# Aspect-Specification based on Structural Type Information

**Stefan Hanenberg, Mohammed Al-Mansari, Rainer Unland**
**University of Duisburg-Essen**

*Symposium on Applied Computing, Dyon, 2006*

# Content

- **Motivation**
  - Background: *Types and Crosscutting*
  - Example 1: Closable objects
  - Example 2: Applet implementing 2 interfaces
- **Solution**
  - Structural Types
  - Compound Types
- **Discussion and Conclusion**

# Background

- **Background aspect-orientation**
  - Crosscutting phenomenon still not that well-understood / formalized
  - Still no common view on good / valid pointcut languages
- **Type Systems**
  - Studying the impact of type systems on the crosscutting phenomenon (problem domain)
  - Studying the impact of type systems as solutions to crosscutting phenomenon (solution domain)
- **Overall intention**
  - Expressive pointcut languages
  - Good means to specify join point adaptations
  - Highly generic aspects

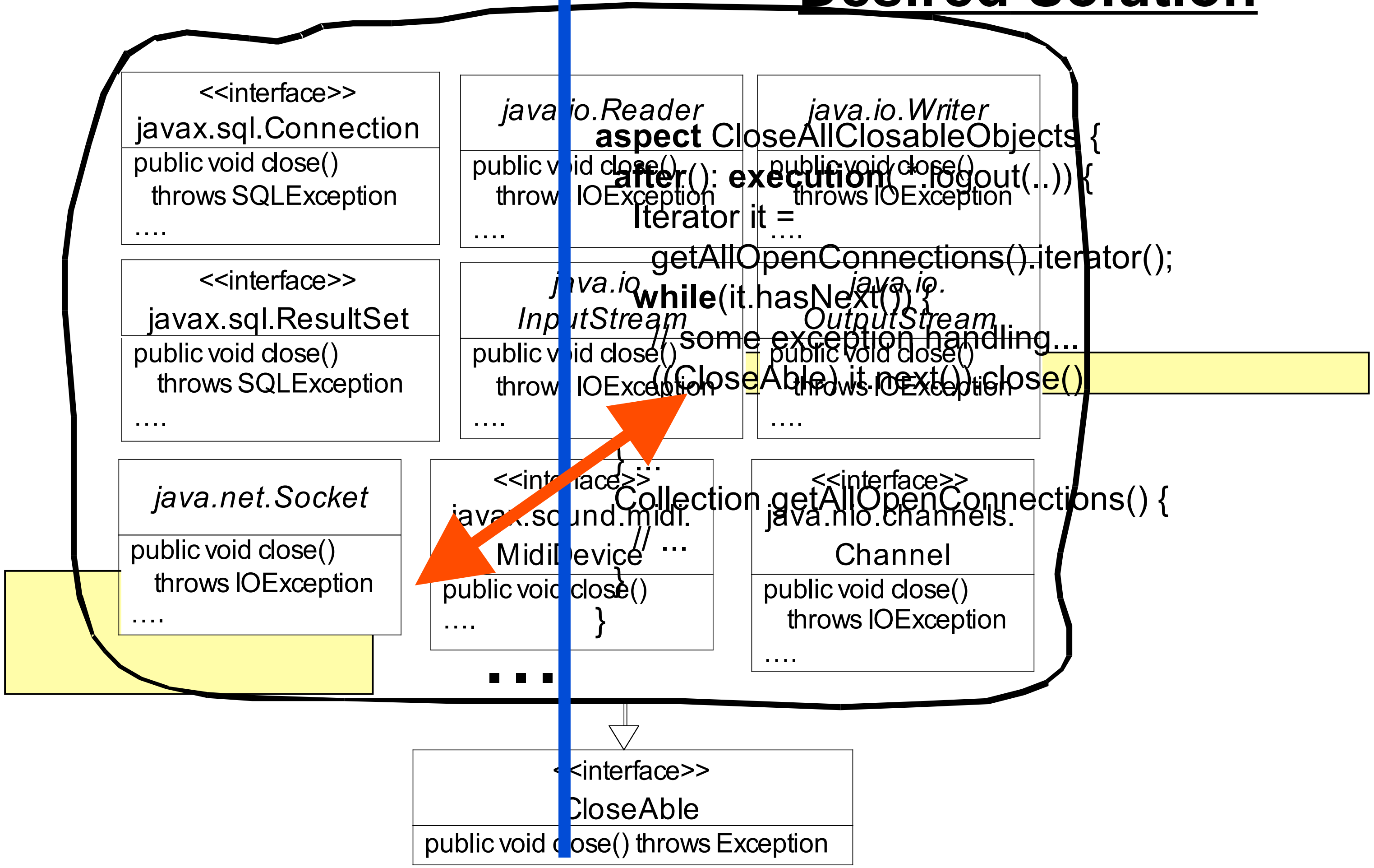
# Example: Closable Objects (1)

