Overview of Generative Software Development

Krzysztof Czarnecki
University of Waterloo

Overview

- ➡ Motivation
- Generative Software Development
- Examples
- Summary & Outlook

Scaling Up Software Development

- New requirements and technologies are increasing complexity and rate of change
  - Business process automation, medical informatics, mobile devices, digital media, smart appliances
- Tension between complexity and change
  - Easy to change simple things
  - Easy to build complex things that don't change
- Creating software development problems
  - Low quality, low productivity, legacy issues

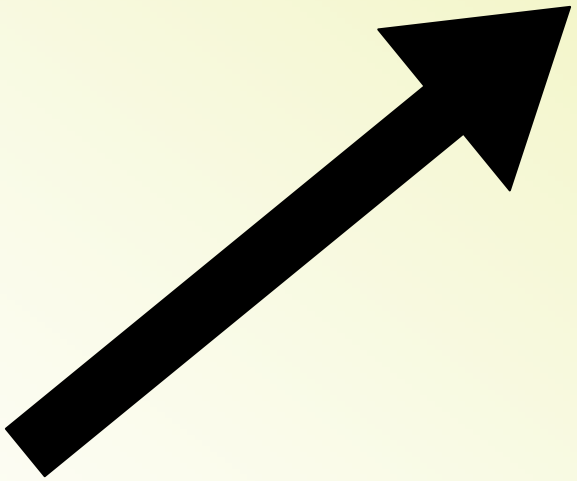
Current Practices Will Not Scale Up

- One off development yields ad hoc reuse
  - Limits capture and reuse of domain knowledge
- Cannot assemble systems from components
  - Architectural mismatch makes reuse difficult
  - Component technologies not mature enough
- Working at too low a level of abstraction
  - Building line by line from fine grained statements

Example

- Object oriented reuse techniques
  - Classes are too small as units of reuse
  - Frameworks are large enough, but frameworks from different vendors do not integrate well
  - Design patterns are pieces of reusable knowledge, but they do not exist as executable code
- Object oriented analysis and design
  - Models not precise enough to be source artifacts – become obsolete as software is cut
  - Generic set of models based on general purpose modeling language doesn't capture enough information
  - Relationships among models not defined well enough to support automation

Scaling Up Requires Transition



Craftsmanship

- Small teams create whole products by hand

Industrialization

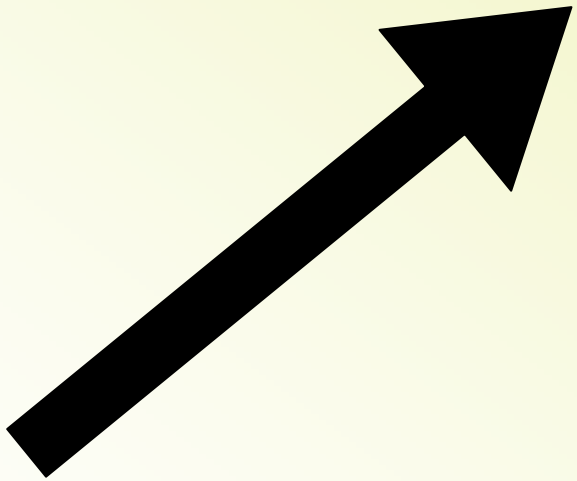
- Integrators use standard processes and tools to automate menial tasks
- Rapidly assemble a wide range of product variants based on standard designs
- From reusable components provided by upstream suppliers in standard packages

Overview

- Motivation
- ➡Generative Software Development
- Examples
- Summary & Outlook

Generative Software Development

Generating the needed components and automatic assembly



Searching for and adapting components and manual assembly

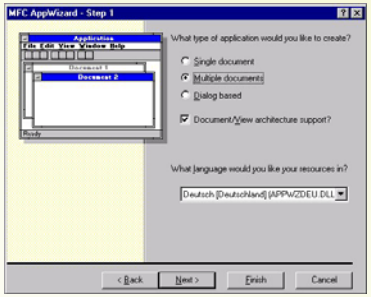
Generative Software Development

- Is a software family approach
- Automates the creation of family members
- Allows generating a family member based on a specification in a *domain-specific language*

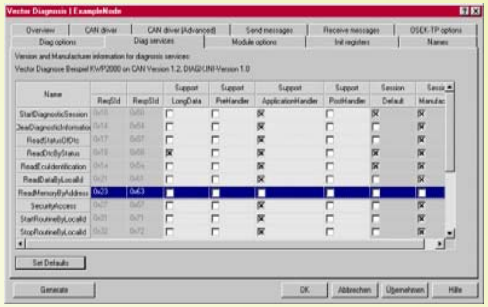
System Family Approach

- **Domain Engineering**
  - **Analysis**: Domain scoping and defining a set of reusable, configurable requirements for the systems in the domain
  - **Design**: Developing a common architecture for the systems in the domain and devising a production plan
  - **Implementation**: Implementing the reusable assets, for example, reusable components, domain-specific languages, generators, a reuse infrastructure, and a production process
- **Application Engineering**
  - Producing concrete systems using the reusable assets developed during Domain Engineering.

