

“... today ... 1,700 special programming languages used to ‘communicate’ in over 700 application areas.”

– *Computer Software Issues*, an American Mathematical Association Prospectus, **July 1965**, quoted in P. J. Landin’s famous article “The Next 700 Programming Languages”, 1966

Feature Modeling in Generative Software Development

Krzysztof Czarnecki
University of Waterloo



Overview

- ➔ Generative Software Development
 - Example
 - Staged Configuration
 - Tool support
 - Summary

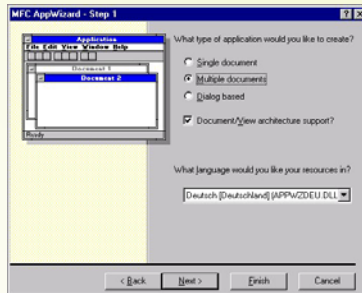
Generative Software Development

- Is a product-line engineering approach
 - Product-line development
 - Product development
- Automates product development
- Based on a product specification in a set of *domain-specific languages*

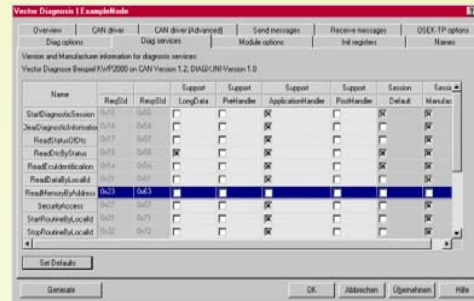
Advantages of DSLs Over General-Purpose Languages (GPLs)

- Domain-specific abstractions
 - Pre-defined abstractions to directly represent concepts from the application domain
- Domain-specific concrete syntax
 - Natural notation for a domain avoiding syntactic clutter that often results when using GPLs
- Domain-specific error checking
 - Analysers that find more errors than similar analysers for GPLs and report errors in a language familiar to the domain expert
- Domain-specific optimizations
 - Code optimization based on domain-specific knowledge which is usually not available to a GPL compiler
- Domain-specific debugging, version control, etc.
 - Opportunities for improving all aspects of a development environment

Different Forms of DSLs



Wizard



Configuration tool

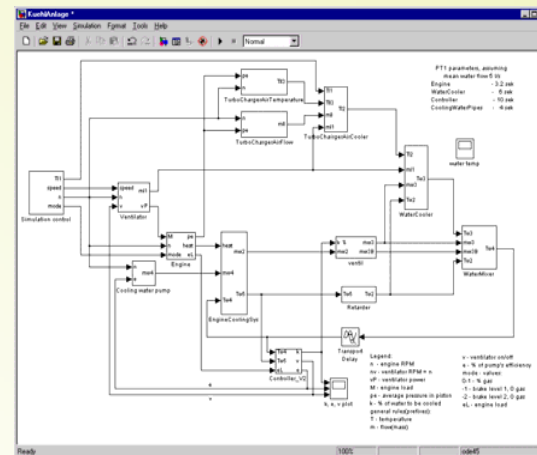


Library in a programming language

```

page MainPage () {
  filename = index.html
  placeHolders {
    header = fragmentCall { include = Header() }
    leftNavigation = fragmentCall { include = LeftNavigation(nil) }
    rightNavigation = fragmentCall { include = RightNavigation() }
    body = fragment { filename = main/main.html
                    filterElement = body }
    footer = fragmentCall { include = Footer() }
  }
}
    
```

Textual DSL

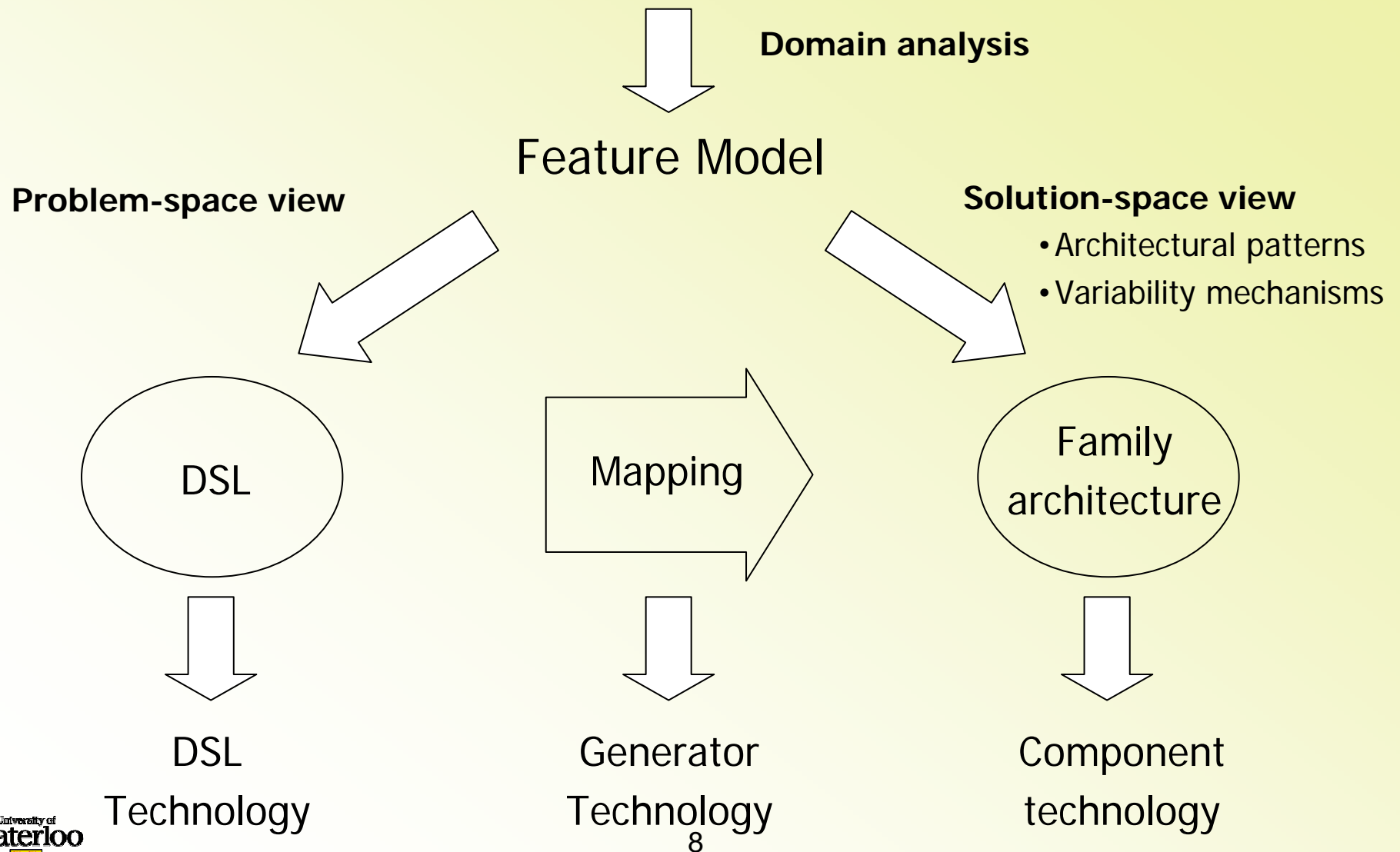


Visual DSL

Modeling and DSLs

- Models are abstract representations of systems
 - Answer questions of interest to stakeholders
- Capture stakeholder intentions more directly
 - Without accidental implementation details
- Model driven development makes them source artifacts
 - Fully integrated with the code and other source artifacts
 - Not documentation that becomes obsolete as software is cut
- Used across the software life cycle
 - Requirements, design, development, deployment, testing, debugging, maintenance, enhancement
- DSLs are perfect for model driven development
 - Capture more information with higher fidelity than general purpose modeling languages designed for documentation

