Composing Configurable Java Components

Tijs van der Storm, CWI, Project Deliver

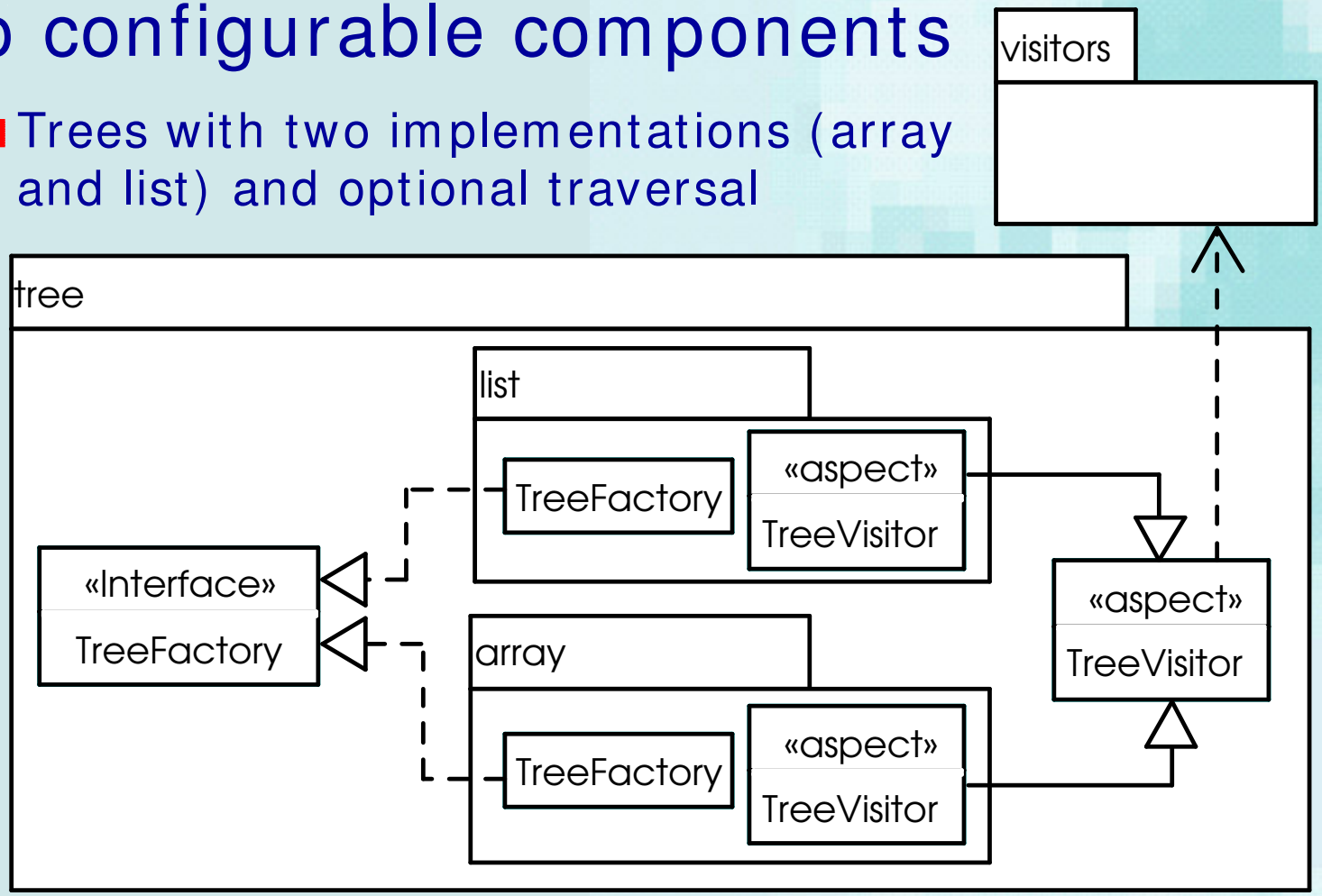
Introduction

- Project *Deliver:*
  - Intelligent Knowledge Management for Software Delivery
- Focus on release and delivery for product lines
- This talk:
  - Composing Configurable Java Components
- How to apply product line technology in a setting where components themselves are configurable?

