Supporting Product Line Architectures using Configuration Management Techniques

Nenad Medvidovic
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
neno@usc.edu
http://sunset.usc.edu/~neno/
Overview

• Software architectures in a nutshell
• Configuration management in a nutshell
• Motivating scenario
• SA + CM
• Implications on product lines
Software Architectures

- Model-level artifacts
  - Components
  - Connectors
  - Configurations
  - Constraints
  - Architectural styles and patterns
  - Architecture description languages
  - Reference architectures
  - Product line architectures
Configuration Management

- Source-level artifacts
  - Versioned archives
  - Revisions
  - Variants
  - Optional elements
  - Locks
  - Branches
  - Version trees
  - System models
  - Configurations
Scenario

Word

Architecture

Design
Architecture-Level CM

• Based on and applied to software architecture concepts
• Domain-, ADL- and style-independent
• Types and instances
  • Component
  • Connector
  • Interface
• Versioning types to capture architectural evolution
Mae

• Architectural evolution environment
• Instance of the generic system model
• Mapping onto a specific ADL
Example Product Line

Legend:
- Component
- Connector
- Communication link

- File Repository
  - Active Document Repository
  - SpellCheck Repository
- Font Dialog
- Color Dialog
- File Dialog
- Word Counter
- Spell Checker
- Canvas
- GUI driver
- Toolkit
Component Variations

Name = tSpellChecker
Revision = 2
Interface = { iSpellCheck }
Component = { iTokenizer,
  iResultCollector }
Connector = { iC2bus1,
  iC2bus3 }
Behavior = { iSpellCheck* }
Constraint = { }
Representation = { << … >> }
Ascendant = { tSpellChecker 1 }
Descendant = {tSpellChecker 3}
Style = { C2 }
SubType = { beh \and int }

Name = tSpellChecker
Revision = 3
Interface = { iSpellCheck }
Component = { iTokenizer,
  iResultCollector,
  iStatistics, collectStatistics, true }
Connector = { iC2bus1,
  iC2bus2, collectStatistics, true,
  iC2bus3 }
Behavior = { iSpellCheck* }
Constraint = { }
Representation = { << … >> }
Ascendant = { tSpellChecker 2 }
Descendant = { }
Style = { C2 }
SubType = { beh \and int }
Implications on Product Lines

• CM at the level of architectures
  • Direct applicability to product-line architectures
• Automated change script generation
  • Patches at the level of product-line and product architectures
• Variability and optionality
  • Components
  • Connectors
  • Entire configurations
• Propagation to implementation and back
• CM provides ability to change *anything*
• Architecture constrains that change
Acknowledgements

• Andre van der Hoek, UC Irvine
• Marija Rakic, USC
• Roshanak Roshandel, USC

Additional information: