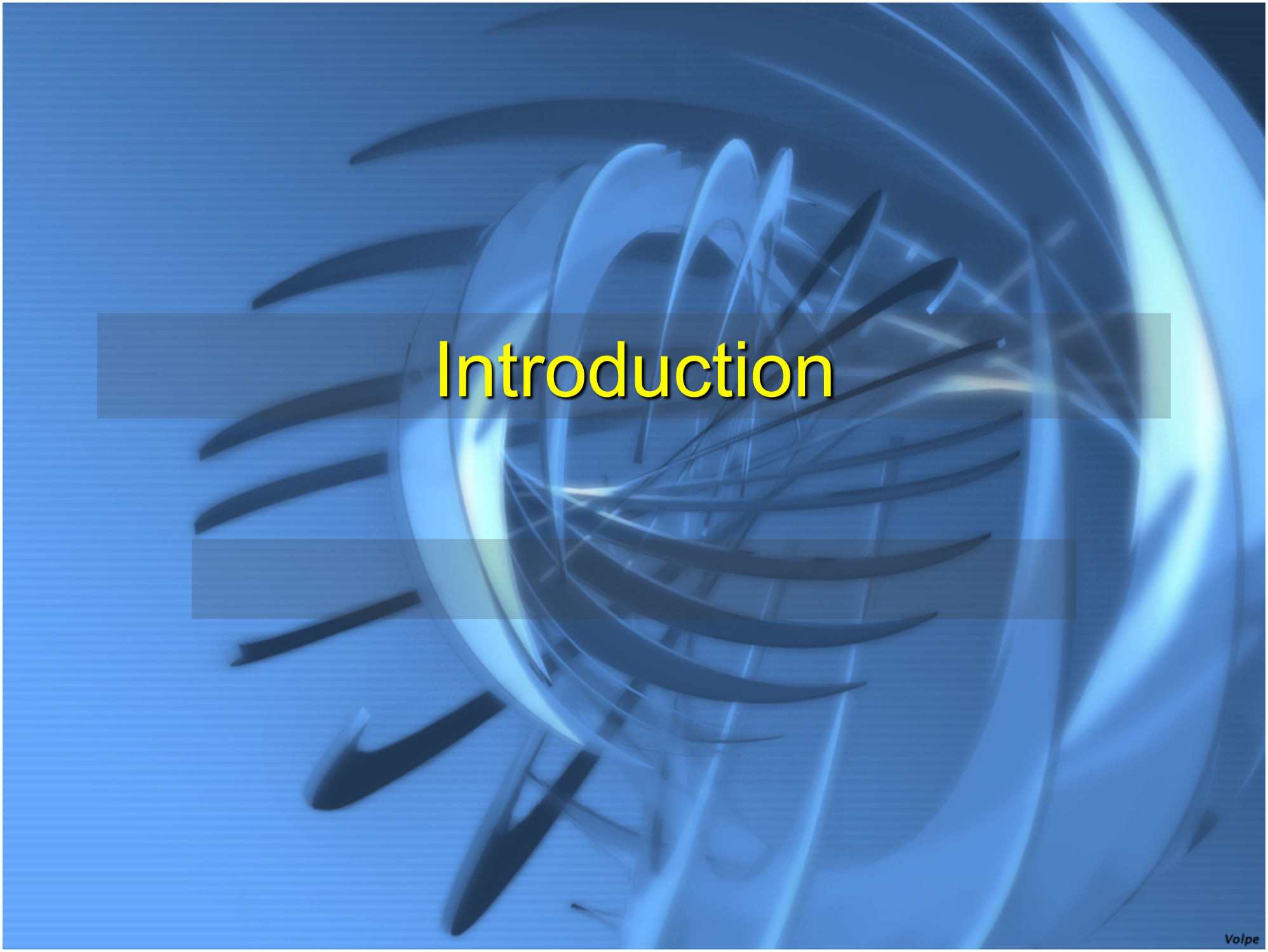
Aspect-Oriented Programming in Smalltalk

Kris Gybels
Robert Hirschfeld

Outline

- Introduction to AOP
- Using AOP for Smalltalk with ...
  - AspectS
  - Andrew
- Conclusion


Introduction
Volpe

Aspect-Oriented Programming

Improve quality of software
by introducing new modularization mechanisms
to deal with cross-cutting concerns.

