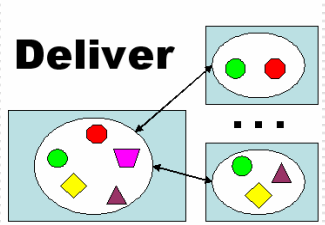
Generic Feature-Based Composition

Tijs van der Storm
Centrum voor Wiskunde en Infromatica
storm@cwi.nl



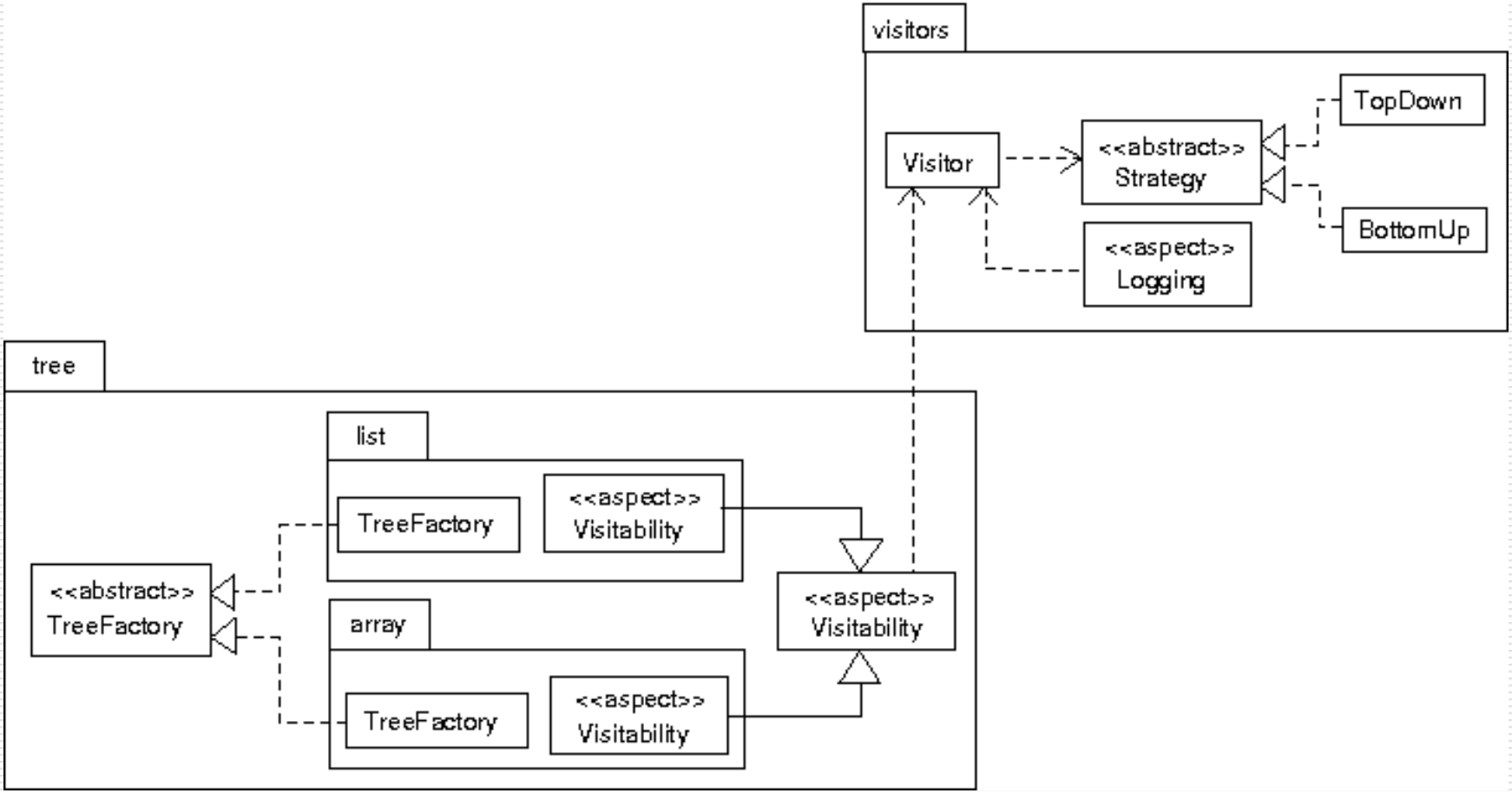
Introduction

- Me: Tijs van der Storm
- Phd Student, project *Deliver*:
  - Intelligent Knowledge Management for Software Delivery
- My focus: software configuration management
- This talk:
  - Relating “variability” to “variation”
  - Steps towards bridging
    - Problem domain (features)
    - Solution domain (artifacts)

