LogicAJ – The Power of Uniformly Generic Aspects

Tobias Rho
Günter Kniesel

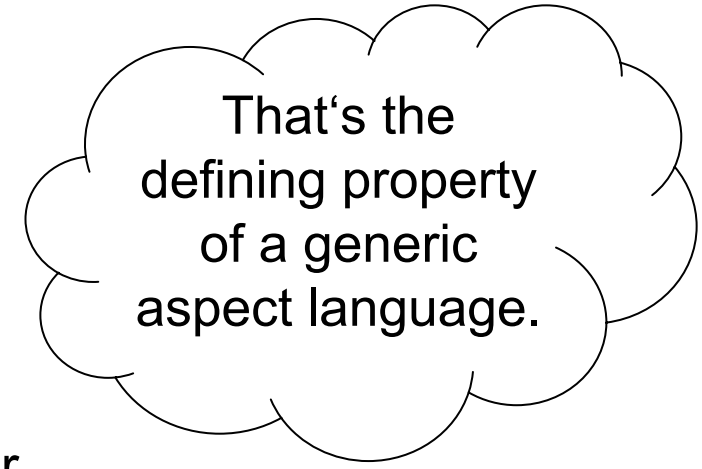
ROOTS group
Computer Science Department III
University of Bonn
Germany

{rho, gk}@cs.uni-bonn.de

universitätbonn

Adressed Problem: Context dependent effects

- Homogeneous aspect effects
  - same advice applies at all join points
  - too rigid in many cases
- Aim: Heterogeneous aspect effects
  - Let aspect effects vary depending on the context of a join point
- Dynamic binding for objects
  - **receiver type** determines behaviour
- Weave-time binding for aspects
  - **join point context** determines behaviour



universitätbonn

Adressed Problem: Context dependent effects

- Scenario:
  - New subclass should be used consistently instead of its superclass
- Language feature (ObjectiveC): Class posing
  - Replace superclass constructor calls by calls of subclass constructors
    - Many constructors
    - Different signatures
    - Constructor names depend on the replaced classes
- Can we achieve this with aspects?

universitätbonn

Class Posing via Aspect Code Bloat

- Example: Replace superclass S by new class C
  - Superclass constructors: S(String), S(String,int), ...
  - Subclass constructors: C(String), C(String,int), ...

```
aspect AJ_Replace_S_by_C {

 S around(String arg1) :
    call(S.new(..)) && args(arg1)
    {
     return new C(arg1);
    }

 S around(String arg1, int arg2) :
    call(S.new(..))&& args(arg1,arg2)
    {
     return new C(arg1,arg2);
    }

  // Continue like this ...
}
```

- Redundancy
  - New advice for every constructor
  - Similar structure
- No Reusability
  - Need to write another aspect to replace another pair of classes
  - S → C1, S → C2, ...
  - S1 → C, S1 → C1, ...

universitätbonn

Class Posing via Reflection

- Example: Replace superclass S by new class C
  - Superclass constructors: S(String), S(String,int), ...
  - Subclass constructors: C(String), C(String,int), ...

```
aspect AJClassPosingWithReflection {
 abstract String getOldClassName();
 abstract String getNewClassName();
 Object around() : call(*.new(..)) {
    Class class =  thisJoinpoint.getSignature().getDeclaringType();
    Class newClass;
    if (!class.getName().equals(getOldClassName()){
        return proceed();
    }
    try {
       newClass = Class.forName(getNewClassName());
    } catch (Exception ex) {
       return proceed();
    }
    Object[] args = thisJoinPoint.getArgs();
    try {
       Class[] types =
              ((CodeSignature)thisJoinPoint().getSignature()).
               getParamtypes();
       Constructor constr = resolveConstructor(newClass, types);
       return constr.newInstance(args);
    } catch (Exception ex) {
       throw new RuntimeException("Instantiation failed");
    }
 }
}
```

- Verbose
  - 30 line helper method not shown here
- Obscure
  - What's going on here?
- No static safety
  - Run time exceptions if class / constructor unavailable

universitätbonn

Limitations of Wildcards

```
aspect EclipsePackageAccessContract {
  declare warning :
    call(* **.internal.**.*.*(..))      // an internal package
    && !(                               // may only be accessed
         within(**) ||                  // from its superpackage
         within(**.internal.**) )       // or another internal ...
    :
    "The call violates the Eclipse package access contract.";
}
```

- Limitation of wildcard based matching
  - Cannot express which wildcards must match the same value
  - Matches too much

First step to Genericity: Metavariables

```
aspect EclipsePackageAccessContract {
 declare warning :
    call(* ?apiPackage.internal.**.*.*(..))
    && !(
         within(?apiPackage) ||
         within(?apiPackage.internal.**)
    )
    :
    "The call violates the Eclipse package access contract.";
}
```