Software Engineering

- Break problems into relatively independent smaller ones
- Implement and compose subproblems as modules
- What we want
  - Natural mapping between problems and modules
  - Localization and separation of concerns

Why OOP?

- OOP excellent for modeling entities of problem domain
- Problem domain objects map naturally to implementation objects
- Example: e-Library system
  - Books -> Book class
  - Library clients -> Client class
  - Printers -> Printer class

Why AOP?

- Example
  - "Concurrent processing" concern?
  - OO: protect all objects with semaphores ...
- OOP: problems when having to take into account special 'aspects' that cross-cut a system
- Requires code …
  - In different places
  - Not really related to modeling behavior from the problem domain

Problems with cross-cutting concerns

- Concern implementation intertwined with implementation of other concerns
- Concern not localized
- Hurts code …
  - Readability
  - Evolvability
  - Maintainability
  - …

AOP Goals

- "Modularize cross-cutting concerns"
- Create language/system to express
  - Points in a program where aspects influence the program (joinpoints)
  - How the aspect influences the program there

Types of AOP languages

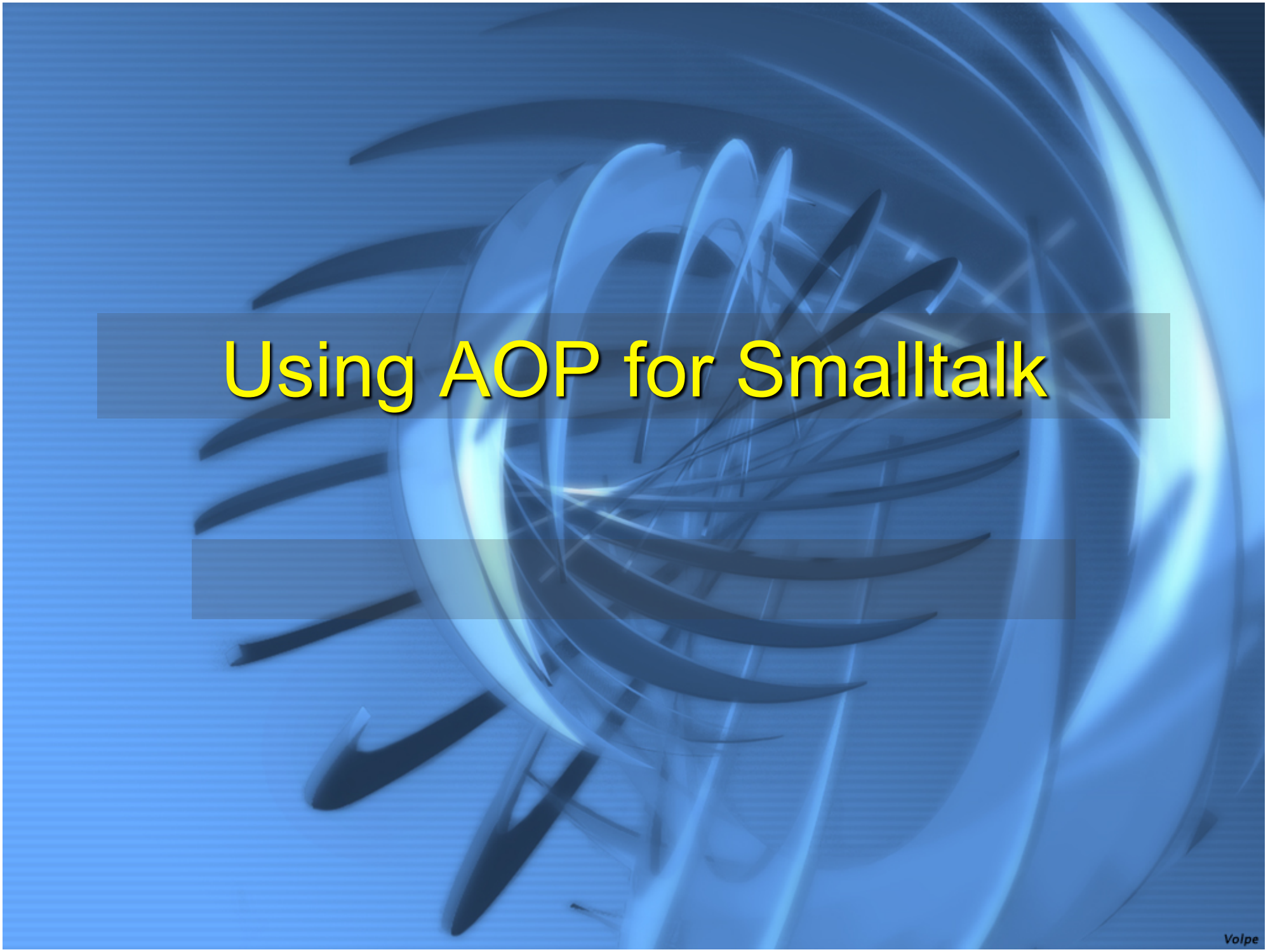
- High-level
  - Special aspect-specific languages to express aspects in (e.g. COOL)

