugins in XAspects

- ClassDictionary
  - Generate Classes and parsers in 1st pass
  - Return same files in 2nd pass
- Traversal
  - Generate Traversal Stubs in 1st Pass
  - Generate Traversals using Reflection in 2nd pass
- TraversalAdvice
  - Generate Visitor classes in 1st Pass
  - Return same files in 2nd pass
- AspectJ
  - Generate Aspects in 1st pass
  - Return same files in 2nd pass

Advantages of XAspects

- Domain Specific Aspects are easy to write
- Extensible and highly Flexible System
  - Plugins may be
    - added to provide additional functionality
    - Plugins may be removed when a technology is out-dated !!
    - Plugins may be upgraded

    without recompiling the whole XAJC

Advantages of XAspects

- Updates to underlying AJC compiler are automatically reflected
- Bring “Ex – Aspects” back to life
  (eg. COOL, RIDL)
- Support Forthcoming adaptive techniques
  (eg. Aspectual Collaborations)
- Program Modularization / Componentization
  Better Organization of Modules among files

XAspects: Issues

- Just a Powerful Macro System ?
- Inefficient due to large scale communication between main compiler and plugins
- Collaboration between plugins
- Clashes in Names / FileNames of various aspects

XAspects Implementation

- Initial Prototype XAJC 1.1 is ready for Release

XAspects: Future Work

- Modify ajc itself
- Provide ClassGraph instead of ByteCodes
  (Irony: Generate Code using Code !!)
- Improve Error / Warning Reporting System
- Handling of Flags

XAspects: Future Work

- Related Projects:
  - COOL
  - RIDL
- Support Upcoming Techniques
  - Personalities
  - Aspectual Collaborations

XAspects on the Web

- Detailed description available on XAspects Home Page

  http://www.ccs.neu.edu/home/mshonle/xaspects



Thank You

Macneil Shonle * Ankit Shah