- Introduce *Metavariables* for entities of base programs
  - Types, fields, methods, … → **?var**
  - Lists of parameters, arguments, … → **??var**
- But metavariables alone don‘t give you genericity!

universitätbonn

Generic Aspect Language

- Introduces metavariables for base program entities
  - Singular metavariable:
    `?var` → Types, fields, methods, …
  - List metavariable:
    `??var` → Lists of parameters, arguments, …

```
aspect EclipsePackageAccessContract {
  declare warning :
    call(* ?apiPackage.internal.**.*.*(..))
    && !(
         within(?apiPackage) )
      || within(?apiPackage.internal.**)
    :
    "The call violates the Eclipse package access contract.";
}
```

universitätbonn

Uniformly Generic Aspect Language

- Use of Metavariables in all aspect language elements
  - pointcuts
  - introductions
  - advice

➔ Context-dependent aspect effects

  - Metavariables are bound in pointcuts providing values for introductions / advice

```
// Generic advice: replace constructor calls
Object around(?sub, ??args) :
    replace(?super,?sub) &&
      // Check if ?sub is a subclass of ?super:
    subtype(?sub,?super) &&
      // Intercept ?super constructor invocations:
    call(?super.new(..)) &&
      // Bind ??args to the argument list of the invocation:
    args(??args)
{     // Return instance of posing subclass ?sub
    return new ?sub(??args);
}
```

LogicAJ

- Uniformly generic aspect language
  - Generic constraints
  - Generic advice
  - Generic introductions
- Additional “pointcut predicates”
  - Constrain values of metavariables
  - Do not select pointcuts
  - Examples
    - method(public ?retType Target.?m(??params))
    - subtype(?sub, ?super)

universitätbonn

Overview

- ✓ Many Problems
- ✓ A Concept → Uniform Genericity
- ✓ A Language → LogicAJ
- ➔ Many Solutions → Demos
- Wrap up

universitätbonn

Visitor revisited

- Visitor Pattern with AspectJ
  - [Hanemann & Kiczales02]
- Fixed number of node types in aggregate hierarchy
  - Visit method for terminal nodes
  - Visit methods for non-terminal nodes
  - *“In cases where the aggregate structure contains more types of nodes, this aspect cannot be used without modifications.”*

```
public interface NodeVisitor {

   public void visitRegularNode(VisitableNode node);

   public void visitLeaf(VisitableNode node);

   public String report();
}
```