Overview

- Example product line
- Goal of this work: automation of configuration
- Description of
  - Variability model
  - Composition model
  - Link between them
- Conclusions

Example product line



Some Questions

- What are valid configurations?
- What are the variation points?
- How are they bound?
- How many variants are there?

Goal: automatic instantation

- Why?
  - Configuration space can be large
  - Binding is manual/error-prone
- Automation requires:
  - Formal description of configuration space
  - Formal link to solution space artifacts

Configuration impedance mismatch

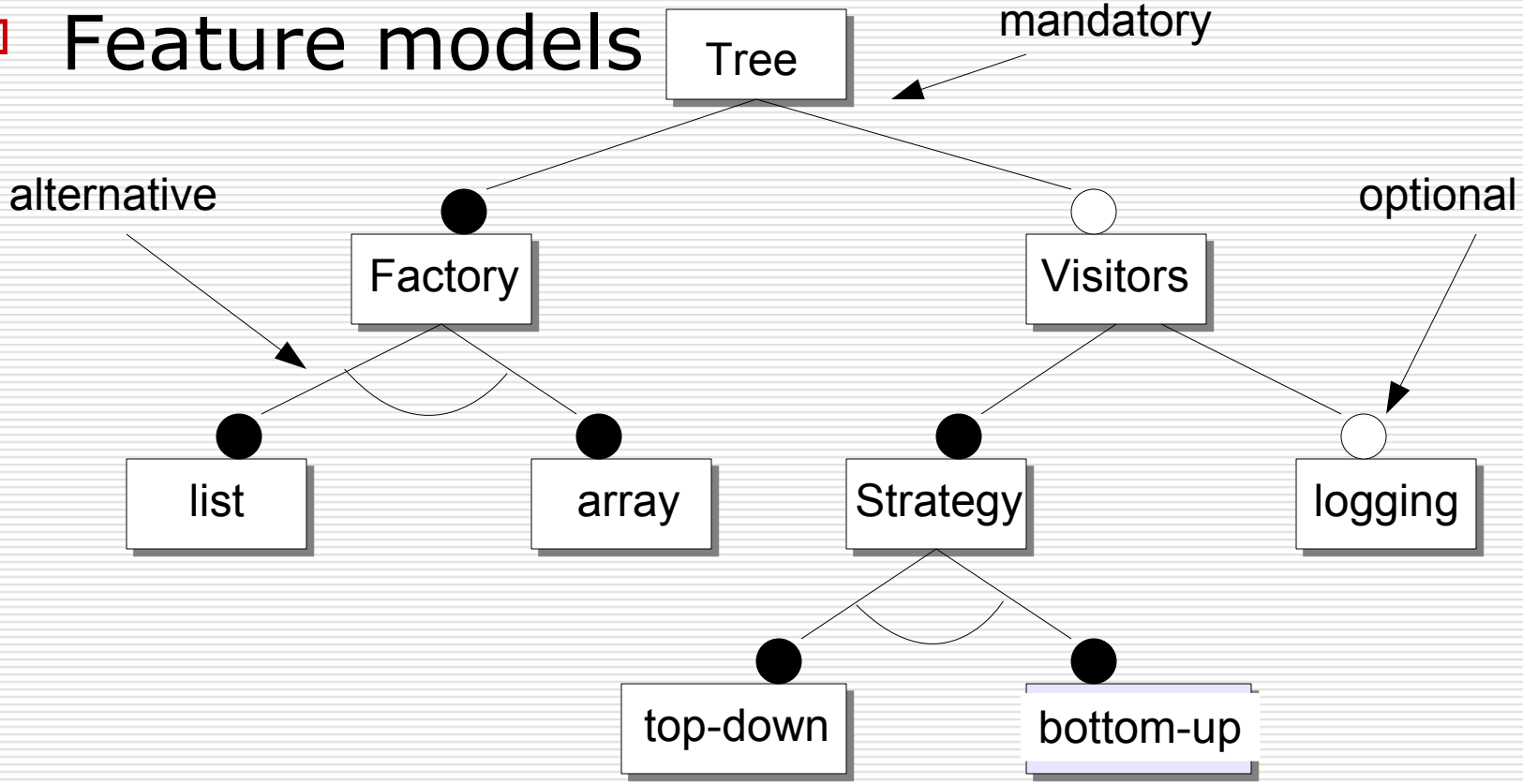
- Configuration based on *features*
  - Abstract
  - Declarative
  - Logical
- Composition based on *artifacts*
  - Concrete
  - Structural
- Never the twain will meet?

