QSOUL/Aop

Aspect Oriented Software Development using Logic Meta Programming

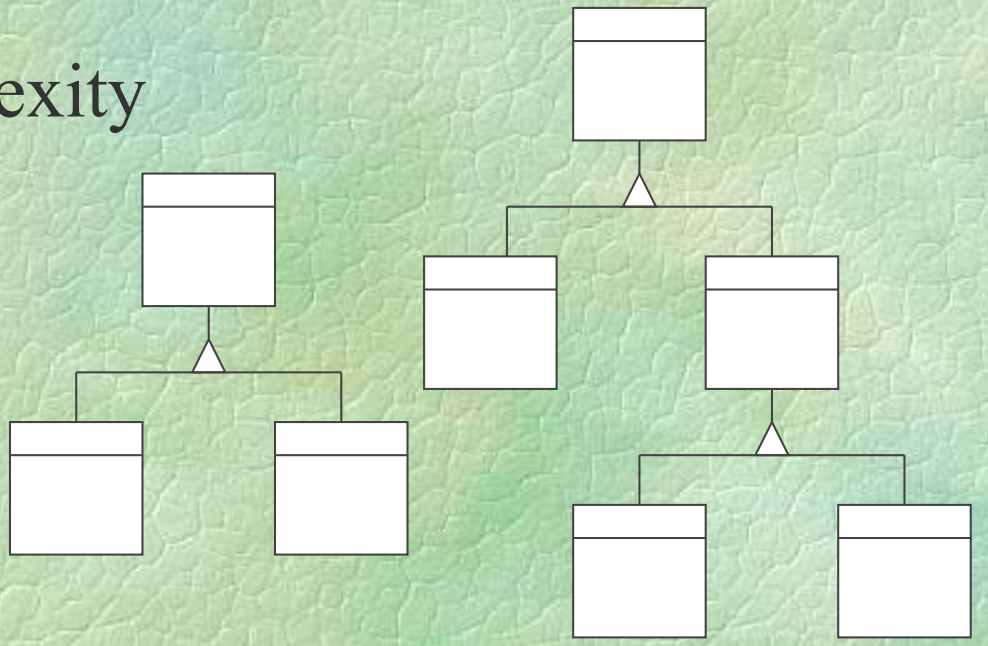
Johan Brichau,
Programming Technology Lab,
Vrije Universiteit Brussel,
Brussel, Belgium

Overview

- Aspect Oriented Software Development
- Goals of QSOUL/Aop
- Logic Meta Programming in QSOUL
- Aspect Oriented Logic Meta Programming
- QSOUL/Aop tool
- Demo time

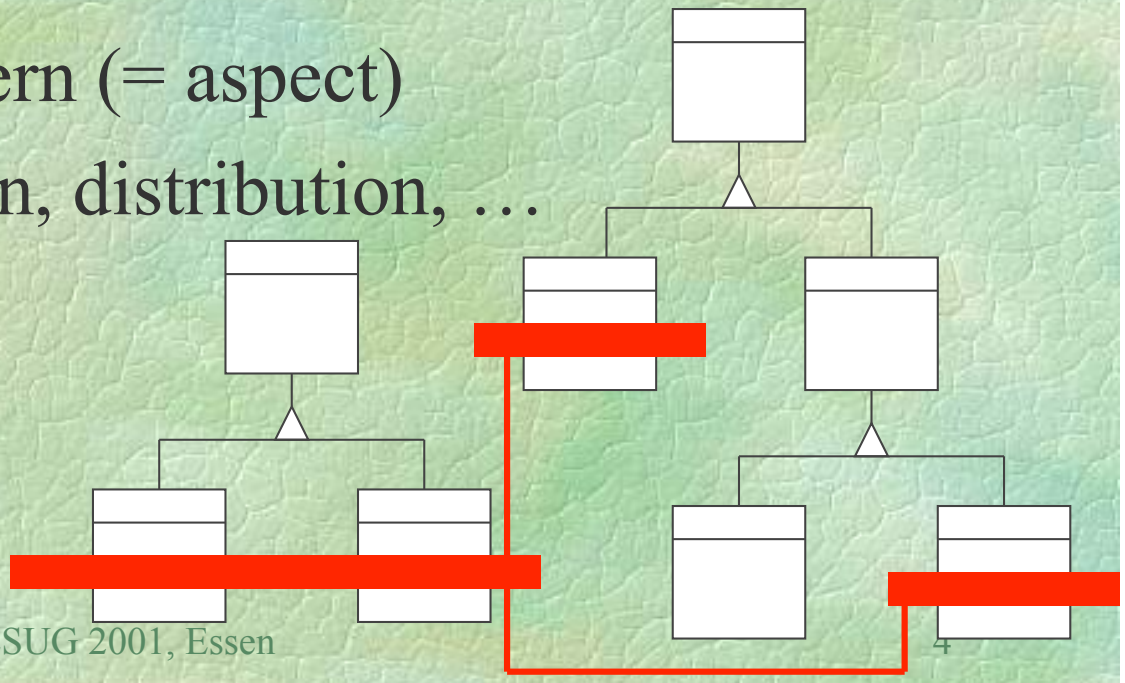
Software Development

- Software development today happens through hiërarchical decomposition in ‘generalised procedures’
  - Break program complexity
  - Modularize concerns