- Multi-user application
- Multiple objects representing ressources that need to be released (database connections, files, network connections, streams...)
  - javax.sql.Connection, java.io.Reader, etc.
- Multiple types provide corresponding methods close() (static crosscutting)
- Ressource objects should be closed (join point adaptation = *advice*) when user logs out (join point)

# Example: Closable Objects (2)

## Target hierarchy

## Desired Solution



<<interface>>
javax.sql.Connection
public void close()
throws SQLException
….

java.io.Reader
public void close()
throws IOException
….

java.io.Writer
public void close()
throws IOException
….

<<interface>>
javax.sql.ResultSet
public void close()
throws SQLException
….

java.io.InputStream
public void close()
throws IOException
….

java.io.OutputStream
public void close()
throws IOException
….

java.net.Socket
public void close()
throws IOException
….

<<interface>>
javax.sound.midi.MidiDevice
public void close()
….

<<interface>>
java.nio.channels.Channel
public void close()
throws IOException
….

…

<<interface>>
CloseAble
public void close() throws Exception

```
aspect CloseAllClosableObjects {
  after(): execution( *.logout(..)) {
    Iterator it =
      getAllOpenConnections().iterator();
    while(it.hasNext()) {
      // some exception handling...
      ((CloseAble) it.next()).close()
    }
  } ...
  Collection getAllOpenConnections() {
    // ...
  }
}
```

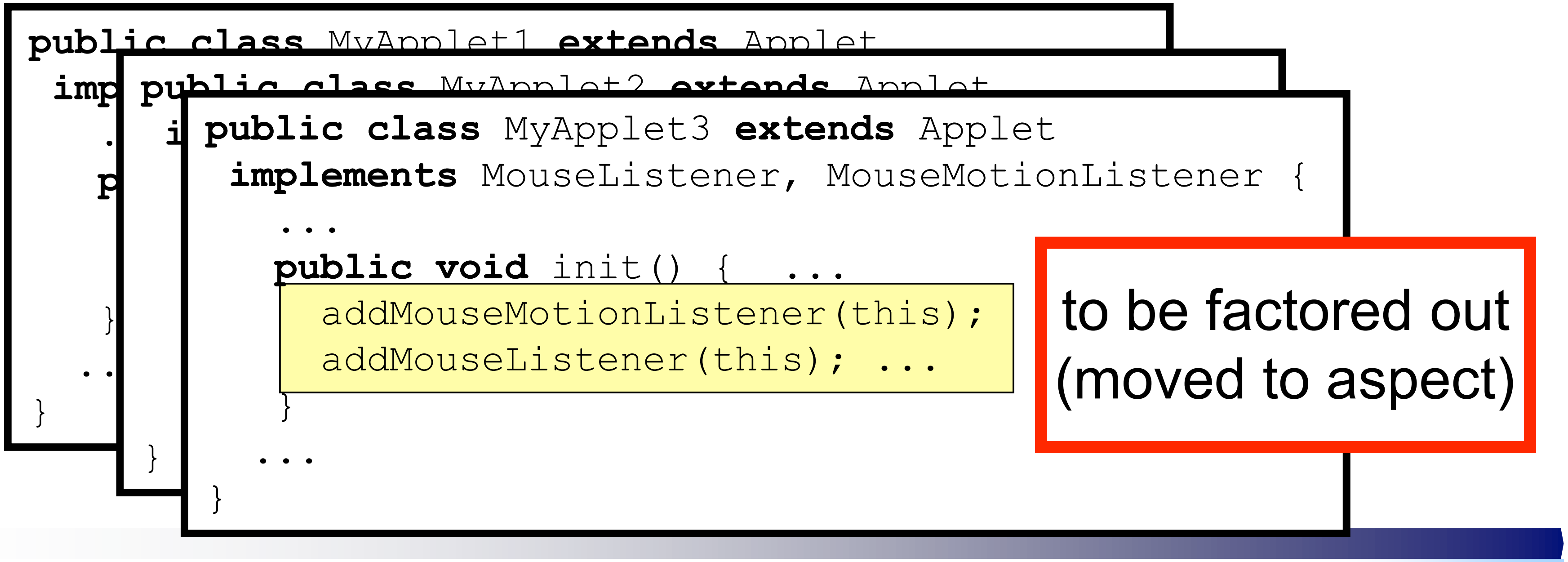
# Example: Closable Objects (3)

```
interface CloseAble { void close() throws Exception; }
aspect CloseAllCloseableObjects {
 declare parents:
  (javax.sql.Connection || javax.sql.ResultSet ||
   java.io.Reader|| java.io.Writer ||
   java.io.InputStream || java.io.OutputStream||
   java.net.Socket || java.nio.channels.Channel ||
   javax.sound.midi.MidiDevice .../* additional types */ )
 implements CloseAble;
 ...
```