Jacquard

Two configurable components

- Trees with two implementations (array and list) and optional traversal



Jacquard

Implementing variation

- Programming language imposes restrictions on variation mechanisms
- Use Java properties to select TreeFactory implementation
  - Runtime configuration at program start-up.
- Use Aspect-Oriented Programming (AOP) to add Visitor design pattern
  - Trees should implement Visitable interface
  - AOP is used to add this interface

Jacquard

Binding Factory implementation

- Property used to resolve concrete factory
- Example:

```
treefactory=list.TreeFactory
```

- Dynamic loading and reflection are used to instantiate the factory
- Consequences:
  - Property must be *set* according to requirements

Jacquard

Enabling optional Tree traversal

- AOP is used to weave in Visitor design pattern
-
```
public aspect TreeVisitor {
   declare parents: Tree extends Visitable;
   public void Tree.accept(Visitor v) {
      v.visit(this);
   }
}
```
- Consequences:
  - Build process must call AspectJ
  - There is a dependency on the Visitors component

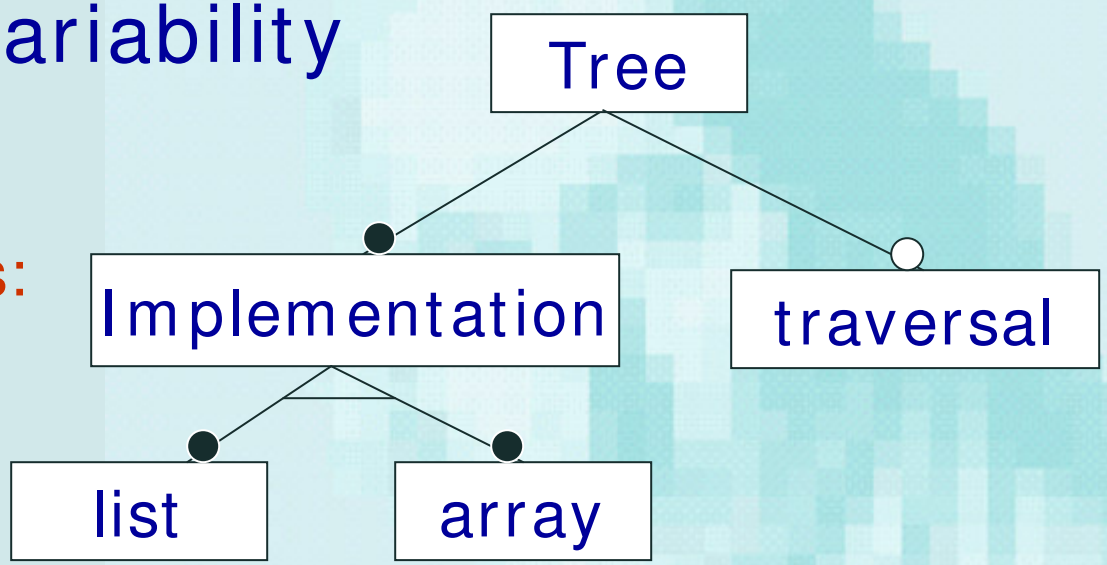
Towards automatic delivery of compositions

- Tree component has 4 variants:
  - array with traversal, array without traversal
  - list with traversal, list without traversal
- How to implement this component family?
  - How do we set the factory property?
  - Do we call AspectJ by hand?
- Moreover: propagation of variability
  - What if Visitor component is configurable as well?
  - How to verify inter-component configuration?
- Our approach: component description language
  - Variability: feature descriptions
  - Binding: (conditional) binding actions

Jacquard

Description of variability

Feature diagrams:



Feature descriptions:

Tree: **all**(Implementation, traversal**?**)