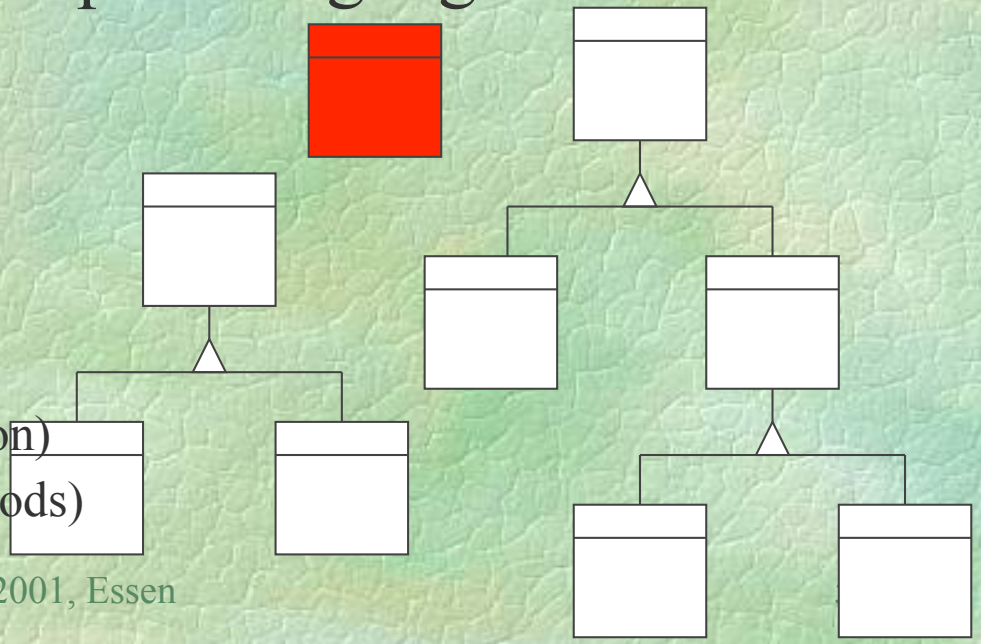
Aspect Oriented Software Development

- But some concerns cannot be modularized and occur in every module of the decomposition
  - Cross-cutting concern (= aspect)
  - E.g. synchronization, distribution, …



Aspect Oriented Software Development

- AOSD tries to modularize these aspects.
- Aspects are combined with component program using a weaver
- Aspects specified in an aspect-language
  - Describing
    - Cross-cutting
    - Functionality
  - Examples:
    - COOL (synchronisation)
    - RG (loop fusion optimisation)
    - AspectJ (advices over methods)



Overview

- Aspect Oriented Software Development
- Goals of QSOUL/Aop
- Logic Meta Programming in QSOUL
- Aspect Oriented Logic Meta Programming
- QSOUL/Aop tool
- Demo time

Goals of QSOUL/Aop (1)

- Declarative Aspect Language
  - Aspects have a declarative nature
  - Examples
    - Synchronisation: declare what methods are synchronized
    - Error handling: declare what errors should be catched where and what should be executed.
    - Wrap methods: declare what methods should be wrapped and what should be executed.

Goals of QSOUL/Aop (2)

- User-defined aspect-languages
  - An open framework that allows definition of user-defined aspect-languages
  - Express one aspect-language in another aspect-language
  - Examples:
    - A '*wrap methods*' aspect-language could be used to introduce synchronisation or error handling, but is less suited than a specialized '*synchronisation'* or '*error handling*' aspect-language.

Goals of QSOUL/Aop (3)

- Combination and composition of several aspects
  - Implemented in one aspect-language
  - Implemented in different aspect-languages
  - Detect and resolve conflicts between aspects
- Example conflict:
  - If a scheduler-aspect does not know about a synchronisation aspect, deadlocks can occur!