- **Problem:**
  - Common type required (Closeable)
  - Desire to state „*all classes having a method close are subtypes of Closable*“
  - AspectJ solution
    - Enumeration-based crosscutting
    - Developer needs to find matching type by hand
    - Need to extend enumeration if new types are added

# Example: Applet Mouse Listener (1)

- Often used design guideline for Applets among <u>different projects</u>
  - Applet implements interfaces MouseListener and MouseMotionListener
  - Applet registers itself as corresponding listeners



```
public class MyApplet1 extends Applet
public class MyApplet2 extends Applet
```

```
public class MyApplet3 extends Applet
  implements MouseListener, MouseMotionListener {
    ...
    public void init() {  ...
      addMouseMotionListener(this);
      addMouseListener(this);  ...
    }
   ...
}
```

to be factored out (moved to aspect)

# Example: Applet Mouse Listener (2)

```
interface ListeningComponent
  extends MouseListener, MouseMotionListener {
  public void addMouseListener(MouseListener l);
  public void addMouseMotionListener(MouseMotionListener l);
}
aspect RegisterMouseListener {
  after(ListeningComponent c):
    initialization(* *.new(..)) && this(c){
      c.addMouseListener(c); c.addMouseMotionListener(c);
  }
}
```

- **Problem**
  - Common type required
  - Desire to define type „extension of Applet and implementing MouseListener and MouseMotionListener“
  - Target description of AspectJ not sufficient (requires further discussion)

# Summary so far

- **Crosscutting caused by type system / design guideline**
  - Example 1: not possible to send message close() to unknown type
  - Example 2: not possible to send addMouseListener to object of unknown type
- **Means for modularizing such crosscutting not sufficient**
  - Example 1: enumeration-based crosscutting
  - Example 2: not (*really*) possible to specify target class as combination of different types
- **Consequences**
  - Reconsidering type system
    ►**structural** and **compound types**