universitätbonn

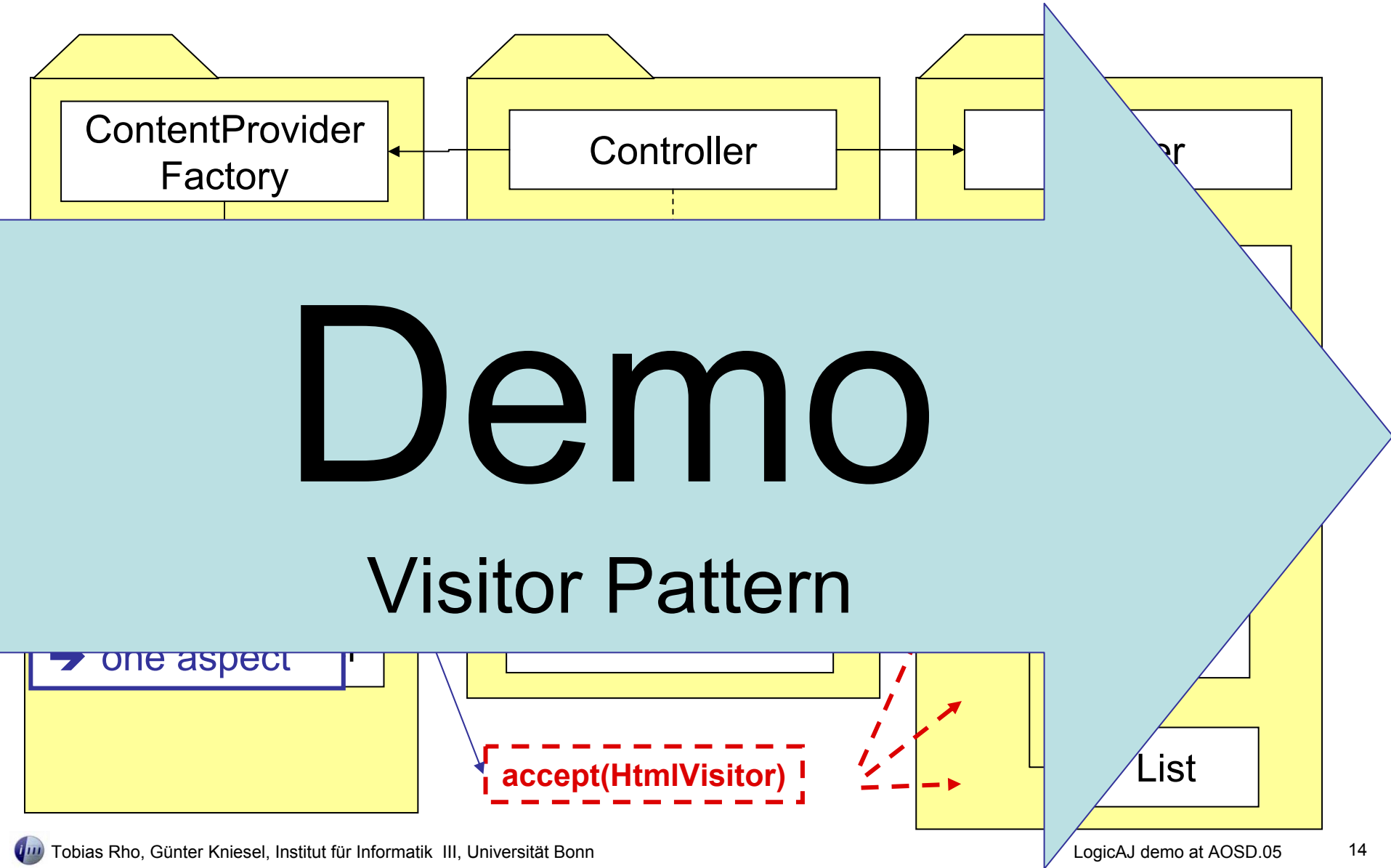
Visitor revisited

- Visitor Pattern with AspectJ
  - [Hannemann & Kiczales02]
- Visit methods not specific for argument types
  - Parameter type of visit methods is too general (Visitable)
  - Need to use type tests and casts
  - Different agregate types maped to the same
  - Partly defeats the purpose of the pattern

```
public void visitNode(VisitableNode node) {
    if (node instanceof BinaryTreeNode) {
        BinaryTreeNode rnode = (BinaryTreeNode) node;
        rnode.left.accept(this);
        rnode.right.accept(this);
    }
}
```