Different Forms of DSLs



Wizard



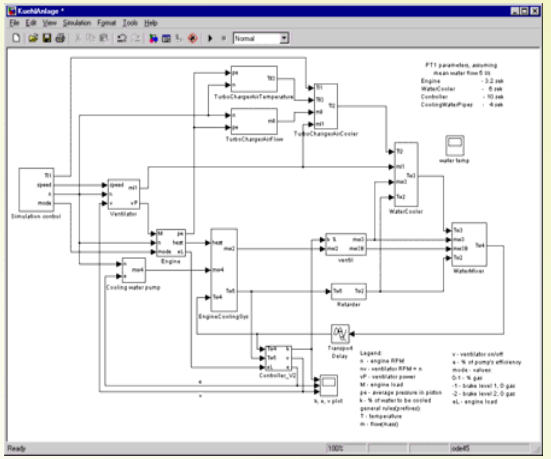
Configuration tool

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

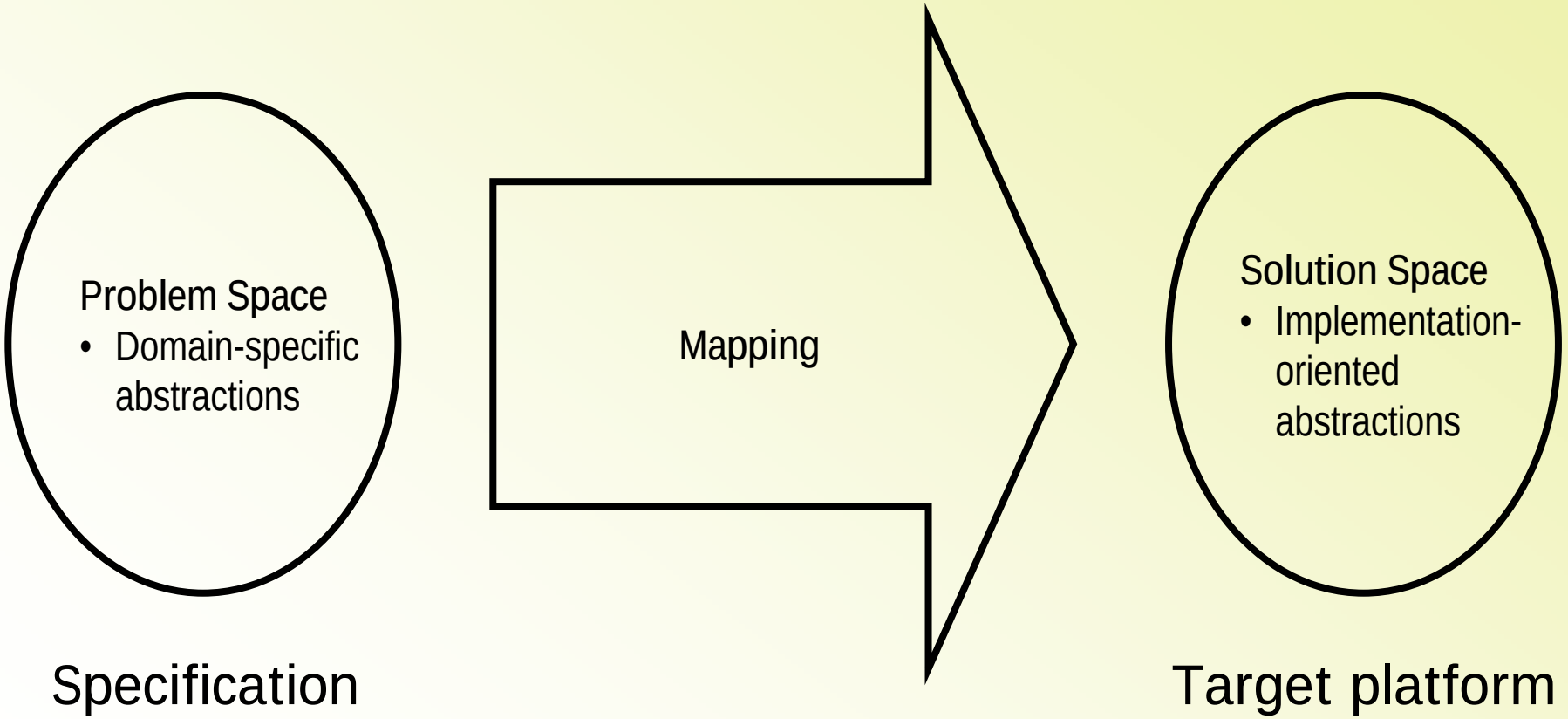
Advantages of DSLs Over General-Purpose Languages (GPLs)