# Structural Types (1)

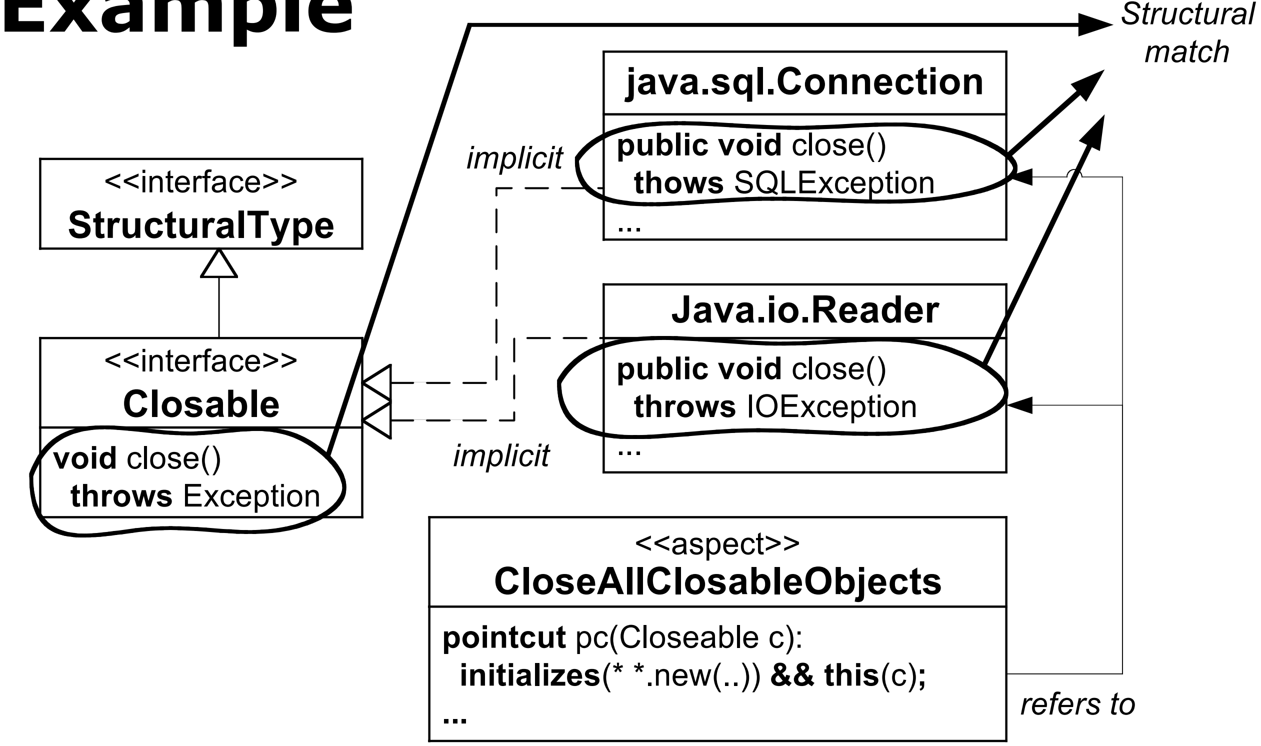
- **Structural Types**
  - Type relationship based on a type`s members, not on a types name
  - Example:

```
class A { void m() {...} }
class B { void m() {...} void n() {...}}
```

  - Type A is supertype of B (since it has corresponding members), hence
  - Type B can be used whenever A is expected (substitution principle)
- **AspectJST**
  - Structural types with AspectJ
  - Still sticking to nominal types (within method declarations)
  - Types extending interface Structural are structurally matched

# Structural Types (2)

- **Example**



- Closable extends StructuralType
- All matching types implicitly extend Structural
- Note: Subtype relationship valid with Exception