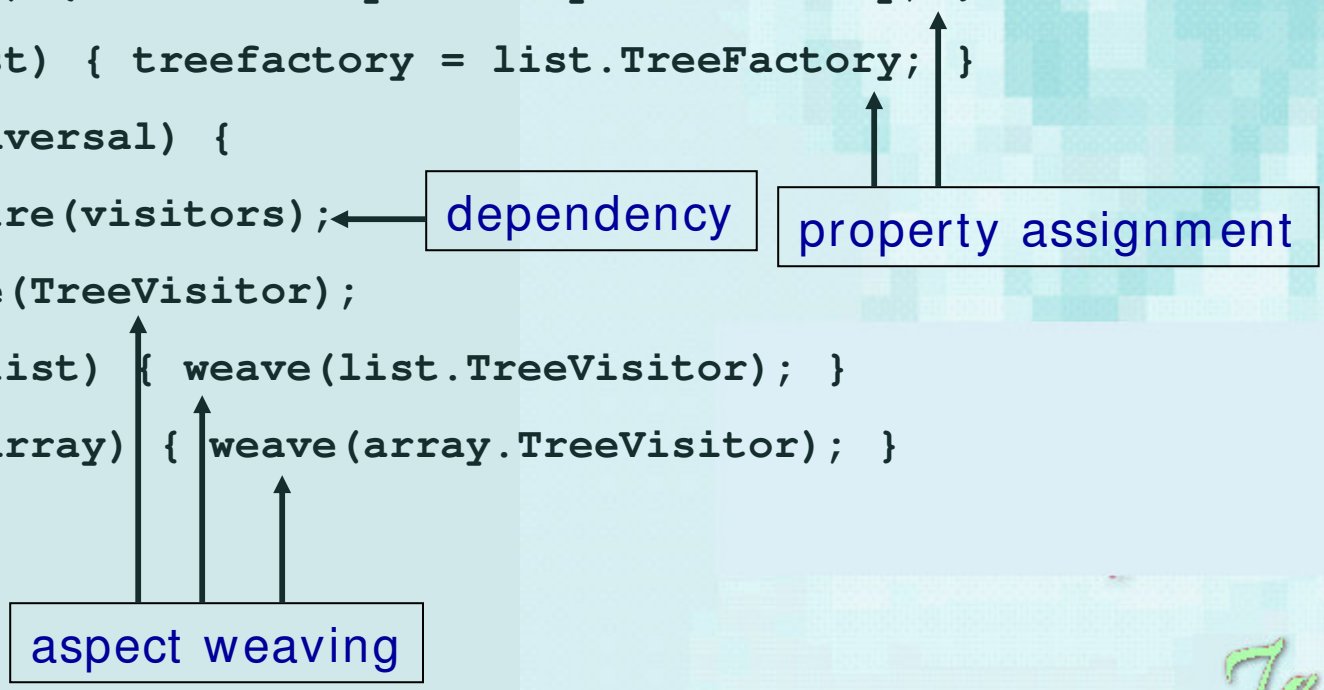
Implementation: **one-of**(list, array)

Jacquard

Description of binding and composition

- (Conditional) binding actions
- Atomic features function as guards:

```
if (array) { treefactory = array.TreeFactory; }
  if (list) { treefactory = list.TreeFactory; }
  if (traversal) {
    require(visitors);
    weave(TreeVisitor);
    if (list) { weave(list.TreeVisitor); }
    if (array) { weave(array.TreeVisitor); }
  }
}
```



Jacquard

Component description language (CDL)

- Feature descriptions + binding actions = CDL
- CDL can be formally analyzed
- Feature descriptions checked for consistency
- Binding checked against feature descriptions

- Correct configuration guaranteed!

Jacquard

Benefits

- Configurable components are more reusable
- Locality of variability improves maintenance
- Consistency of configuration guaranteed
- Compositions are automatically derived

Jacquard

Summary

- Configurable Java Components
- Composition becomes complex:
  - Configuration correctness
  - Binding of features
- CDL for automation:
  - Feature descriptions
  - Binding actions

Jacquard

End

- Deliver project: http://www.cwi.nl/projects/deliver
- Technical report available
- More info: http://www.cwi.nl/~storm

Thank you!

Loading the concrete factory

```
String cls = System.getProperty(“treefactory");
ClassLoader cl = ClassLoader.getSystemClassLoader();
TreeFactory tf =
  (TreeFactory)cl.loadClass(cls).newInstance();
```