    “Method setTitle: and setYear: on book objects cannot run concurrently”
    ->
    coordinator Book {
      mutexclusive {setTitle, setYear}
    }

Types of AOP languages

- General purpose
  - General purpose languages useable for different aspects (e.g. AspectJ, AspectS, Andrew, …)

    “When message setTitle: is sent to a book object first do a wait on the semaphore”

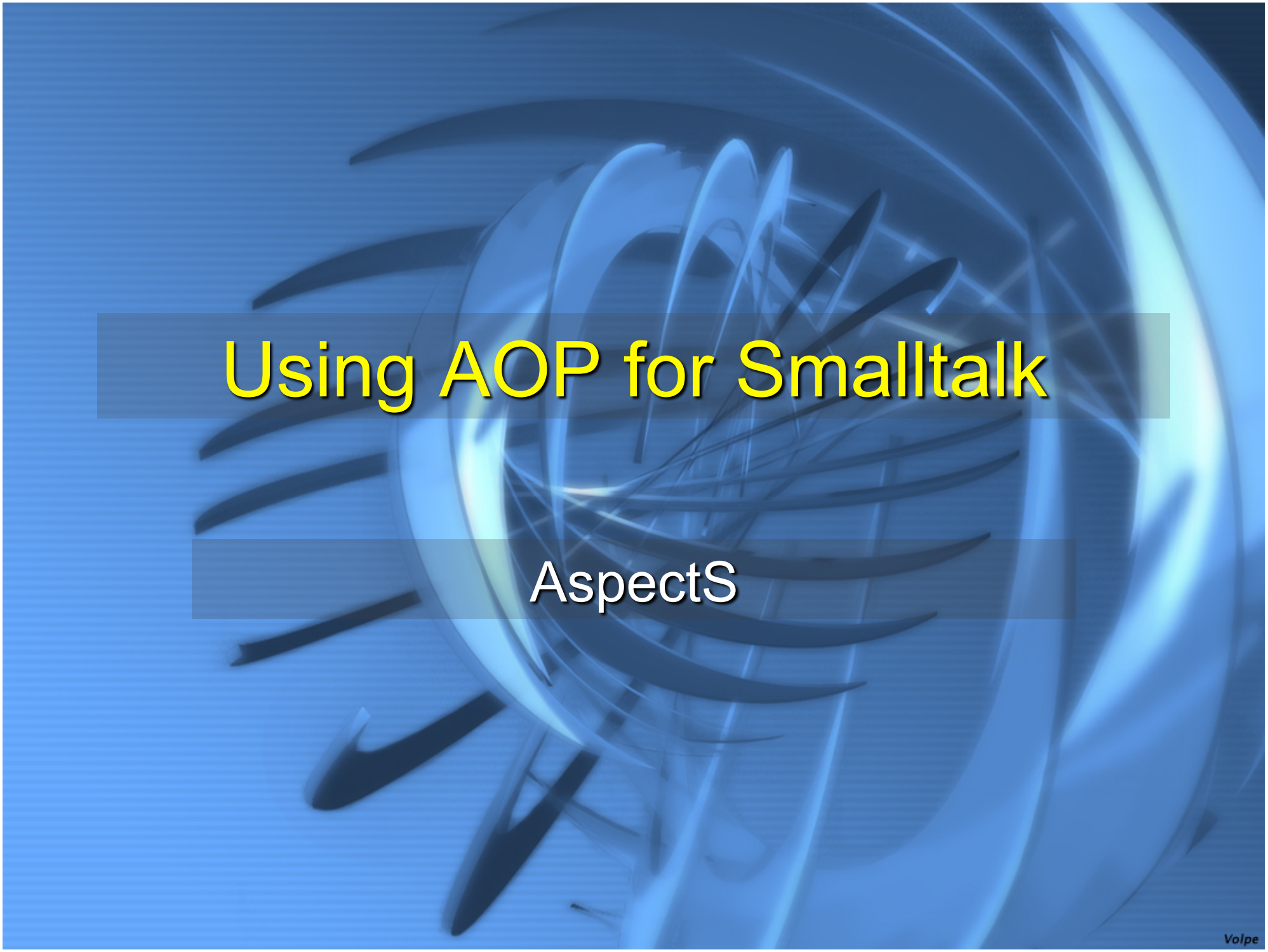

Using AOP for Smalltalk
Volpe

AspectS & Andrew

- Both based on the AspectJ general-purpose approach: *when ... before/after do ...*
  - Joinpoints: key events in the execution of an OO program
    - Message sends
    - Message receptions
    - State updates/accesses
  - Aspect's influence: advice
    - Smalltalk code

AspectS vs Andrew

- AspectS
  - Uses Smalltalk to express joinpoints
- Andrew
  - Uses special language based on Logic Meta Programming to express joinpoints


Using AOP for Smalltalk
AspectS
Volpe

Aspects in AspectS

- Implemented as subclasses of a specific class, can have:
  - regular variables
  - regular methods

for keeping and acting on state particular to the aspect (e.g. Semaphores)

  - special methods used to implement advice

Writing advice: example

```
AsMorphicMousingAspect>>adviceMouseEnter
    ^ AsBeforeAfterAdvice
        qualifier: (AsAdviceQualifier attributes: { #receiverClassSpecific. })
        pointcut: [ Morph allSubclasses
                select: [:each |
                        each includesSelector: #mouseEnter:]
                thenCollect: [:each | AsJoinPointDescriptor
                        targetClass: each
                        targetSelector: #mouseEnter:]]
        beforeBlock: [:receiver :arguments :aspect :client |
                self
                        showHeader: '>>> MouseENTER >>>'
                        receiver: receiver
                        event: arguments first]
```

Pointcuts

- Blocks which compute a collection of JoinpointDescriptors
- JoinpointDescriptors indicate method execution joinpoints the advice influences: class + selector
- The full Smalltalk meta system can be used!