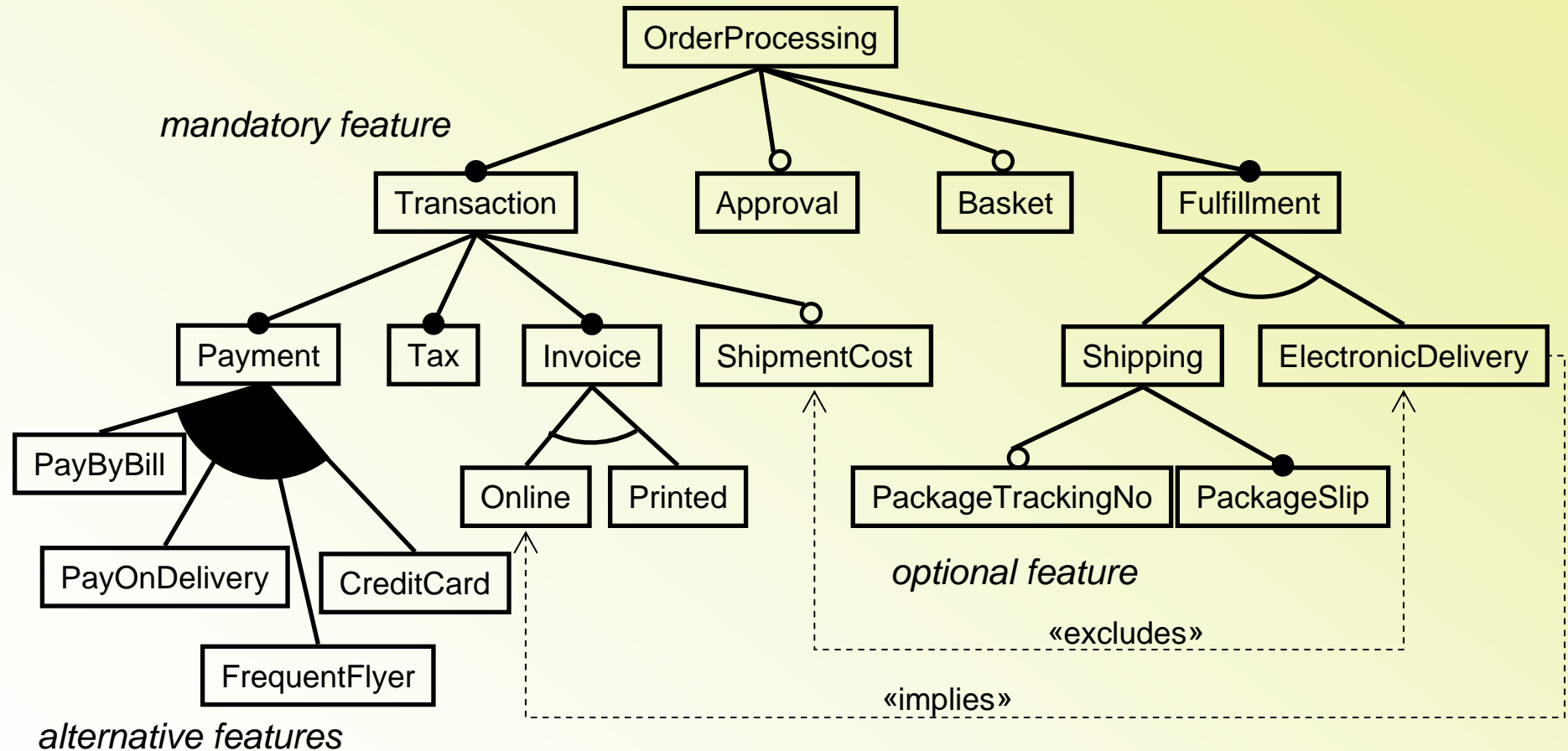
Feature-Oriented Approach



Overview

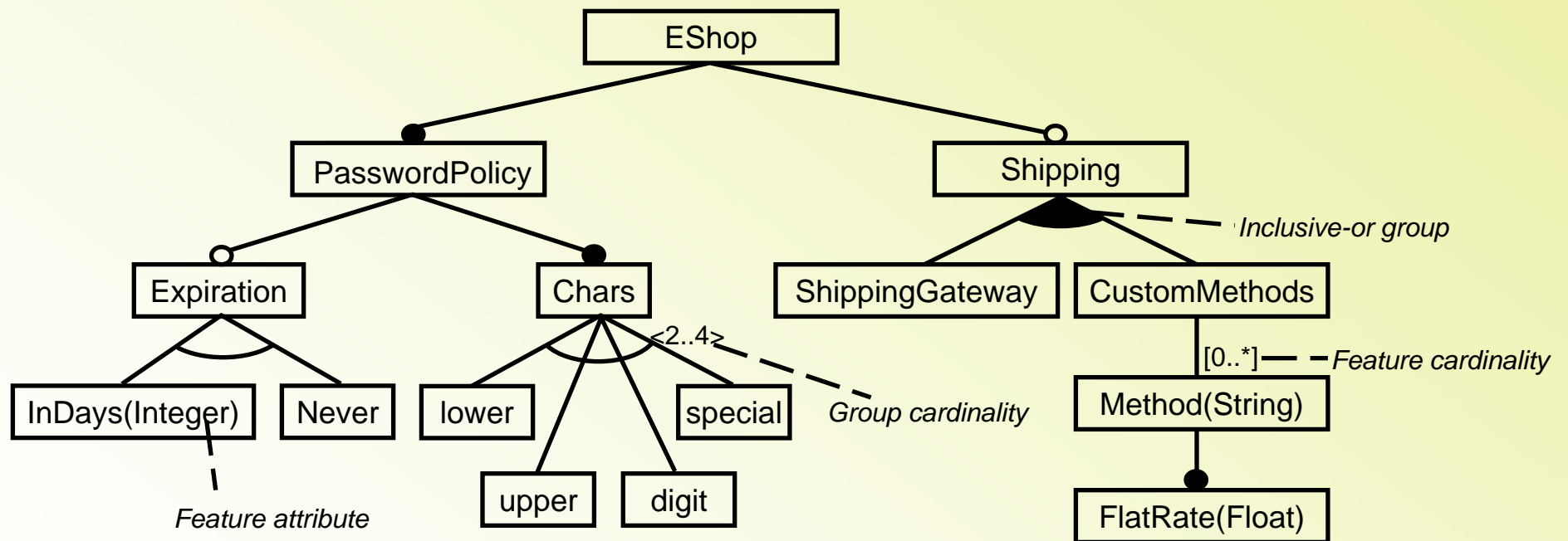
- Generative Software Development
- ➔ Example
- Staged Configuration
- Tool support
- Summary

Feature Diagram



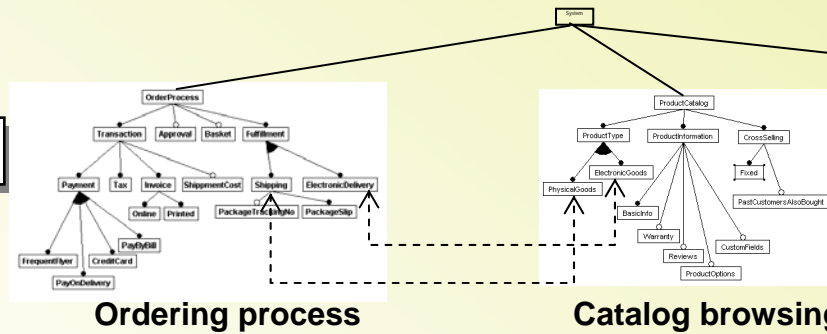
No decision regarding the mechanism for implementing variability!

Cardinality-Based Feature Modeling



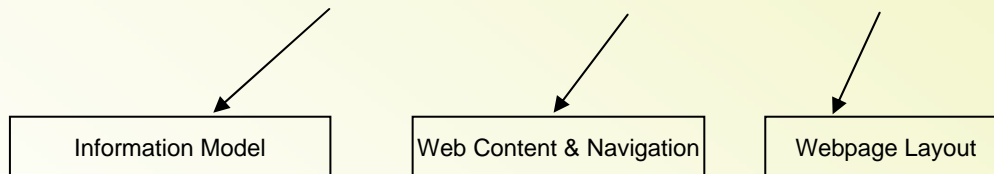
Mapping Feature Variations To Software

Variability



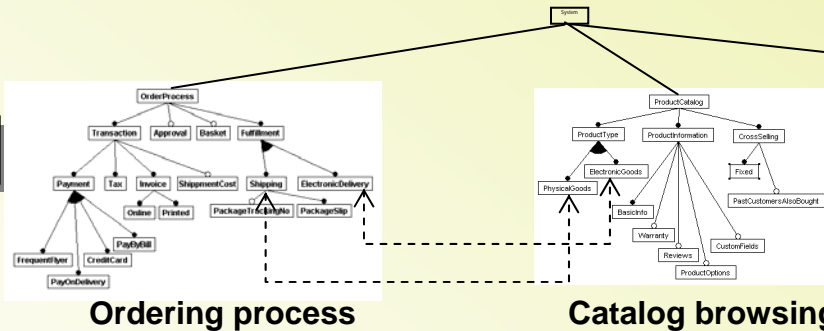
Targeting & Profiling

- ...
- Security polices**
- Transaction polices**
- Caching polices**
- Presentation polices**
- ...