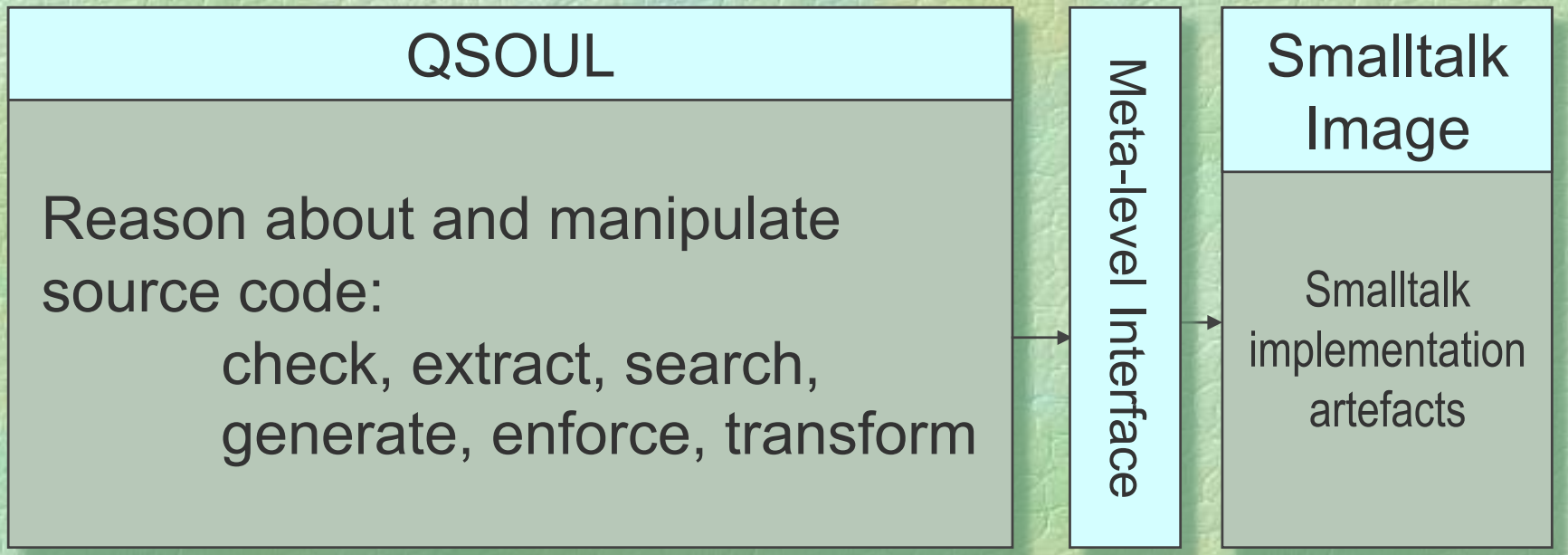
Overview

- Aspect Oriented Software Development
- Goals of QSOUL/Aop
- Logic Meta Programming in QSOUL
- Aspect Oriented Logic Meta Programming
- QSOUL/Aop tool
- Demo time

Logic Meta Programming

- Combines a logic meta language with a standard object-oriented base language
  - base-level programs are expressed as *terms, facts and rules* at the meta level
  - meta-level programs can *manipulate* and *reason about* the base-level programs

QSOUL: setup


QSOUL
Reason about and manipulate source code:
check, extract, search, generate, enforce, transform
Meta-level Interface
Smalltalk Image
Smalltalk implementation artefacts

QSOUL language

- Prolog and…
  - Smalltalk terms

```
allClasses([Smalltalk allClasses])
```

aCollection object

  - Smalltalk clauses

```
write(?text) if
        [Transcript show: (?text asString). true].
```

true

  - Quasi Quoted Code

```
methodCode(Foo,bar, { ^ nil } )
```

LMP achievements

- Emerging technique to build state-of-the art *software development tools*
- In particular, *tools* to support *co-evolution* in all facets and phases of the software life-cycle
  - information in implementation and earlier life-cycle phases may evolve independently
  - need to keep information in these phases synchronised
- To support advanced *software engineering techniques*

LMP Achievements

Declarative Reasoning about object-oriented base programs supporting the **Co-Evolution** of design and implementation

- ***Extract*** *design information from the implementation.*
- ***Verify*** *the implementation with the corresponding design.*
- ***Generate*** *the implementation from the design*

Theo D'Hondt, Kris De Volder, Kim Mens & Roel Wuyts, ***Co-evolution of Object-Oriented Software Design and Implementation.*** *In Proceedings of SACT 2000. Kluwer Academic Publishers, 2000*

Overview

- Aspect Oriented Software Development
- Goals of QSOUL/Aop
- Logic Meta Programming in QSOUL
- Aspect Oriented Logic Meta Programming
- QSOUL/Aop tool
- Demo time

Aspect Oriented Logic Meta Programming (AOLMP)

- Aspect language **embedded in logic language**.
- An aspect language consists of two parts
  - the aspect-code
  - how the aspect crosscuts the base program
- **Inference engine** gathers the logic declarations of all aspects and **weaves** them in the base program.
- Using logic rules we can build a **domain-specific aspectlanguage** embedded in the logic language.
- E.g. TyRuBa and QSOUL/Aop

AOLMP using QSOUL/Aop

- **Composition-mechanism** to support composition of aspects
- **Integrated** Smalltalk-weaver
- Exploit symbiosis
  - Use reasoning about base program to specify **user-defined crosscuts.** (E.g.: all places in the program where a certain variable is initialized)
  - Multi-paradigm programming (logic & procedural programming) eases complexity of rules that implement a user-defined aspect-language

Overview

- Aspect Oriented Software Development
- Goals of QSOUL/Aop
- Logic Meta Programming in QSOUL
- Aspect Oriented Logic Meta Programming
- QSOUL/Aop tool
- Demo time

QSOUL/Aop

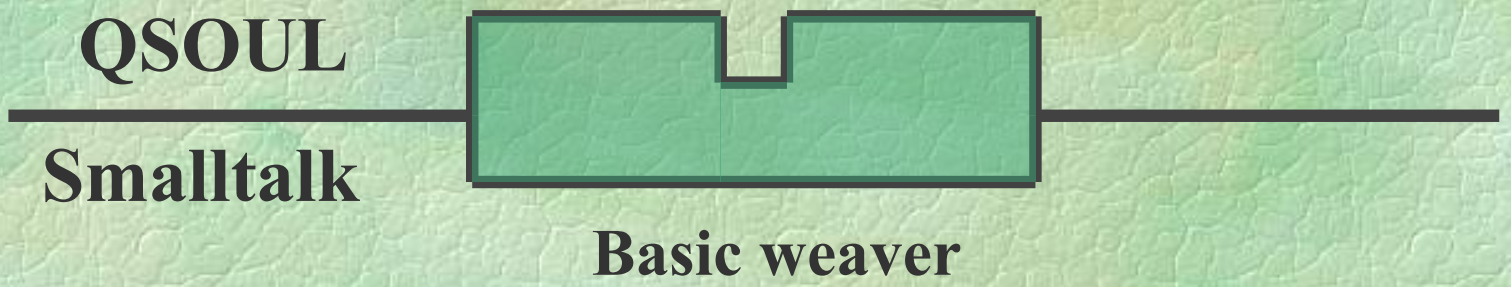
- Consider a simple aspect-language for error-handling:
  - onError(?class,?selector,?error,?error-handling-block)
- Consider two simple error-handling aspects:
  - onError([Array],[#at:put:],[OutOfBoundsError], {[:e | … handle exception e…]})
  - onError(?class,[#at:put:],[OutOfBoundsError],{…}) if subclass([SequenceableCollection],?class)