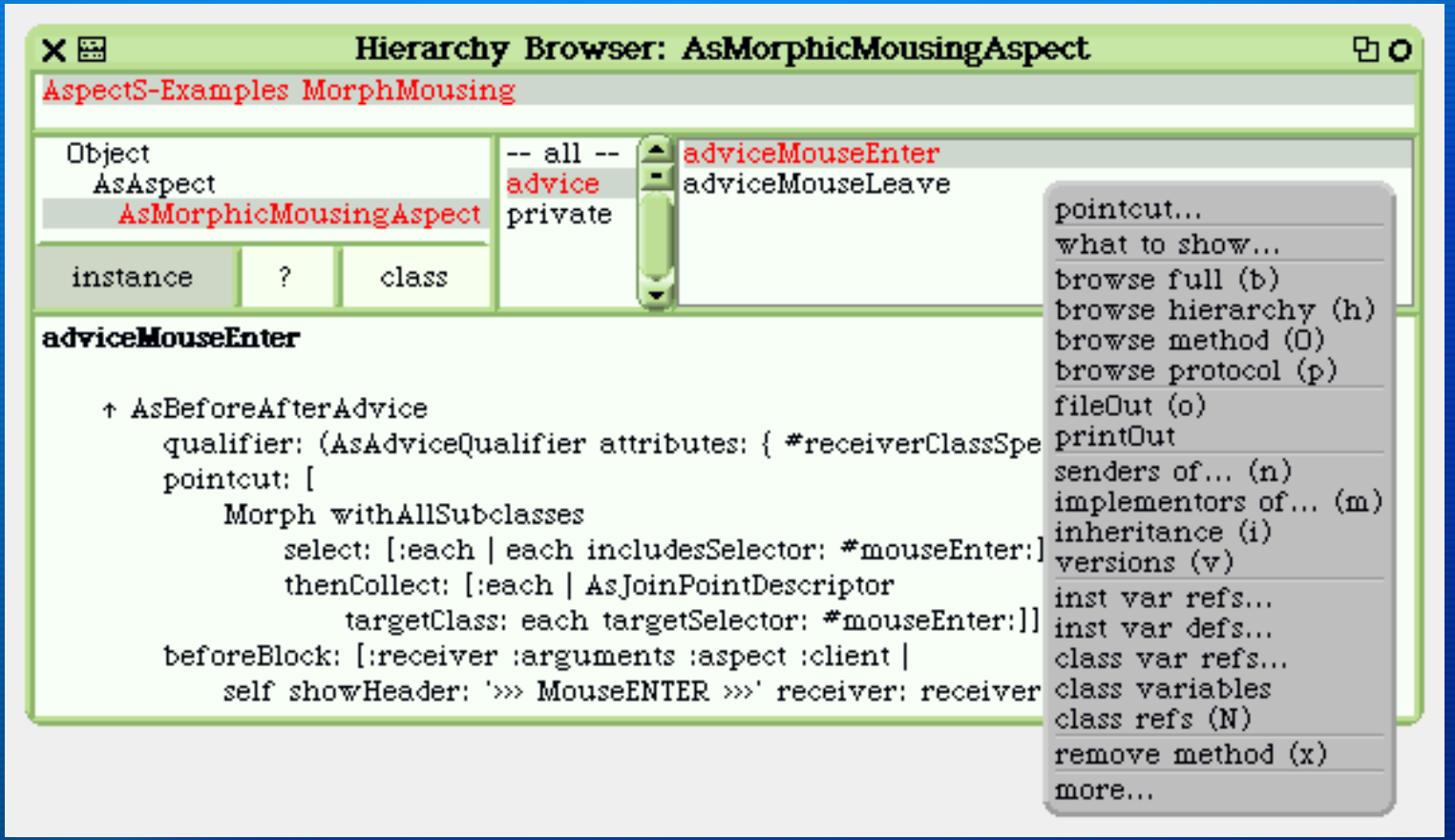
Advice

- A Smalltalk block executed before or after every joinpoint matching the pointcut
- The block is passed context data of the joinpoint
  - actual object that received the message
  - arguments sent with the message