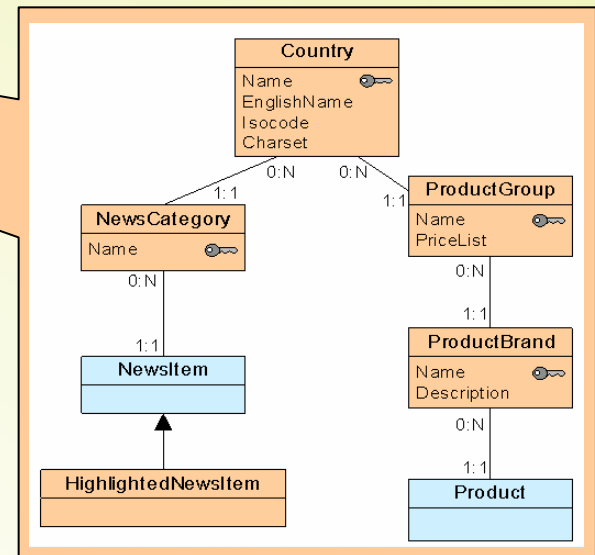
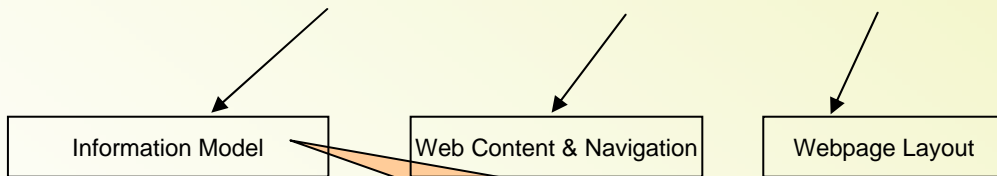


Mapping Feature Variations To Software

Variability

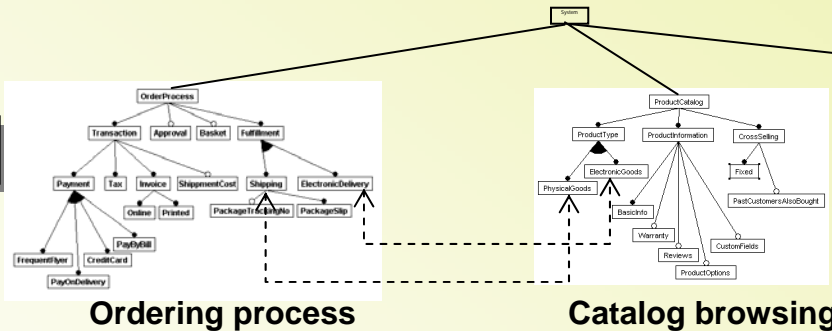


Targeting & Profiling
 ...
Security polices
Transaction polices
Caching polices
Presentation polices
 ...

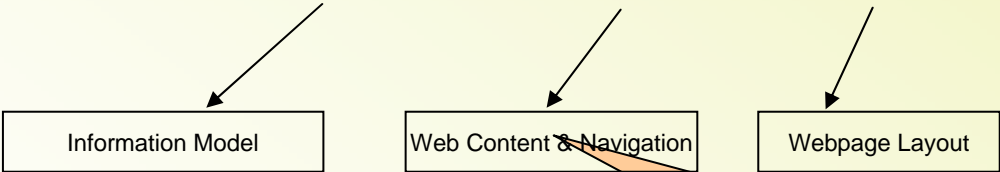


Mapping Feature Variations To Software

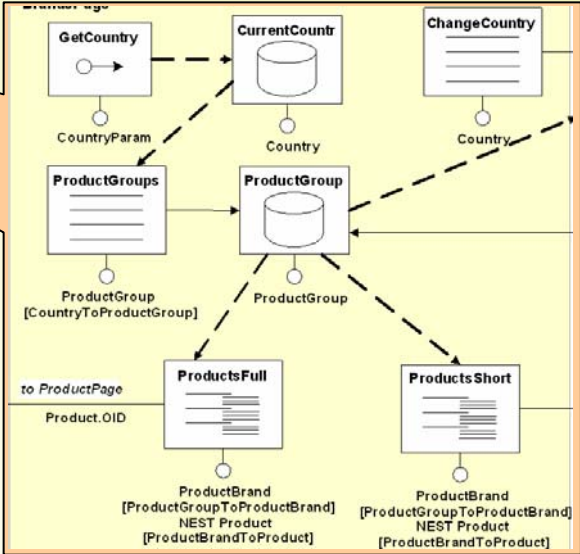
Variability



Targeting & Profiling
 ...
Security polices
Transaction polices
Caching polices
Presentation polices
 ...

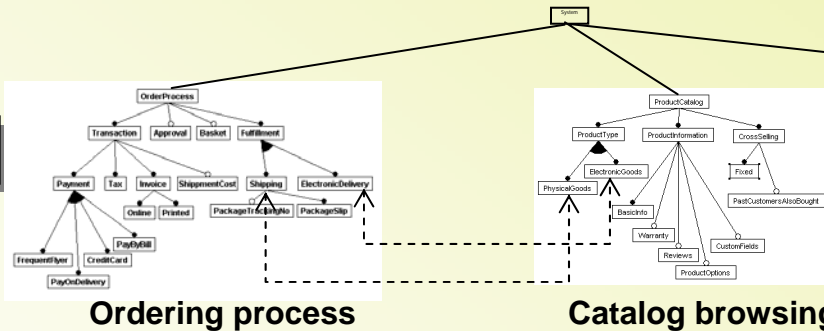


See webml.org



Mapping Feature Variations To Software

Variability



Ordering process

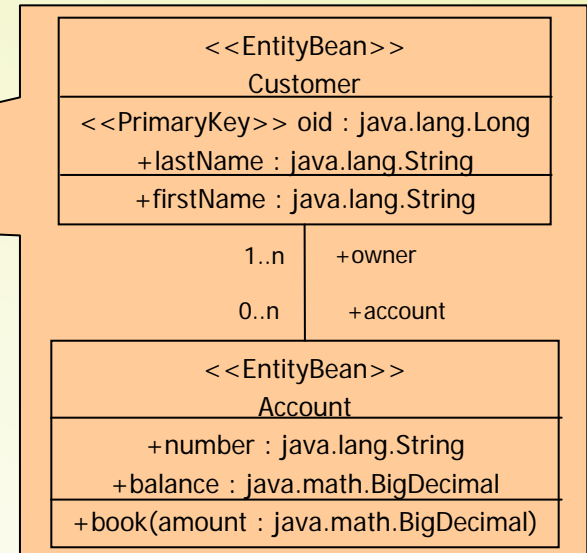
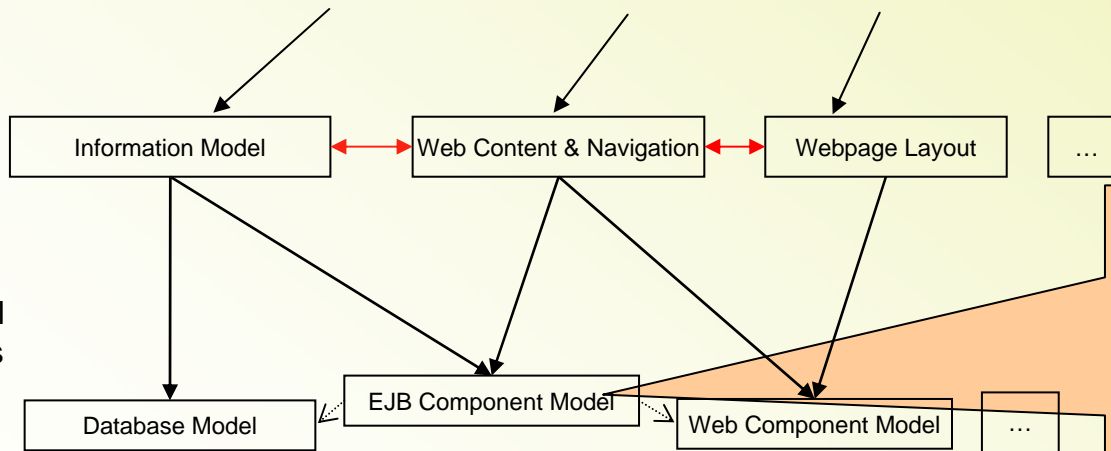
Catalog browsing

Targeting & Profiling

- ...
- Security polices**
- Transaction polices**
- Caching polices**
- Presentation polices**
- ...