QSOUL/Aop

- Consider a wrap-around aspect-language:
  - around(?class,?selector,?code)
- Define the meaning of the error-handling aspect-language in terms of the wrap-around aspect-language
  - around(?class,?selector,
    {[original()] on: ?error do: ?errorcodeBlock}) if
    onError(?class,?selector,?error,?errorcodeBlock).
- Define the wrap-around aspect-language in terms of another aspect-language…

QSOUL/Aop

- Basic weaver
  - Hard-coded in Smalltalk
  - Invisible overriding of methods
    - Only supports method-crosscuts
  - Share state in group of overriden methods



Basic weaver

- Basic aspect language: **crosscut declarations**
  - weave(method(?class,?selector),{<aspect-code>})
    Override the method ?selector in ?class with <aspect-code>

N>0

  - scopeOf(?instVarList,{ <aspect-scope-code> })
    Share instance variables in all executions of aspect-code where <aspect-scope-code> results in same value. Create new instance variables where <aspect-scope-code> results in a new, unique value.

N=0,1

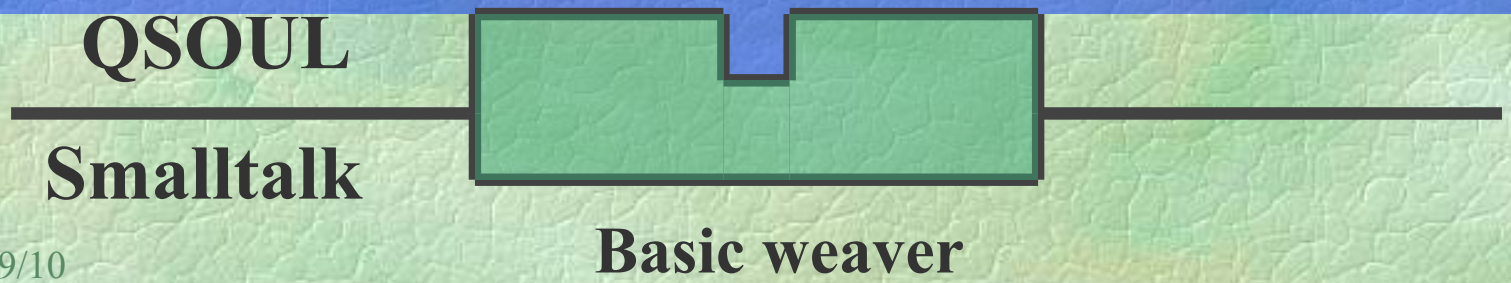
Basic weaver

- Basic aspect language: **aspect-code** weave(method(?class,?selector),{<aspect-code>})
  - Smalltalk code and…
  - thisObject
    Access to the current receiver in aspect-code
  - original()
    Execute the original method (with the original arguments)
  - thisAspect
    not yet…

A basic-weaver aspect: Logging

Write the size of the collection to the Transcript every time after an element is added to an Array or an OrderedCollection

```
weave(?pc, { |tempResult| tempResult := original().
                Transcript write: thisObject size asString.
                ^ tempResult } ) if
        location(?pc).
location(method([Array],at:put:) ).
location(method([OrderedCollection],add:))
```



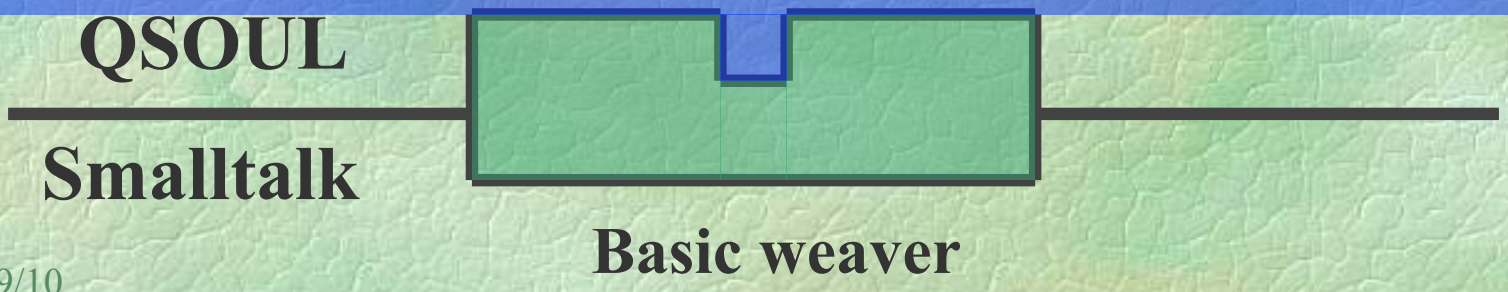
Building your own aspect language

- An AspectWeaverMixin…
  - …defines a new aspect-language in terms of another aspect-language.
  - …defines a transformation of a higher-level aspect language to a lower-level aspect-language
  - …can be mixed with other aspectweavermixins and the basic weaver to form a complete aspectweaver