- 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

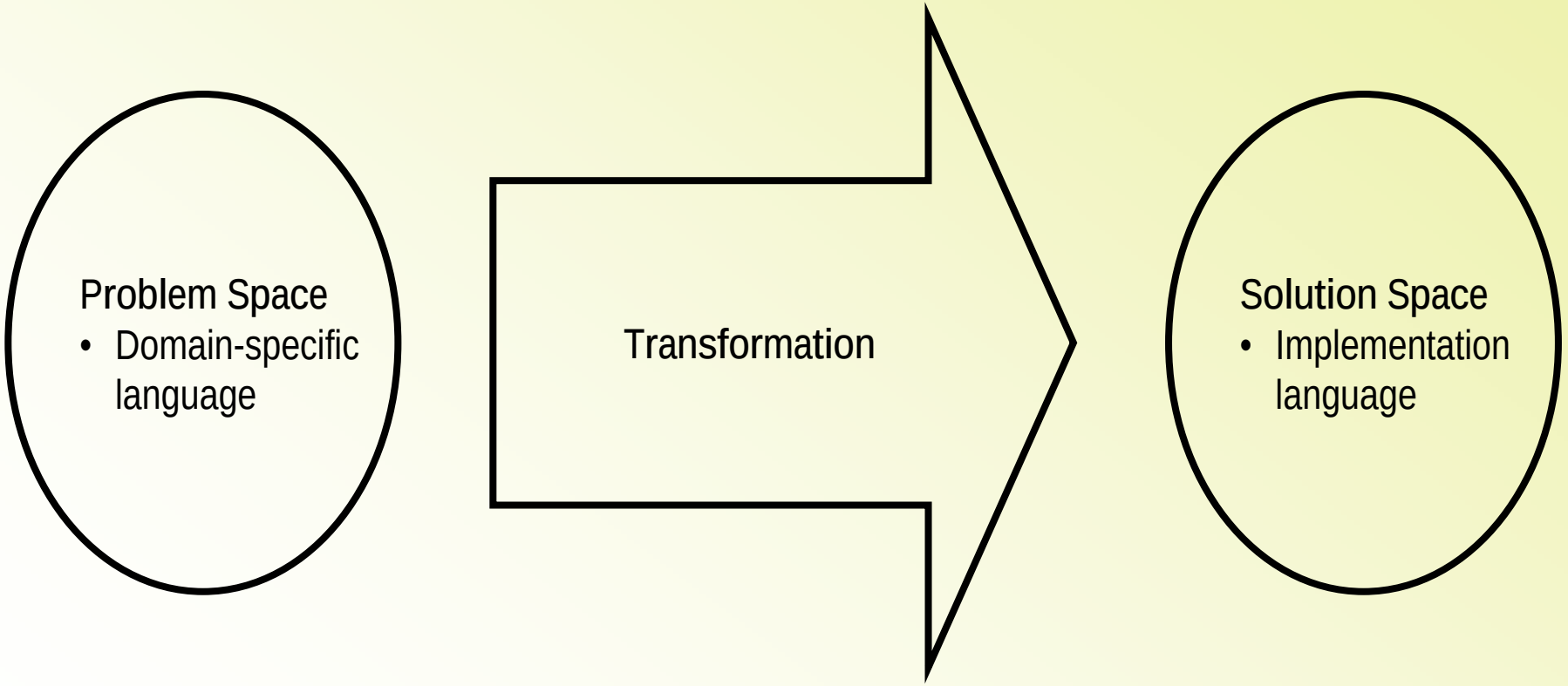
Modeling and DSLs

- 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

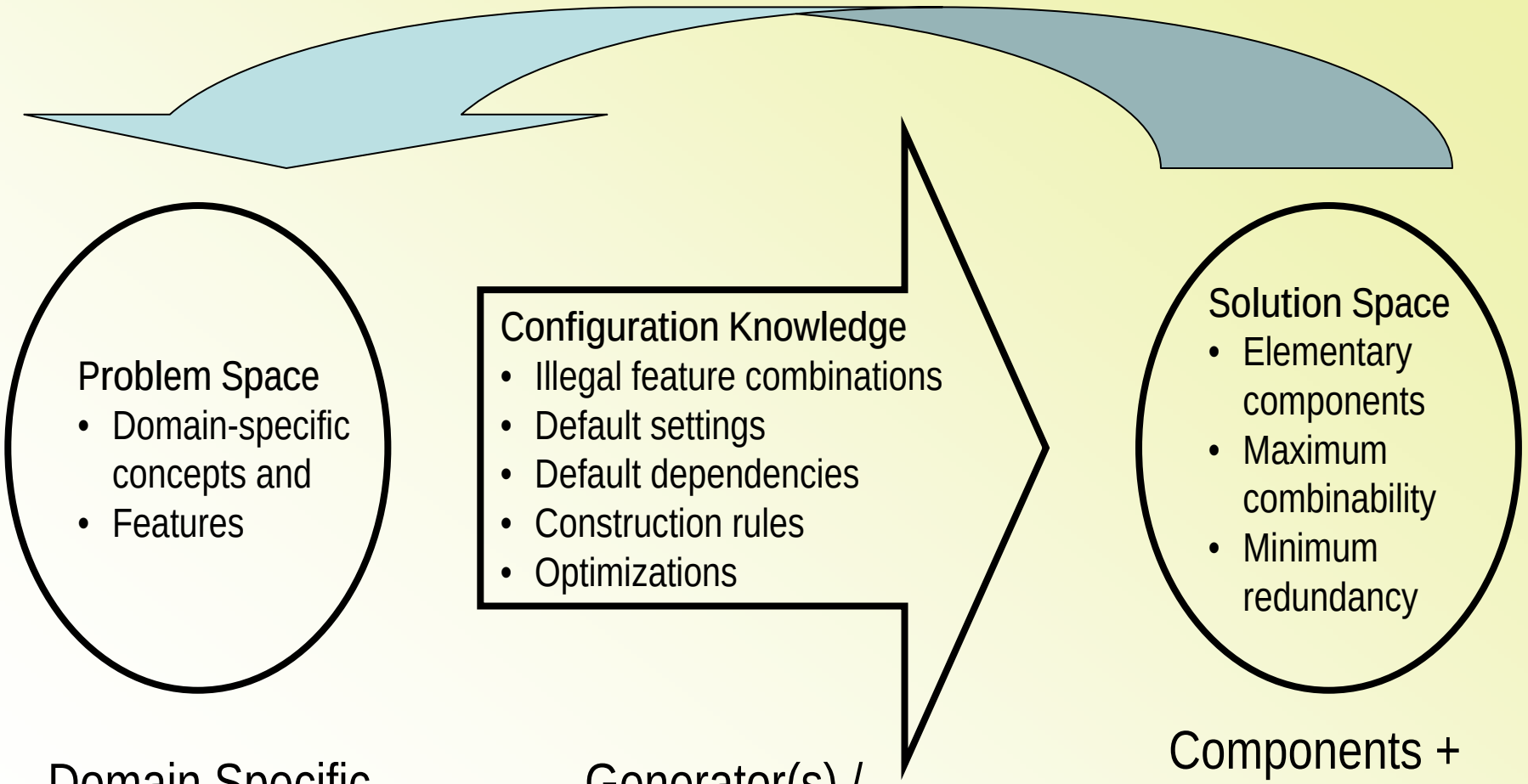
Generative Domain Model



Transformational View



Configuration View



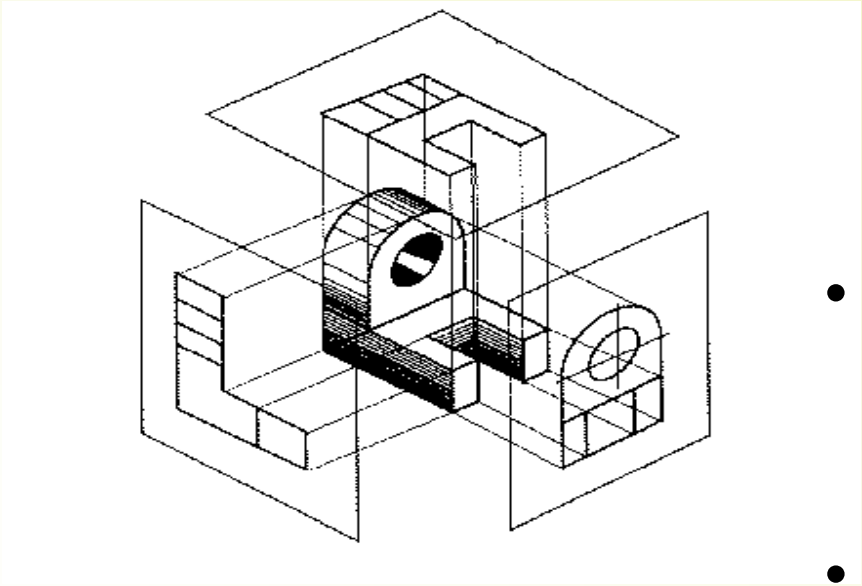
Domain Specific Language(s)

Generator(s) / product configurators

Components + Product-Line Architecture

Also: Reflection, Object Factory, …

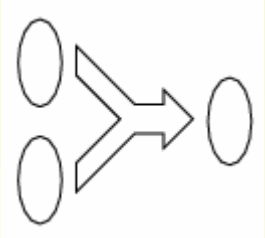
Views And Viewpoints



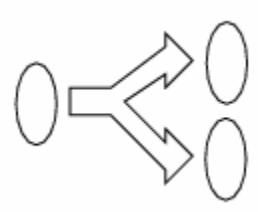
- System models are organized into multiple views
  - Different abstraction levels
  - Different parts of a system
  - Different *aspects*
    - workflow, security, deployment
- Each view conforms to some viewpoint
  - Defines scope, notation, process, validation
- Each viewpoint is relevant to some stakeholder