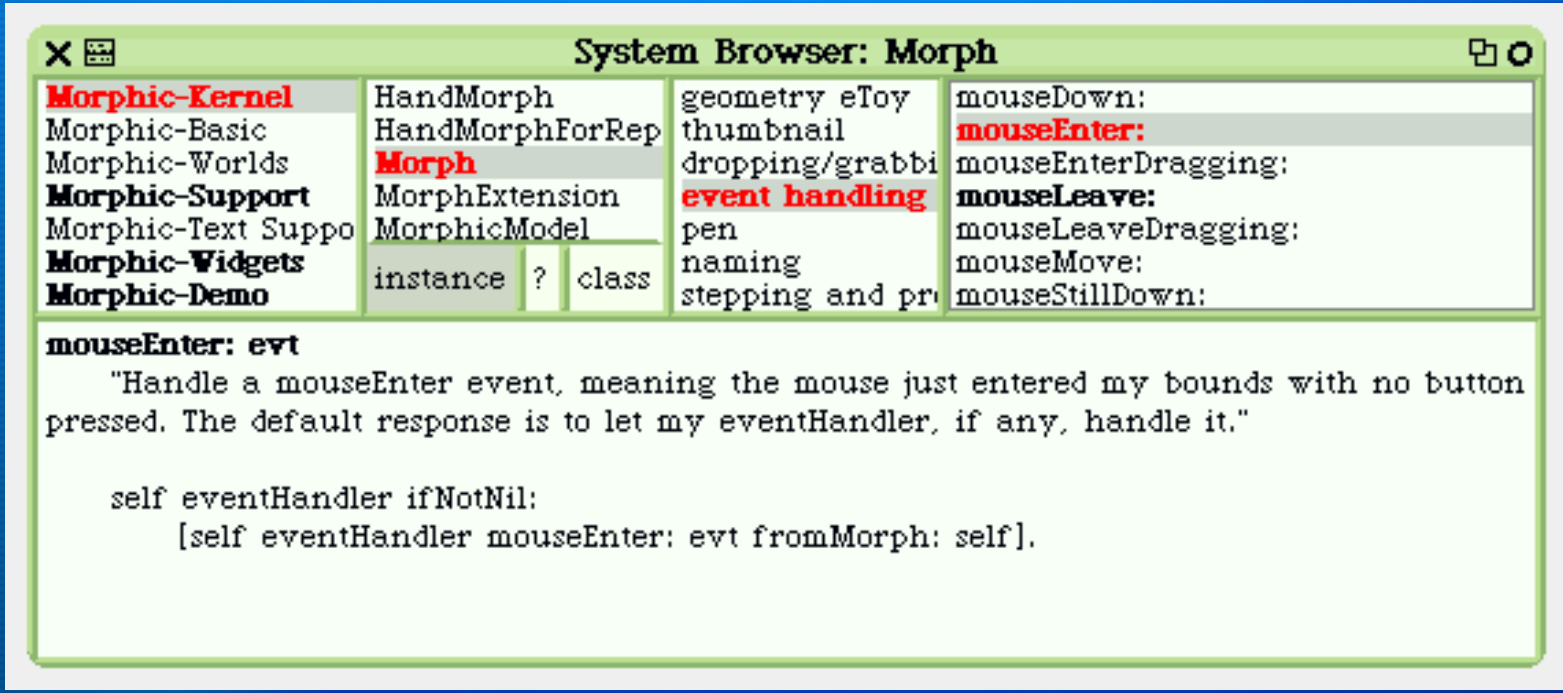
Tools

- No special support for defining and editing aspects, just use system browsers
- Browser extensions for showing the impact of aspects