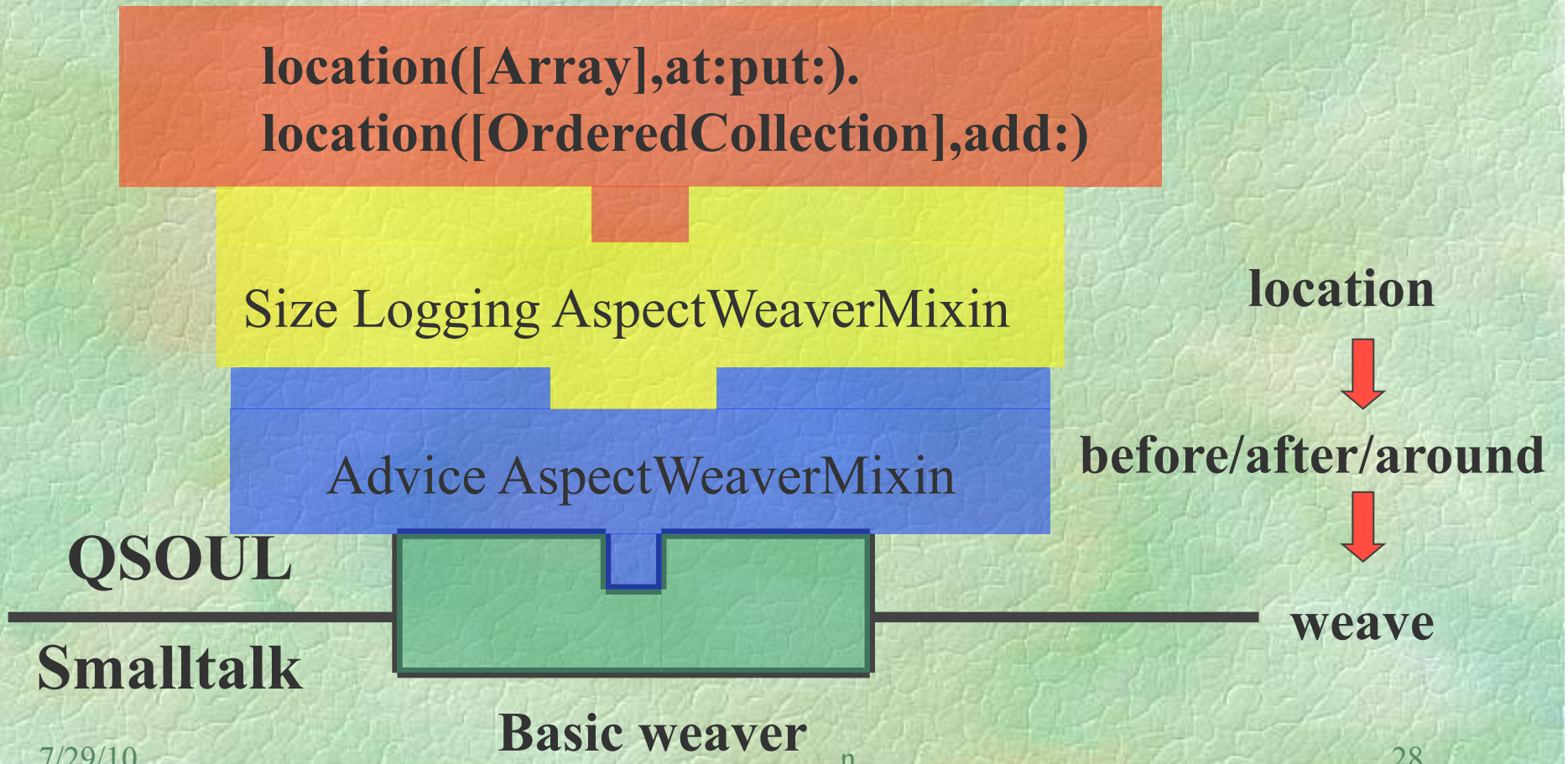
Building your own aspect language

```
after(execution(?class,?selector),{Transcript write: thisObject size asString}) if
        location(?class,?selector).
location([Array],at:put:).
location([OrderedCollection],add:)
```