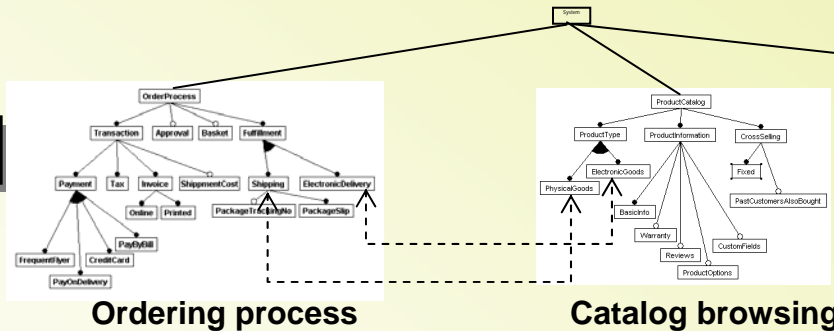
Consistency management & reconciliation

Refinements - Model-to-model transformations



Mapping Feature Variations To Software

Variability



Ordering process

Catalog browsing

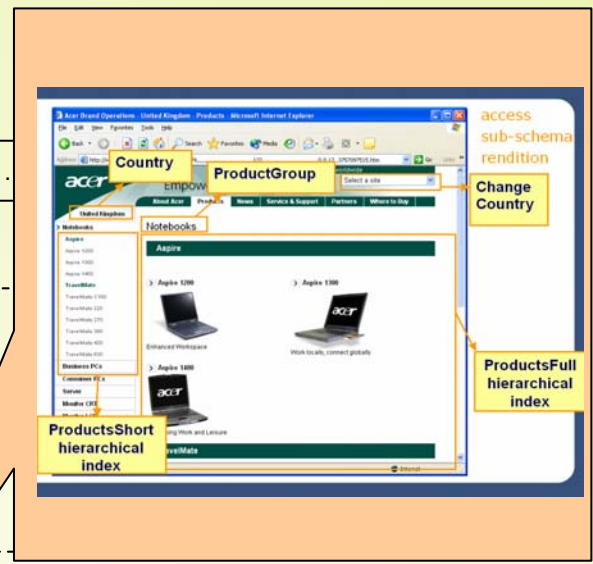
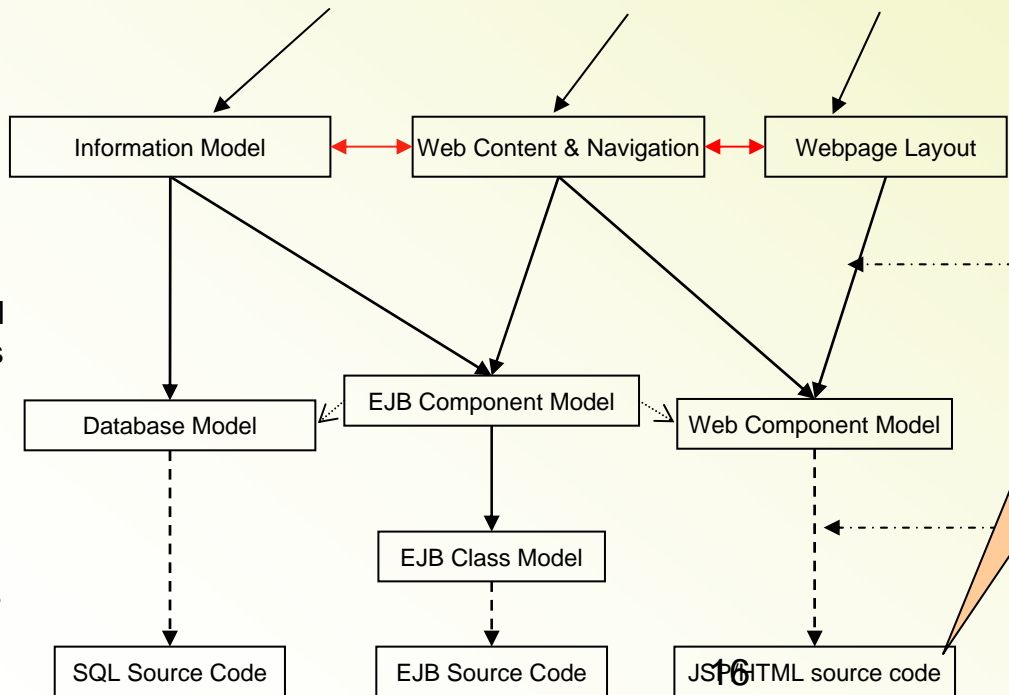
Targeting & Profiling

- ...
- Security polices
- Transaction polices
- Caching polices
- Presentation polices
- ...