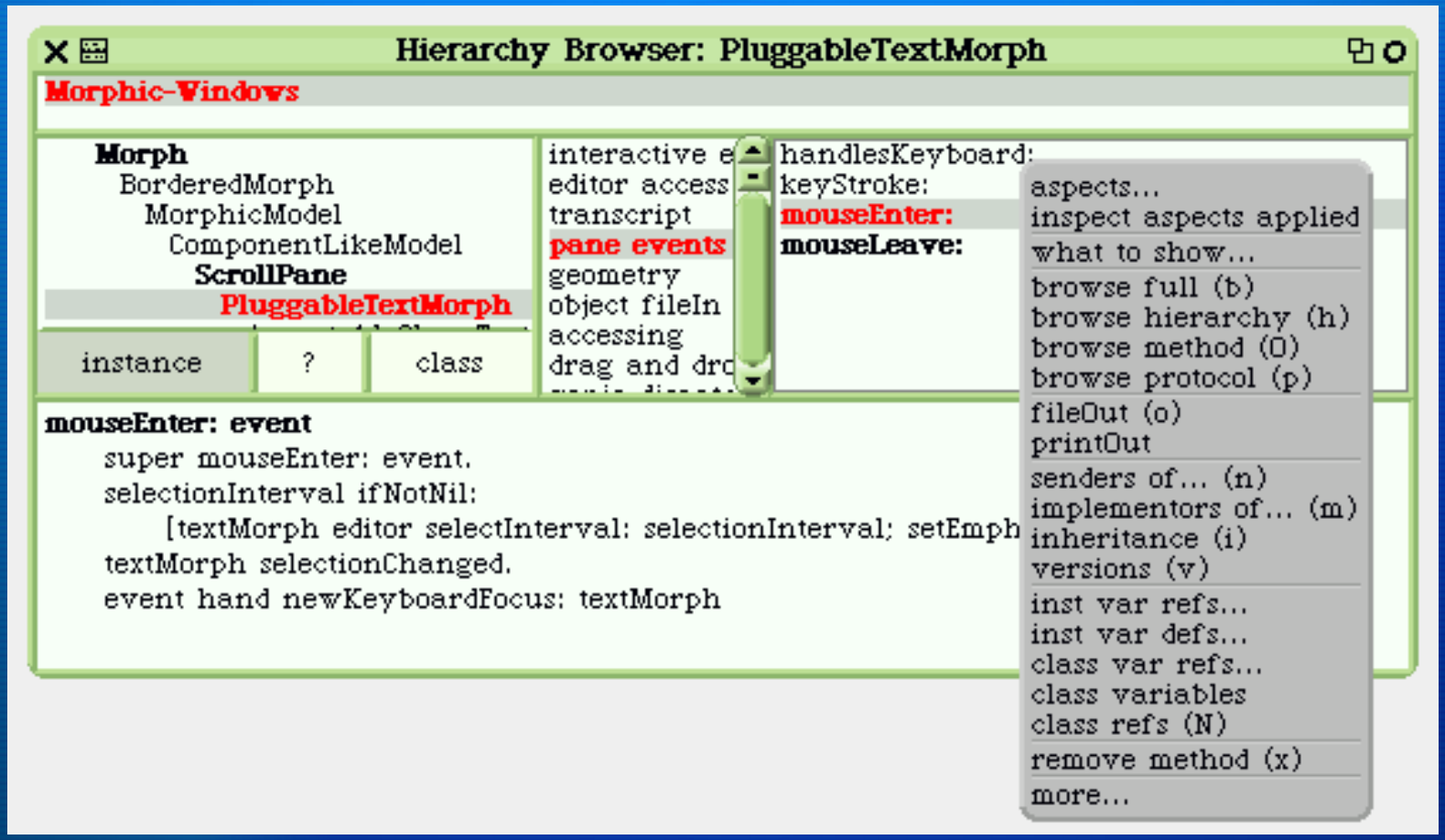
Tools

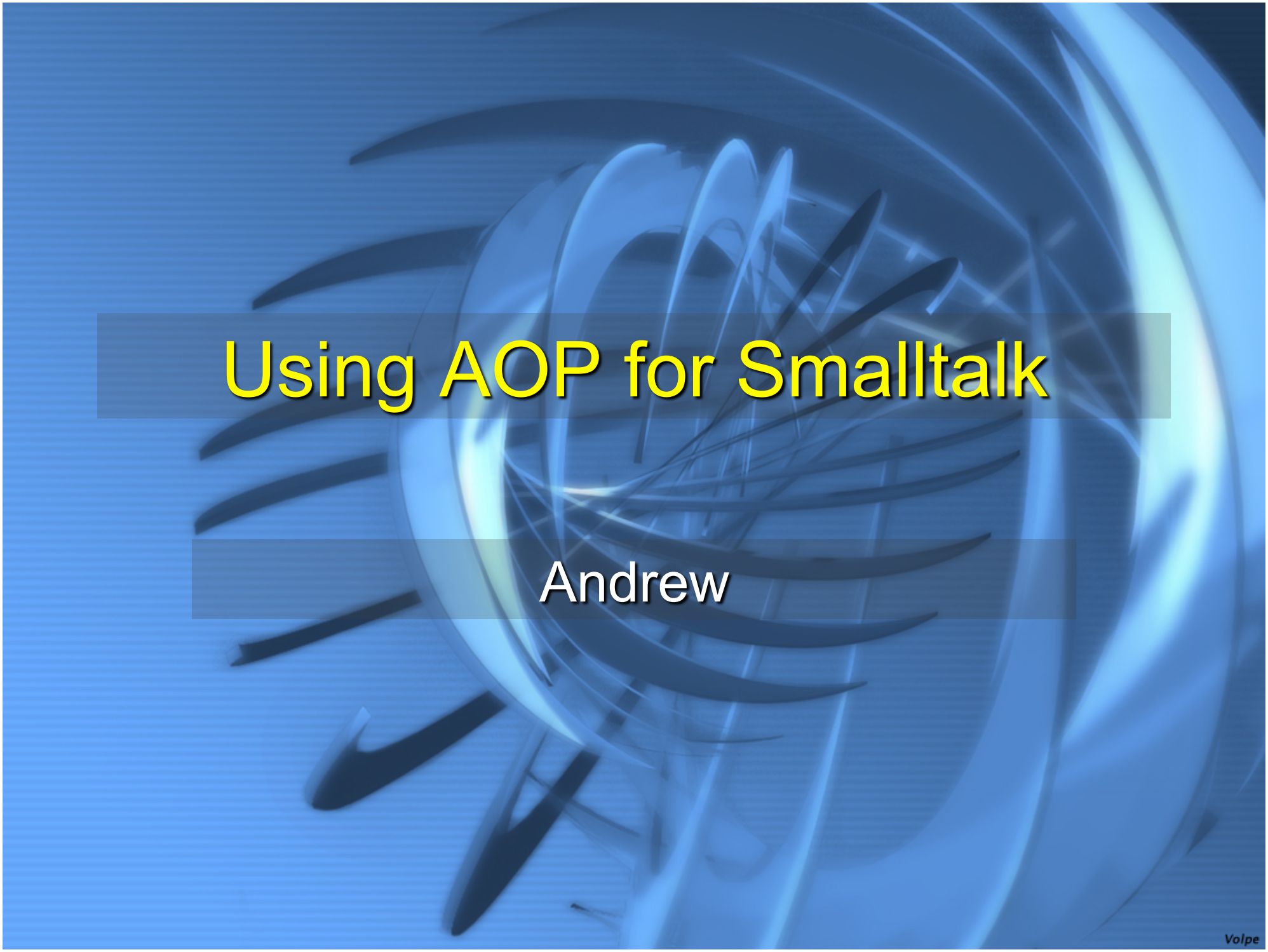


Tools



Tools




Using AOP for Smalltalk
Andrew
Volpe

Aspects in Andrew

- Similar to AspectS, implemented as subclasses of a specific class
  - regular variables/methods
  - advices
  - logic predicates
- Uses logic meta programming to express pointcuts

Why logic pointcuts?

- Cross-cutting: compute or describe?
  - Computing allows flexible cross-cuts
  - Cross-cutting is best kept descriptive
- Logic programming combines the two properties

Logic Meta Programming

- Meta programming for Smalltalk using logic language: SOUL
- Smalltalk programs represented as logic facts
  - class(?c)
  - methodInClass(?c, ?selector, ?m)
- LiCoR extensive library of logic rules to reason about Smalltalk programs

Logic cross-cutting

- Adds predicates to reify joinpoints:
  - reception(?jp, ?selector, ?args)
  - send(?jp, ?selector, ?args)