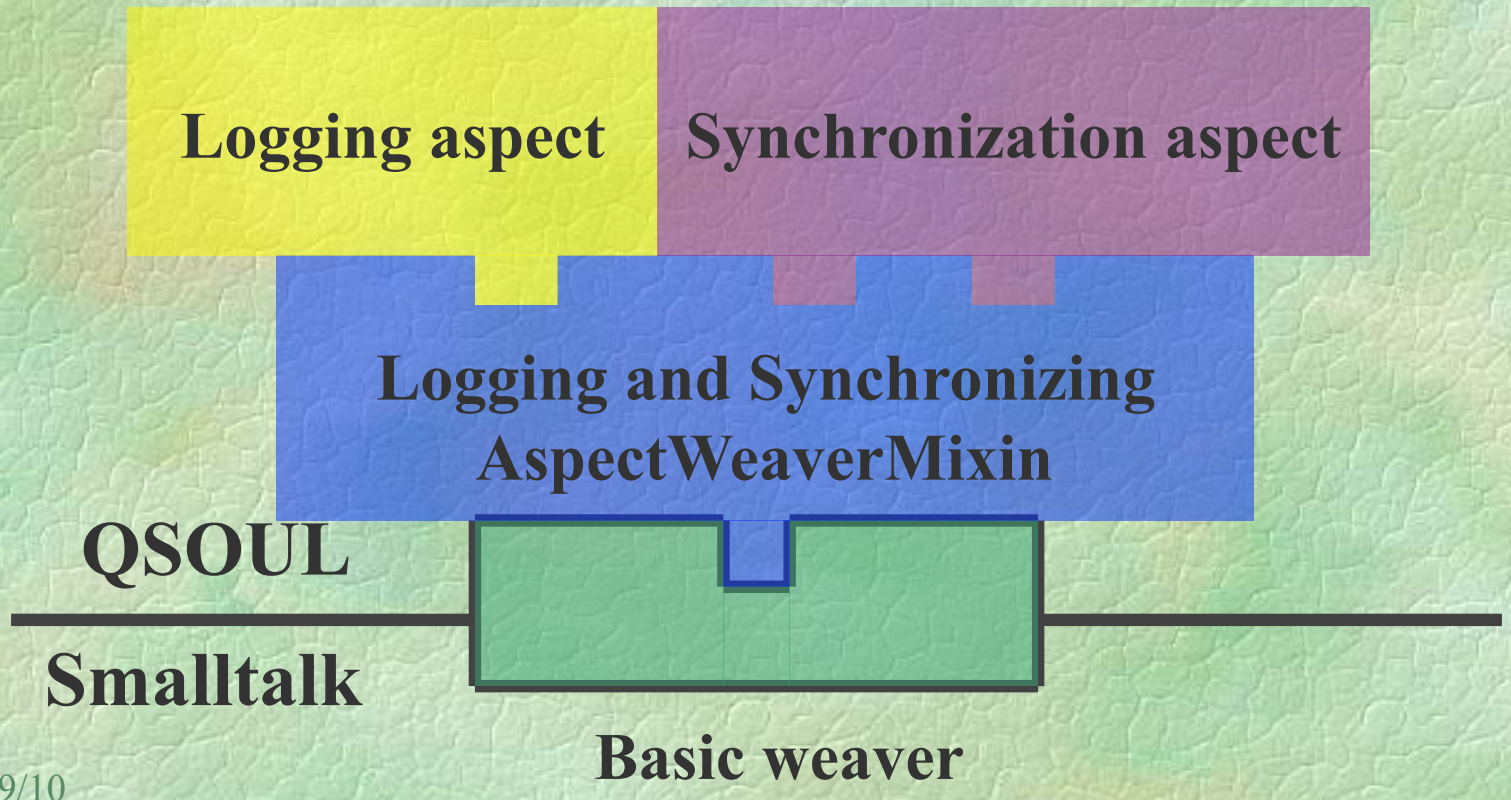
```
weave(method(?class,?sel), { |tempResult| tempResult := original().
        ?code. ^ tempResult } ) if
    after(execution(?class,?sel),?code).
```