Consistency management & reconciliation

Refinements - Model-to-model transformations

Refinements - Model-to-code transformations

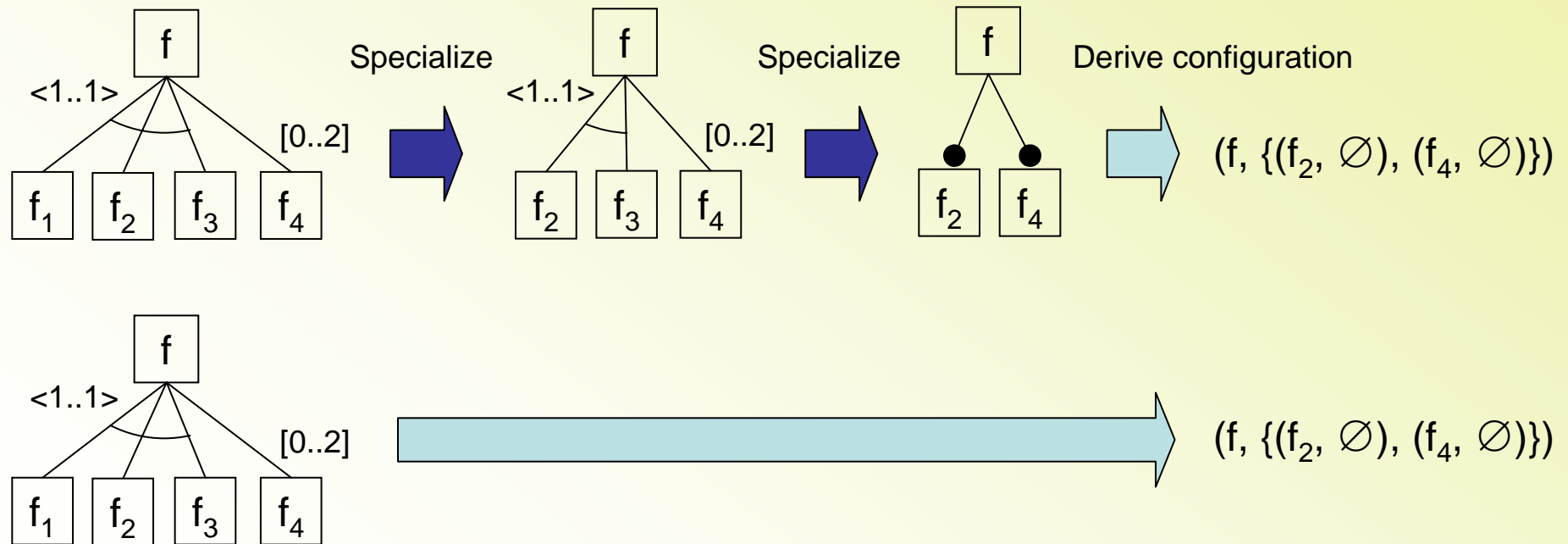


Overview

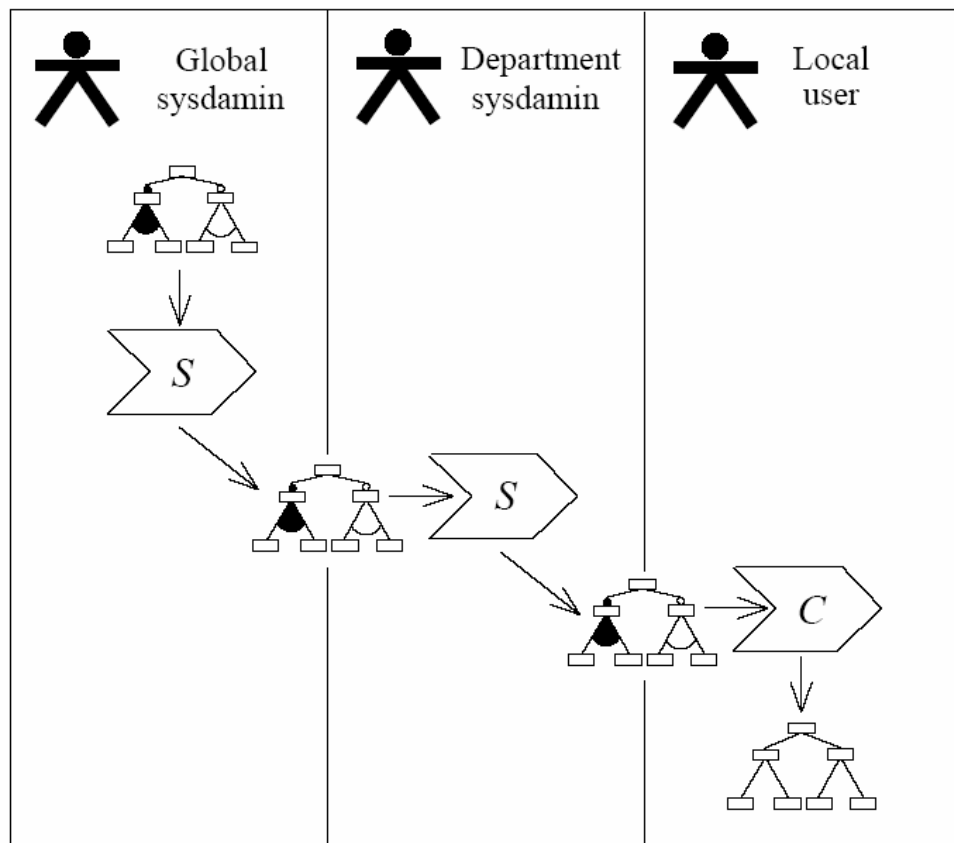
- Generative Software Development
- Example
 - ➔ Staged configuration
- Tool support
- Summary

Staged Configuration Using Specialization

- Specialization transforms a new feature diagram into a new one denoting a subset of configurations of the original diagram



Staged Configuration Example: Policy Specialization



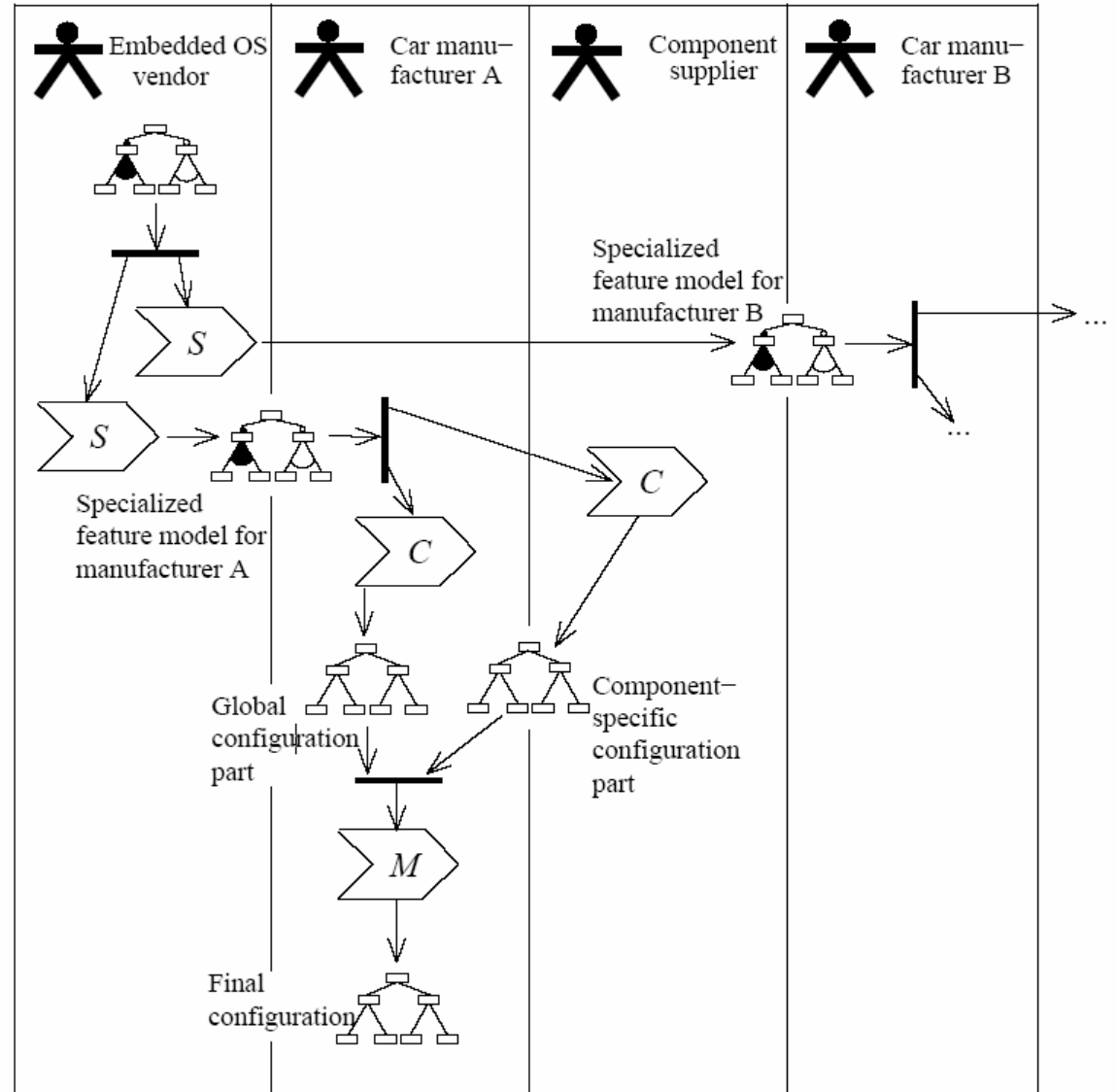
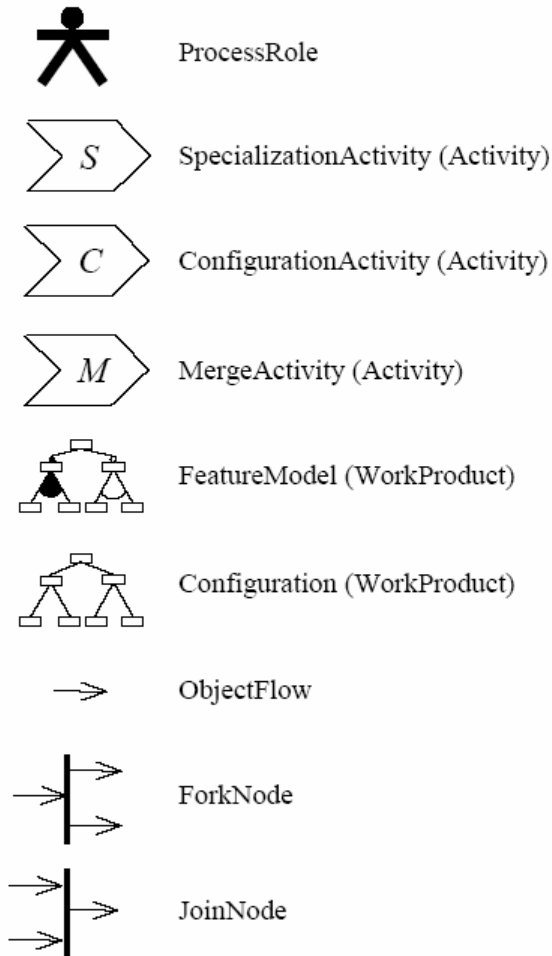
Decisions taken in stages

- Time, e.g.,
 - Phases in product lifecycle
- Roles, e.g.,
 - Component integrator, deployer, administrator, maintainer
- Context/target, e.g.,
 - Multiple deployment contexts

Software Supply Chains

Legend:

(Nonstandard elements have their base class indicated in parentheses.)