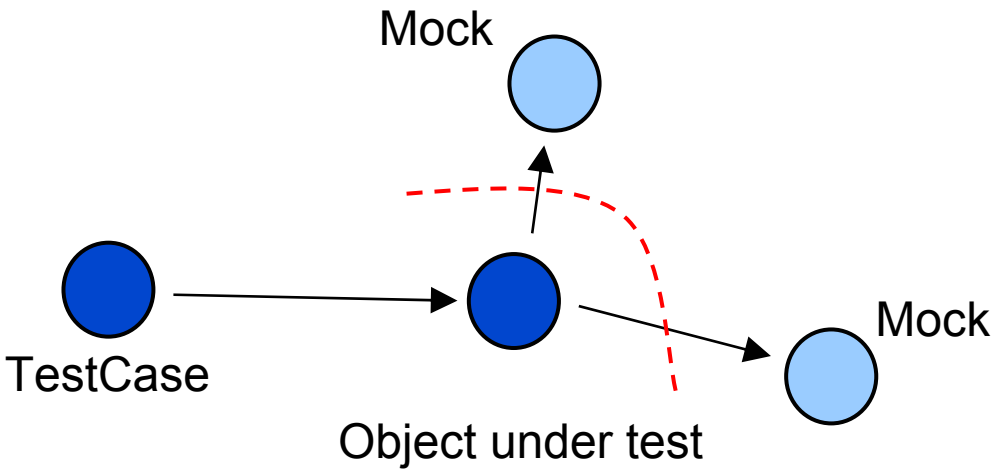
universitätbonn

Demo Application: HTML-Export to Wiki



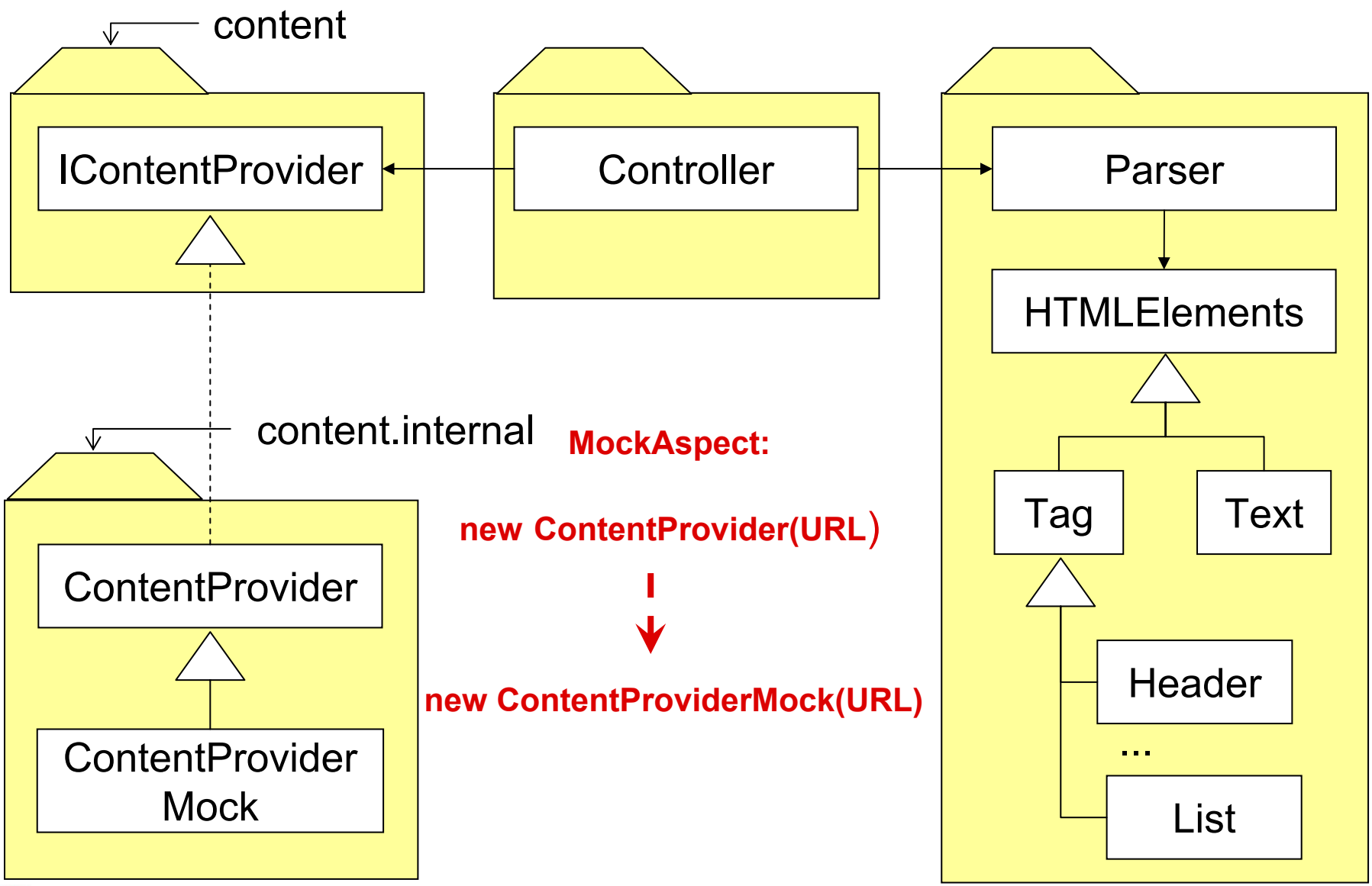
3. Running without network connection

- Mock object
  - used in unit tests
  - mock object implements default behavior for a specific class
  - normal objects are replaced by mock object for a test run



universitätbonn

Demo Application: ContentProvider Mock



Related Work: Parameterization

- [Silaghi03]
  - Genric type variables in pointcuts and advice
- [Loughran&Rashid04]
  - Lancaster Frame Processor
  - Parametrized aspects
  - Textual macro expansion
- Neither can express context dependent aspect effects
  - Missing: Metavariables in pointcuts

universitätbonn

Related Work: Partial Genericity

- [Alvarez04]
  - String-valued parameters for names of types, methods, fields
  - No pointcuts but „parametric expressions“
  - Cartesian product semantics
  - No implementation

- [Gybels01], [Gybels&Brichau03]
  - Metavariables in pointcuts and advice → generic advice
  - No generic introductions
  - Smalltalk prototype

universitätbonn

Related Work: Uniform Genericity

- [Harrison04]
  - PlainWay specification
  - Metavariables in composition specifications
  - Uniform genericity?

- [Hanneberg & Unland 03] Sally
  - Generic introductions
  - Close cooperation [Rho,Kniesel,Hanenberg 04]
  - Generic advice in recent „Sally“ release (Nov. 2004)

universitätbonn

Conclusions

- Uniform Genericity: Metavariables everywhere!
  - Generic pointcuts
  - Generic introductions
  - Generic advice
- Expressiveness Benefits
  - Context-dependent aspect effects
- Software Engineering Benefits
  - Reduced dependency on base entities
  - Reduced aspect-internal redundancy
  - Reduced need to fall back to reflection
  - Truely reusable aspects

universitätbonn

Questions?

- For more infos see: roots.iai.uni-bonn.de/research/

universitätbonn