  - get(?jp, ?instVar, ?value)
  - set(?jp, ?instVar newV, ?, ?oldV)

Writing advice ...

```
before ?jp matching {
    reception(?jp, #name)
} do
    Transcript show: 'name accessed
    through accessor'
```

Code patterns & cross-cutting

- LMP successfully used to detect patterns in code
  - Smalltalk programming idioms
  - Design patterns
  - ...
- Can be used to ...
  - Clearly capture the pattern underlying a cross-cut
  - Specialize joinpoints (“Open weaver”)

Accessor example

- Use LMP to find accessors

```
isAccessor(?class, ?selector, ?varName) if
        class(?class),
        methodNamed(?class, ?selector, ?method),
        statements(?method, ?statements),
        equals(?statements, <return(variable(?varName))>)
```

Observer

```
after ?jp matching {
    reception(?jp, ?selector),
    withinClass(?jp, [Person]),
    notify([Person], ?selector, ?property),
    viewInterestedIn(?property)
} do

    views do: [ :view | view changed: ?property asSymbol ]
```

Observer

notify([Person], [#birthDate:], [#birthDate]).
notify([Person], [#birthDate:], [#age]).
notify([Person], [#name:], [#name])

Observer

viewInterestedIn([#age]).
viewInterestedIn([#name])

Using AOP for Smalltalk AspectS vs Andrew

- Pointcuts more familiar for a Smalltalk programmer
- Pointcuts require one to learn new language

Implementing AOP for Smalltalk

Final notes & Conclusions

Conclusions

- AspectS vs Andrew
  - AspectS's pointcuts more familiar to Smalltalk programmer
  - Andrew pointcut language provides enhanced readability + extensive code reasoning library
- AspectS & Andrew vs AspectJ
  - Use of full MOP allows one to find more code patterns to cross-cut

Links

- prog.vub.ac.be/~kgybels/andrew/
- www.prakinf.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS
- VisualWorks 7 distro
- www.aosd.net

- kris.gybels@vub.ac.be
- hirschfeld@acm.org