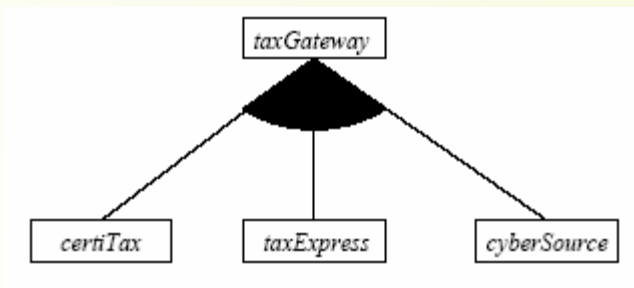
Staged Configuration Using Multi-Level Configuration

- Each stage has a separate feature model
- Each role performs configuration (not specialization) within a stage
- Manual configuration of one stage automatically specializes the feature model of the next stage

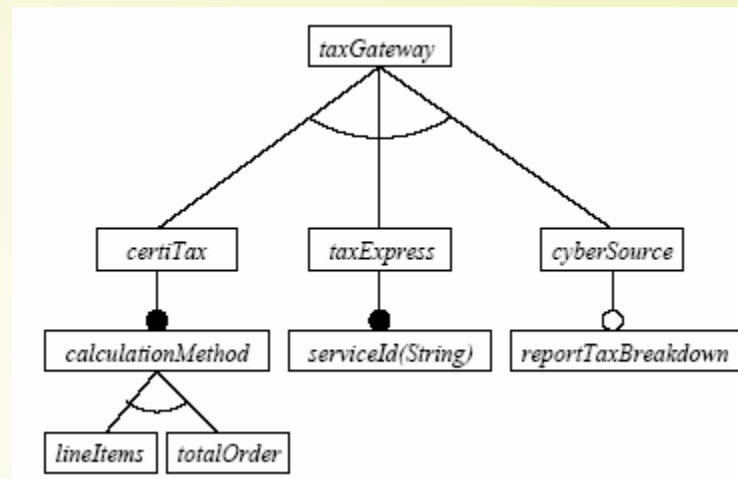
Multilevel Configuration

- Different roles will have different perspectives on the variability

Product-line perspective

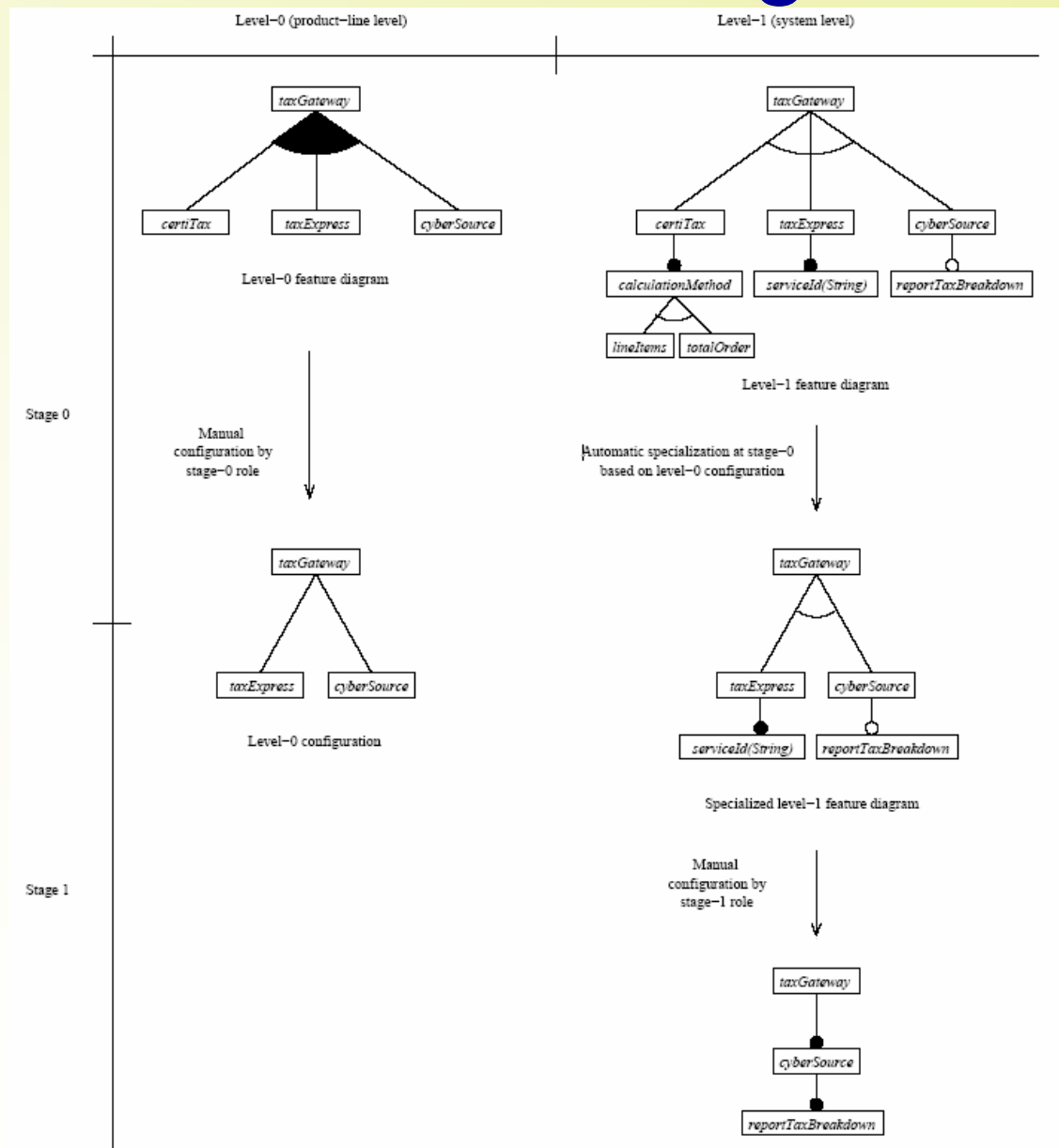


System configuration perspective



Just annotating features with binding times is not enough!

Multilevel Configuration



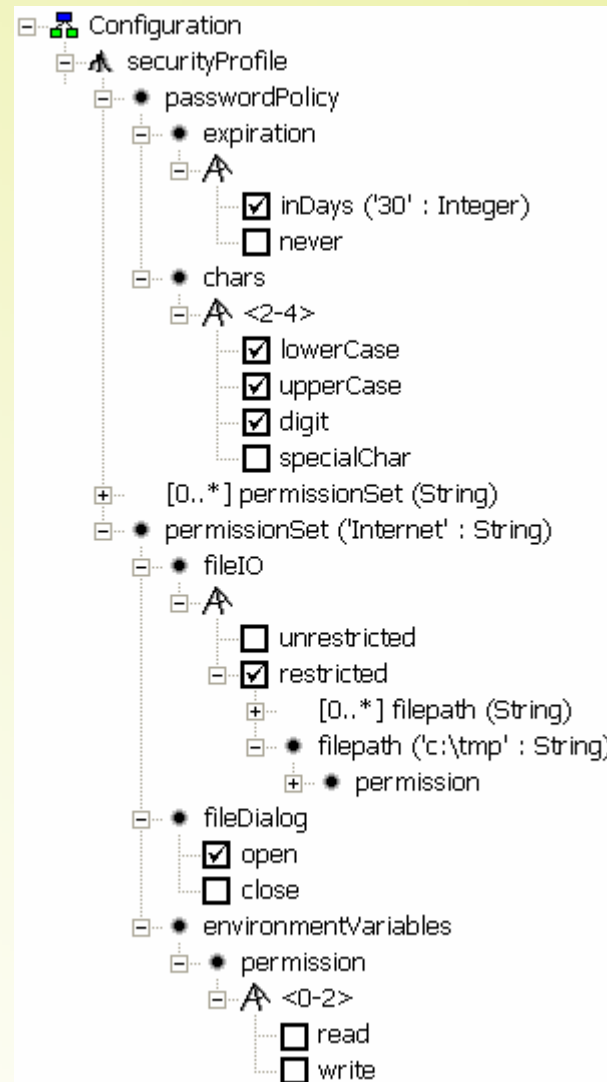
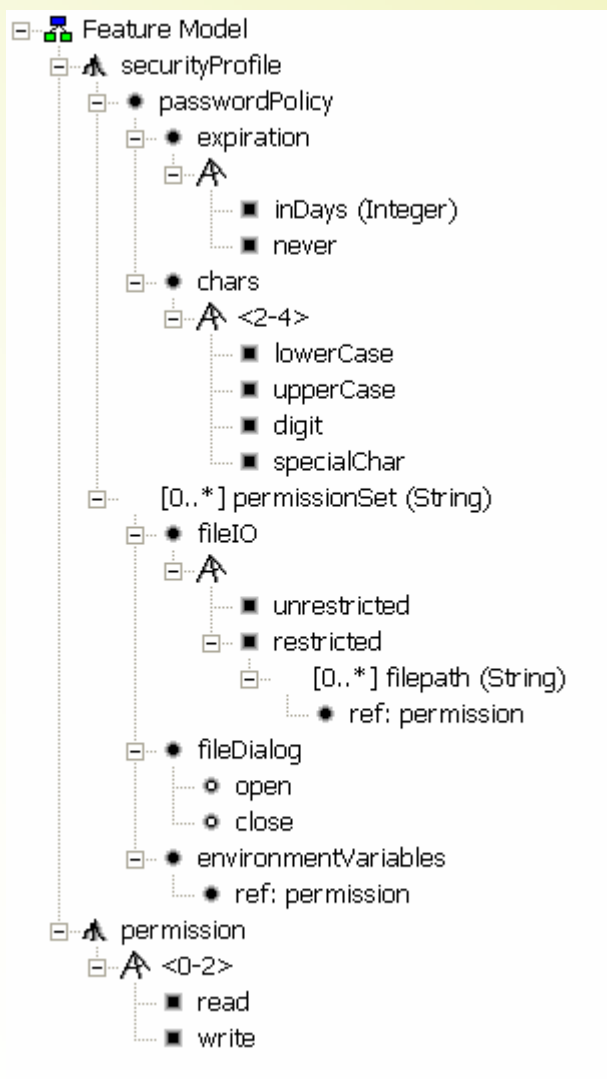
Overview