# Compound Types (1)

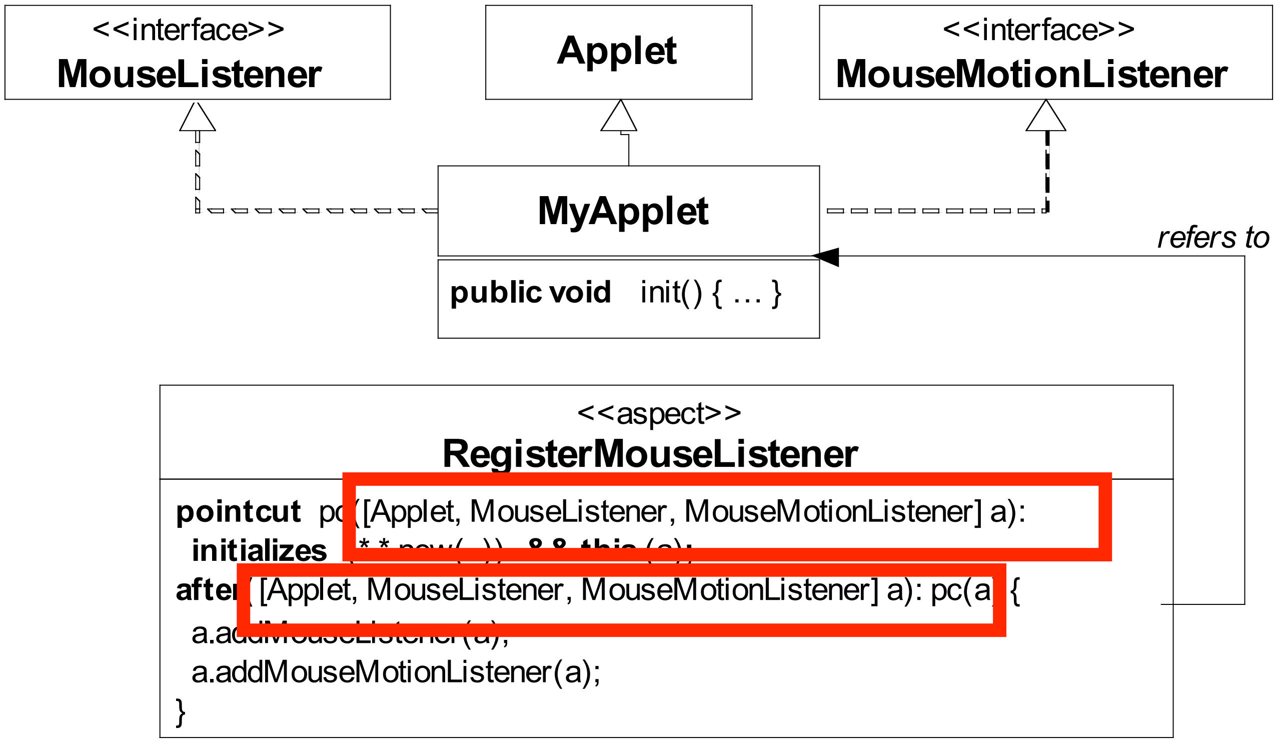
- **Specifying type by combining types**
  - enumeration of types (classes or interfaces)
  - a compound type is a subtype of each of its enumerated types
  - a subtype of compound type is a subtype of all enumerated types
- Example:
  - Type declaration

    [Applet, MouseListener, MouseMotionListerner]
  - In Applet example:

    MyApplet >

    [Applet, MouseListener, MouseMotionListerner]
- **AspectJST**
  - Static generation and compilation of compound types
  - Syntax extension of AspectJ for specifying compound types
  - New type relationships required

# Compound Types (2)



```
public void   init() { … }
```

```
pointcut  pc([Applet, MouseListener, MouseMotionListener] a):
  initializes (* *.new(..)) && this(a);
after([Applet, MouseListener, MouseMotionListener] a): pc(a) {
  a.addMouseListener(a);
  a.addMouseMotionListener(a);
}
```

- **Note**: a pure structural type would not be able to express same selection / adaptation semantics

# Conclusion & Discussion

- Type systems responsible for crosscutting
  - Crosscutting interface declarations
  - Crosscutting subtype relationships (in nominal type system)
  - No *good* aspect-oriented solution possible
- Two extensions to nominal type system
  - Structural types
  - Compound types
- In presense of such type systems well modularized crosscutting
- AspectJST
  - Research prototype based on AspectJ (v 1.06)

# Discussion

- A lot of work required to study impact of specific language features on crosscutting phenomenon
- Validated knowledge about quality of language features required, e.g.
  - What kind of type system demanded by developer?
  - What kind of type system represents „reasonable abstraction“?
  - Types a good abstraction for join point selection?
  - What kind of language feature responsible for what kinds of crosscutting?
- Increasing modulaty without touching current aspect-oriented systems?
- **Still a lot of work to do**