Plan

- Configuration interfaces
  - Feature diagrams/descriptions
- Composition model
  - Dependency graphs
- Relate both through common logical formalism (propositional formulae)
- Derive consistency and product instantiation

Variability model

- Feature models

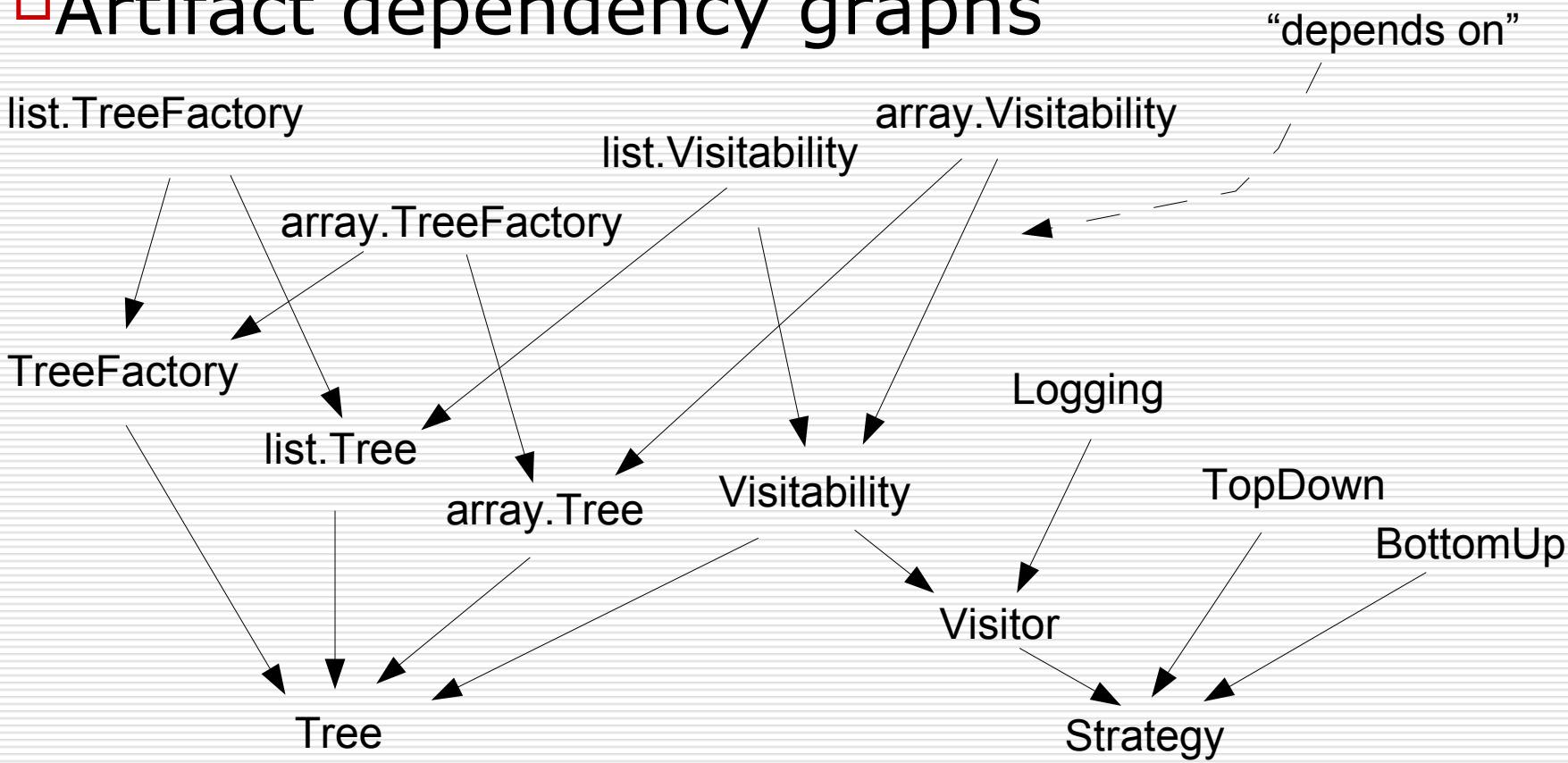


Translation to logic

- Textual description of diagram (FDL)
  - Tree: **all**(Factory, Visitors**?**)
    Factory: **one-of**(list, array)
    Visitors: **all**(Strategy, logging**?**)
    Strategy: **one-of**(top-down, bottom-up)
- Translation to boolean sentence
  - (Tree -> Factory) & (Factory -> ((list & not(array)) | (not(list) & array))) & (Visitors -> Strategy) & (Strategy -> ((top-down & not(bottom-up)) | (not(top-down)) & bottom-up))))

Composition model

- Artifact dependency graphs



Genericity

- Variation can be implemented in many ways
  - design patterns
  - aspect oriented programming (AOP)
  - build files
  - ...
- Dependency relation sufficiently generic
  - independent of programming language
  - works on files or directories

Translation of dependencies

- Nodes are atoms, edges implication
  - (list.TreeFactory -> (TreeFactory & list.Tree)) & (array.TreeFactory -> (TreeFactory & array.Tree)) & (TreeFactory -> Tree) & (list.Tree -> Tree) & (array.Tree -> Tree) & (Visitability -> Tree) & ...
- Follows from artifacts themselves...
  - Future work: derive this automatically

Merging both domains

- Two boolean propositions:
  - Configuration interface
  - Dependency relations
- Bridged using implications:
  - (List & Visitors -> *list.Visitability*) &
    (array & Visitors -> *array.Visitability*) &
    *(*list -> *list.TreeFactory*) &