- Generative Software Development
- Example
- Staged configuration
- ➡ Tool support
- Summary

FeaturePlugin – Eclipse Plugin for Feature Modeling

- Cardinality-based feature modeling
- Configuration support
 - Configuration
 - Specialization
 - Global constraints
- User-extensible notation
 - Defining additional information for a project
 - Meta-level also defined as feature models
- Alternative renderings
 - Explorer view & feature maps
 - Feature trees & tables

FeaturePlugin – Eclipse Plugin for Feature Modeling and Configuration



FeaturePlugin

The screenshot shows the Eclipse Platform interface for editing a Feature Model (eShop.featureMdl). The main workspace is divided into three panes:

- Left Pane:** A tree view of the Feature Model. The root is 'Eshop Feature Model', which contains 'Payment', 'Shipping', 'PasswordPolicy (String)', and 'Eshop'. 'Payment' includes 'CreditCard', 'DebitCard', 'PurchaseOrder', and 'FraudDetection'. 'Shipping' includes 'CustomMethods', 'ShippingGateways', and 'Configuration of Shipping'. 'PasswordPolicy' includes 'Expiration', 'Chars', and 'Specialization of PasswordPolicy'. 'Eshop' includes 'ref: Payment', 'ref: Shipping', 'ref: PasswordPolicy', and 'Configuration of EShop'.
- Middle Pane:** A tree view of the 'EShop' feature model. It shows the same structure as the left pane but with some elements checked or unchecked. For example, 'CreditCard', 'DebitCard', 'Shipping', and 'CanadaPost' are checked, while 'FraudDetection', 'PurchaseOrder', 'USPC', and 'Digits' are unchecked.
- Right Pane:** A configuration dialog titled 'Configuration of Payment feature'. It prompts the user to perform appropriate configuration of the 'Payment' feature. The dialog shows a list of features: 'Payment', 'FraudDetection', 'Shipping', and 'PasswordPolicy'. Under 'Optional features with cardinality [0..1]', 'FraudDetection' is unchecked. Under 'Feature Group', 'CreditCard' and 'DebitCard' are checked, while 'FraudDetection' is unchecked. Buttons for '< Back', 'Next >', 'Finish', and 'Cancel' are at the bottom.

At the bottom of the interface, there is a 'Properties' view and a 'Constraints' table.

Properties View:

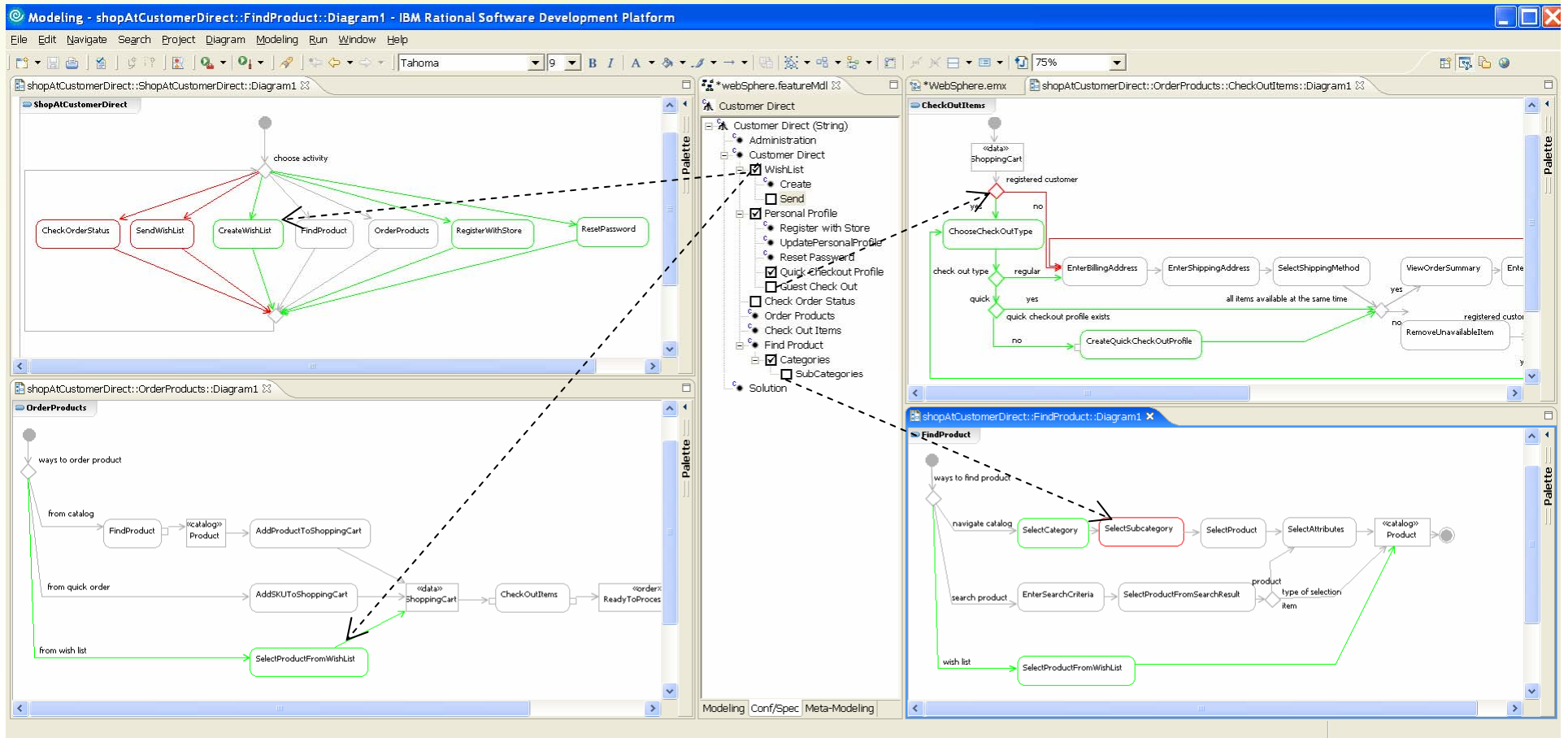
- Name ('Shipping') : String
- Description (String)
- Attribute (checked)
- Integer
- String (checked)
- Value (String)
- DefaultValue (String)
- Float

Constraints Table:

Constraint	Value	Message
<input checked="" type="checkbox"/> every \$x in //CustomMethod satisfies (count(\$x/FlatRate)<10)	true	none
<input checked="" type="checkbox"/> every \$ in //Method satisfies (\$x/FlatRate[@value > 0])	syntax error	net.sf.saxon.xpath.StaticError: XPath syntax error at char 11 in (every \$ in //Method): expected "in", found "/"
<input checked="" type="checkbox"/> count(//CustomMethods) > count(//Method)	false	none