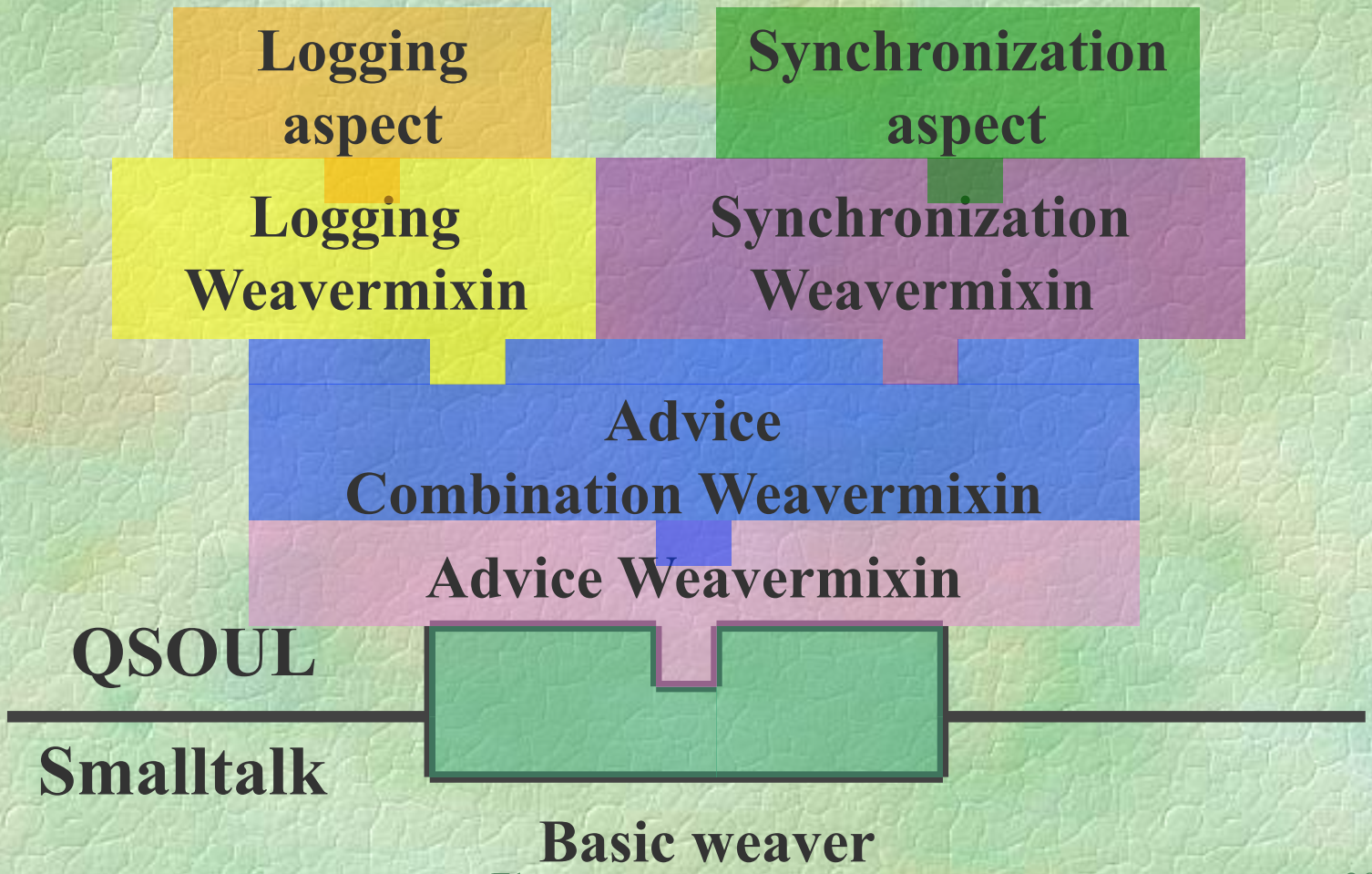
Building your own aspect language



Combination of aspect-languages



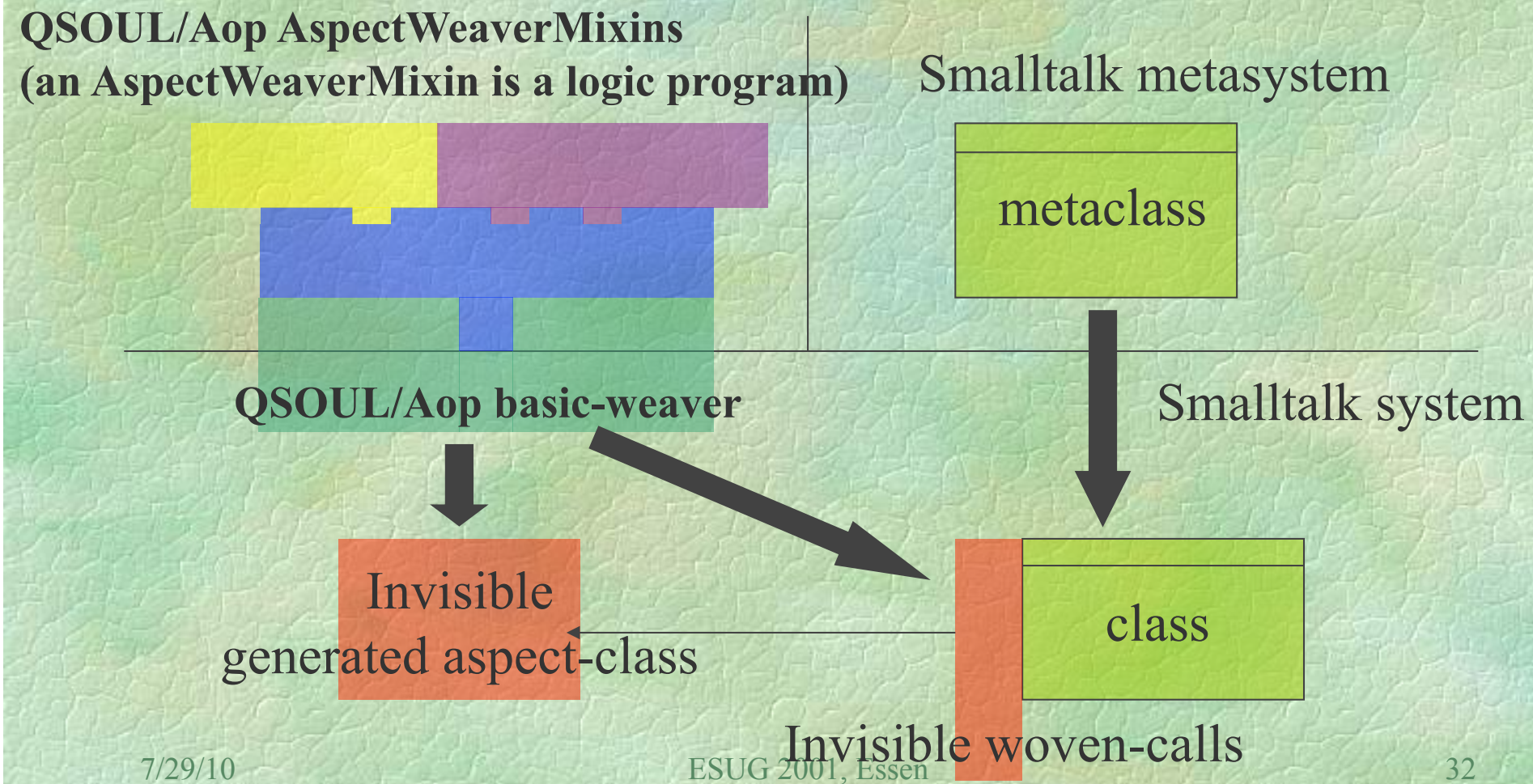
Combination of aspect-languages



QSOUL/Aop: Open crosscut language

- QSOUL's reasoning about Smalltalk basecode allows detection of patterns.
  - Extract implicit call-structure
  - Extract design patterns
  - Etc…
- This information can be used to implement user-defined crosscuts
  - Method that initializes instance variables
  - Methods that send messages to a Stack instance
  - Etc…

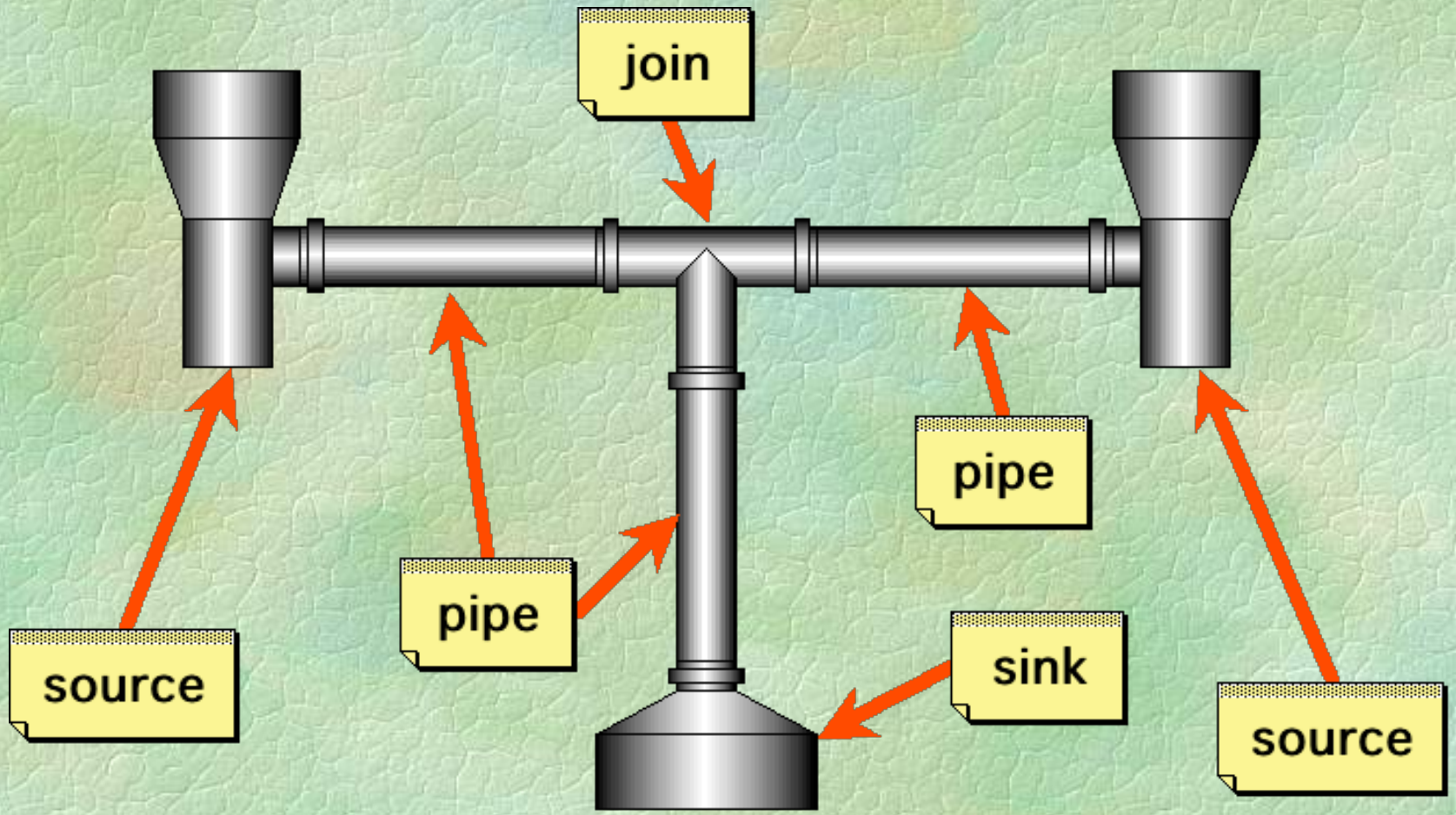
Integration in Smalltalk



QSOUL/Aop: Future Work

- thisAspect
- Scope per aspect-instance variable
- Aspect methods
- Extend basic weaver to weave on other language elements
- Technical improvements
  - Use method wrappers instead of hidden classes
  - Check for uses of self in aspect-code
  - …

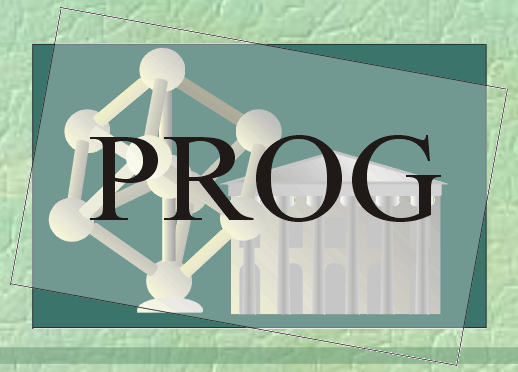
Demonstration: The Conduits Framework



Aspects in Conduits-Framework

- User Interface update
  - After each fill, update view
- Synchronization and message order
  - fill and drain: in alternate order + blocking
- Overflow logging
  - Setting of content everywhere should produce same message (throws error?)
- Etc…

Links



Declarative (Logic) Meta Programming:

http://prog.vub.ac.be/poolresearch/dmp/

QSOUL2:

http://prog.vub.ac.be/poolresearch/qsoul/qsoul2.html

QSOUL/Aop:

http://prog.vub.ac.be/poolresearch/aop/qsoulaop.html

johan.brichau@vub.ac.be

QSOUL *LMP-tool*

- Strong symbiosis between logic language and Smalltalk
  - Logic language acts on current Smalltalk image
  - Smalltalk objects are constants in the logic language
  - Logic clauses can execute parameterised Smalltalk expressions
- Code generation through manipulation of quasi-quoted codestrings

Aspect Oriented Software Development

- Subject Oriented Programming and Multidimensional Separation of Concerns
  - Different views on the program's decomposition, each addressing a concern
  - Compose the different views with composition rules
- Composition Filters
  - Place wrappers around encapsulations, each addressing a cross-cutting concern

Aspect Oriented Software Development

- Design Patterns
  - Use exisiting ‘generalised procedure’ techniques to separate cross-cutting concern (e.g. Visitor pattern)
- Aspect Oriented Programming
  - Encapsulate aspects
  - Aspect-weaver composes aspects with other encapsulations guided by a pointcut-language

AOLMP using TyRuBa