    (array -> *array.TreeFactory*) &
    (top-down -> *TopDown*) &
    (bottom-up -> *BottomUp*) &
    (logging -> *Logging*)

Configuration = valuation

Features	*Logic*
Feature	Boolean formula
Dependency	Implication
Featurename	Atom
Artifact	Atom
Configurability	Satisfiability
Configuration	Valuation
Validity	Satisfaction

Set of all valuations = set of all valid instantiations

Valid configurations for *Tree*

Atomic Features	*Artifacts*
array	Tree, TreeFactory, array.TreeFactory, array.Tree
array, top-down	..., array.Visitability, Visitability, Strategy, TopDown
array, bottom-up	..., array.Visitability, Visitability, Strategy, BottomUp
array, top-down, logging	..., array.Visitability, Visitability, Strategy, TopDown, Logging
array, bottom-up, logging	..., array.Visitability, Visitability, Strategy, BottomUp, Logging
list	Tree, TreeFactory, list.TreeFactory, list.Tree
list, top-down	..., list.Visitability, Visitability, Strategy, TopDown
list, bottom-up	..., list.Visitability, Visitability, Strategy, BottomUp
list, top-down, logging	..., list.Visitability, Visitability, Strategy, TopDown, Logging
list, bottom-up, logging	..., list.Visitability, Visitability, Strategy, BottomUp, Logging

Maintaining the mapping

- Mapping co-evolves with code base
- But, weak coupling
  - Map features to *essential* artifacts
  - Many artifacts induced by transitivity
  - Static checking to spot errors

Conclusions

- Bridge from problem domain to solution domain for automating configuration
- Combination of two models
  - Features
  - Dependencies
- Future work:
  - Tool support
  - Dealing with evolution
  - Extension with binding actions

Thank you

Questions?