Selected Object: FraudDetection

Mapping Features to Workflow



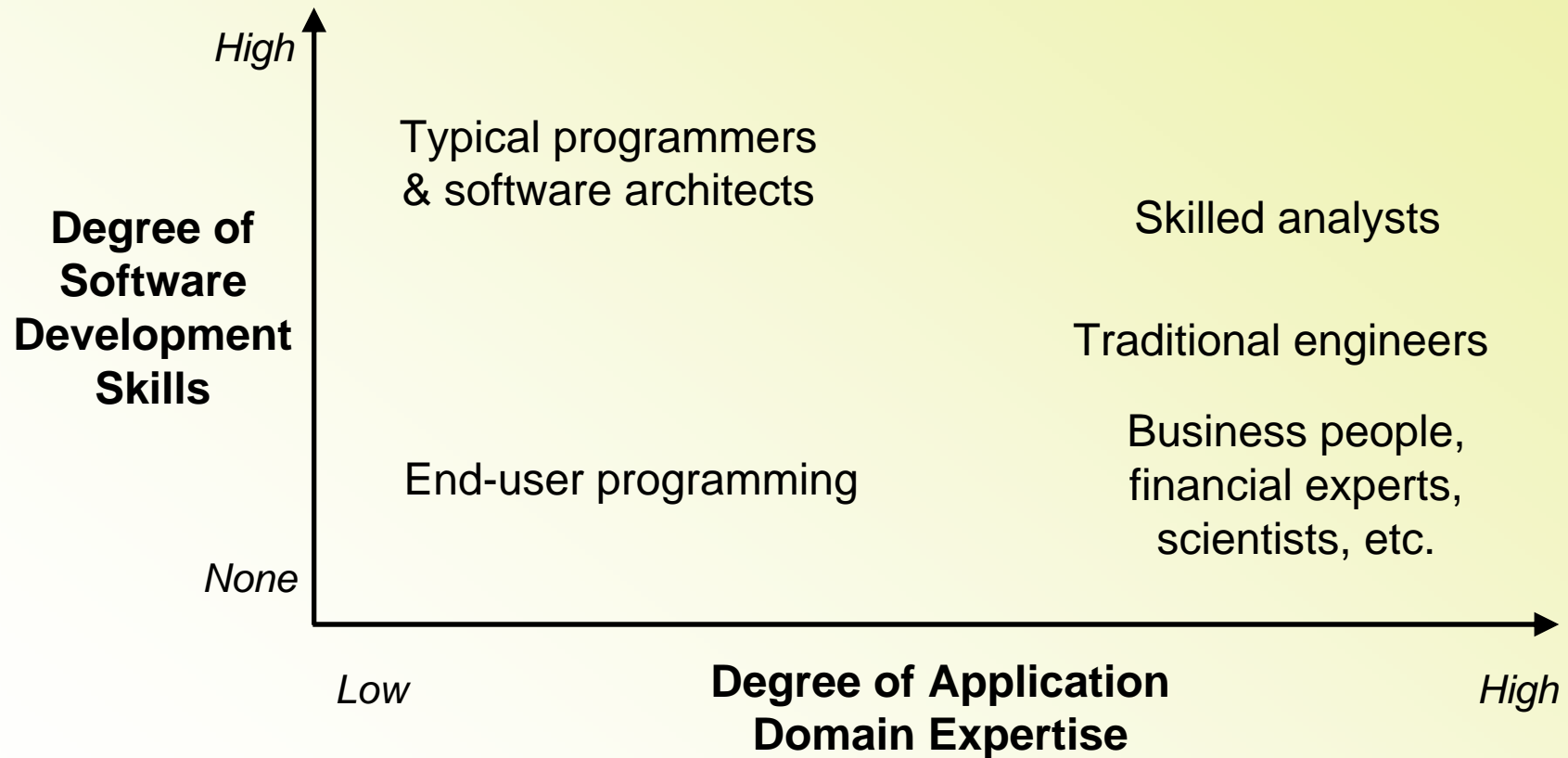
Designing DSLs

- Complex design space
 - Textual, diagrammatic, form-based, grid-based, etc.
 - Hierarchical, graph-like, spatial, etc.
 - Declarative vs. procedural
 - Dynamic vs. static
 - ...
- Usability
 - Mental walkthroughs
 - Cognitive patterns
 - Experience from visualization (architecture visualization, program understanding)

Target Audiences for DSLs

- Different audiences
 - Software engineers
 - Traditional engineers (control, mechanical,...)
 - Business people (accounting, marketing, management, ...)
 - Analysts (consultants with DS expertise)
 - Scientists (biology, chemistry, physics,...; computational science)
 - Casual computer users (my mom...)
 - ...
- Criteria
 - Level of software development skills
 - Depth of application-domain expertise
- Vastly varying requirements

Target Audiences for DSLs



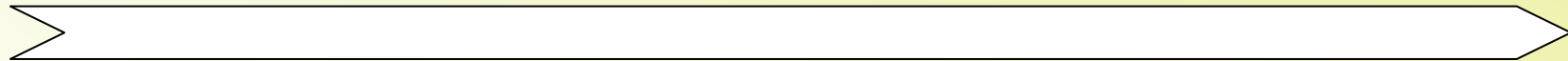
Tools for Implementing DSLs

- To be defined
 - Abstract syntax (e.g., using a class diagram)
 - Rendering and editing
 - Semantics (e.g., by translation)
- Other issues
 - Serialization, language evolution, consistency management and change propagation, etc.
- Tools for DSLs become mainstream...
 - Direct metamodeling
 - Microsoft, Xactium, ..., (EMF)
 - UML profiling
 - IBM Rational Modeller, Softeam, ...

Structure Spectrum of DSLs

Routine configuration

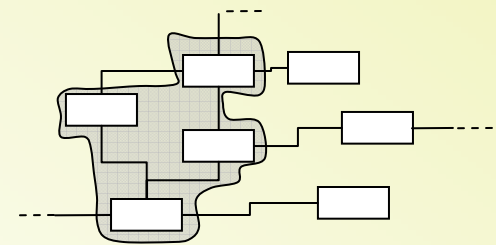
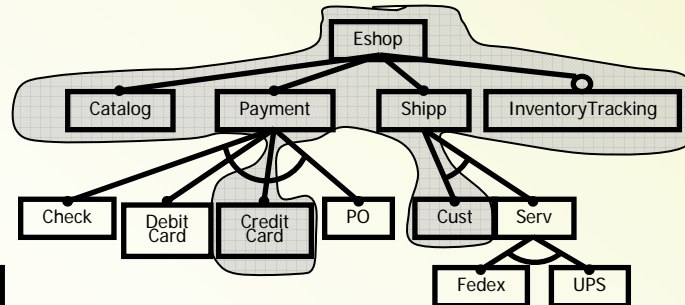
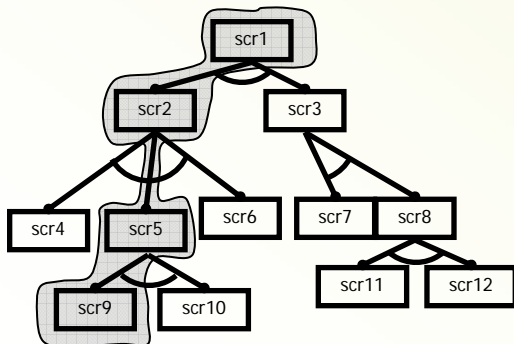
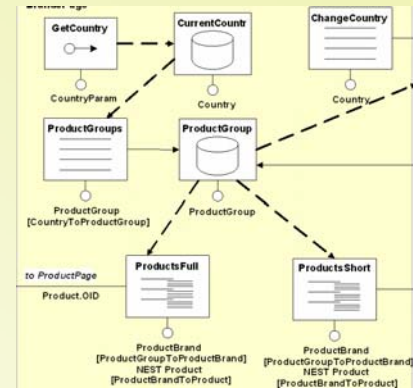
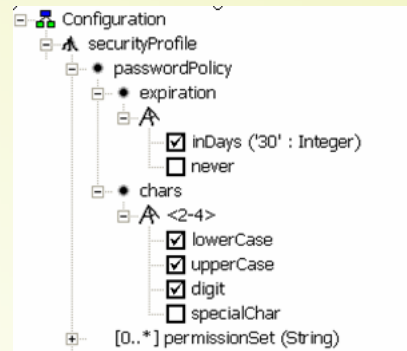
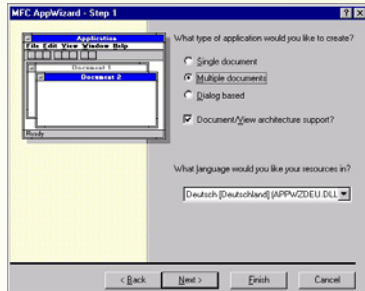
Creative construction



Wizards

Feature-based configuration

Graph-like language (with user-defined elements)



Overview

- Generative Software Development
- Example
- Staged Configuration
- Tool support
- ➔ Summary

Key Concepts in Generative Software Development

- Product lines
 - Cornerstone of systematic software reuse
- Domain-specific languages
 - Optimal support for application developers
- Mappings
 - Design knowledge capture
- Feature modeling
 - Family scoping, DSL & architecture development

Further Information

- Czarnecki & Eisenecker. “Generative Programming: Methods, Tools, and Applications.” Addison-Wesley, 2000
 - <http://www.swen.uwaterloo.ca/~kczarnec/>
- Greenfield & Short. “Software Factories: Assembling Applications With Patterns, Models, Frameworks and Tools.” Wiley, 2004
 - <http://www.softwarefactories.com/>
- 4th Int. Conference on Generative Programming and Component Engineering (GPCE), Sep 29 - Oct 1, 2005, Tallinn, Estonia (co-located with ICFP)
 - <http://gpce.org/>