Mapping Constellations

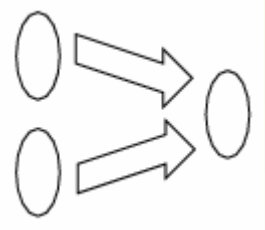
Chaining of mappings



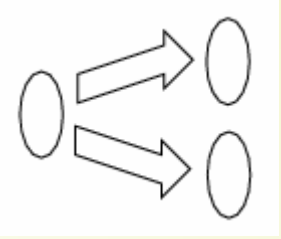
Multiple problem spaces



Multiple solution spaces

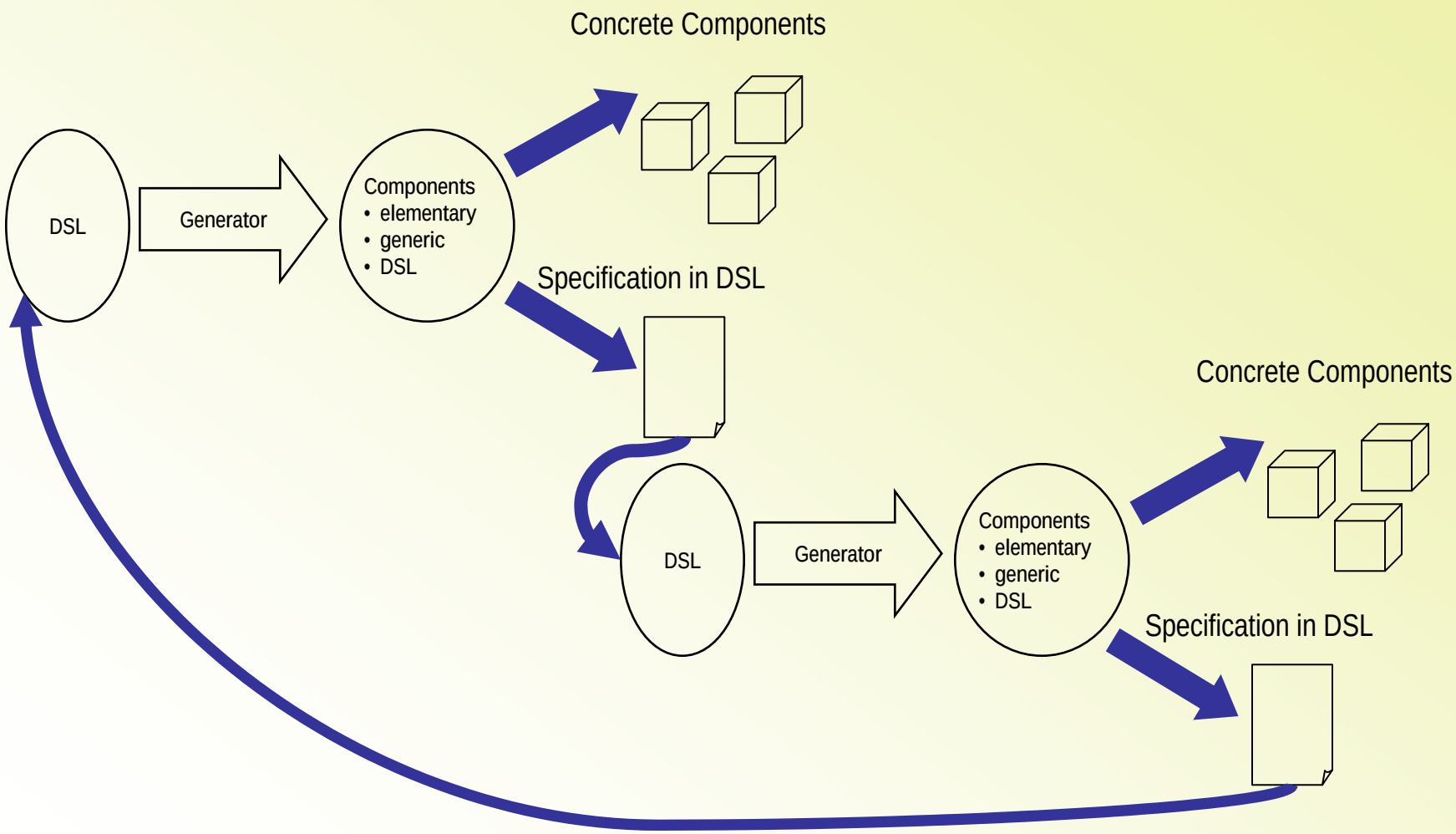


Alternative problem spaces



Alternative solution spaces

Federated Generators



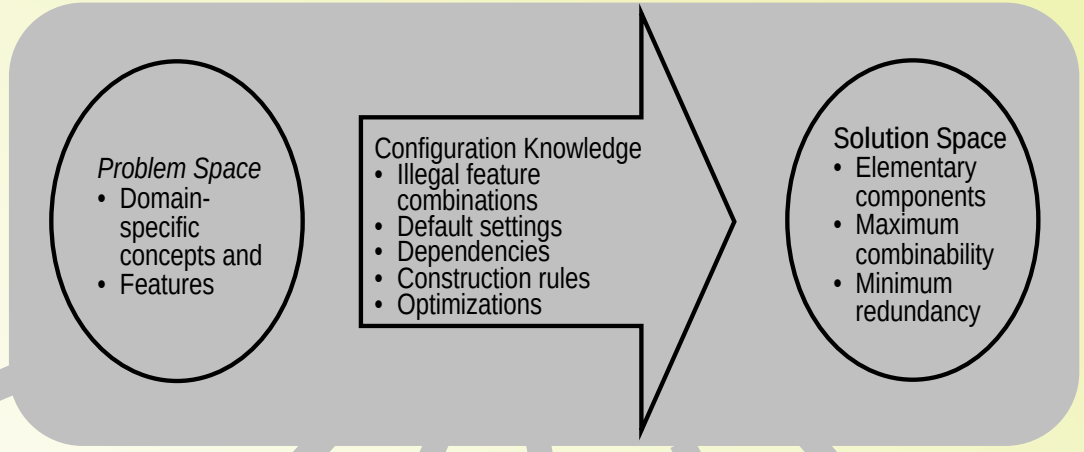
A Technology Projection ...

… is a mapping of the generative domain model onto

- other software development paradigms,
- a programming language,
- several development tools that are combined within one environment or on one platform

- Meanwhile, several technology projections are available

Existing Technology Projections



C++
- ⌘Template meta-programming
- ⌘Static configuration
- ⌘Language-specific means only

Java and JavaBeans
- ⌘Interactive parameterization of JavaBeans
- ⌘Dynamic configuration
- ⌘Language-specific means only

Java und AspectJ
- ⌘Using an AspectWeaver
- ⌘Static configuration
- ⌘Separate preprocessing step

ANGIE
- ⌘Employment of a frame processor
- ⌘Static configuration
- ⌘Separate preprocessing step

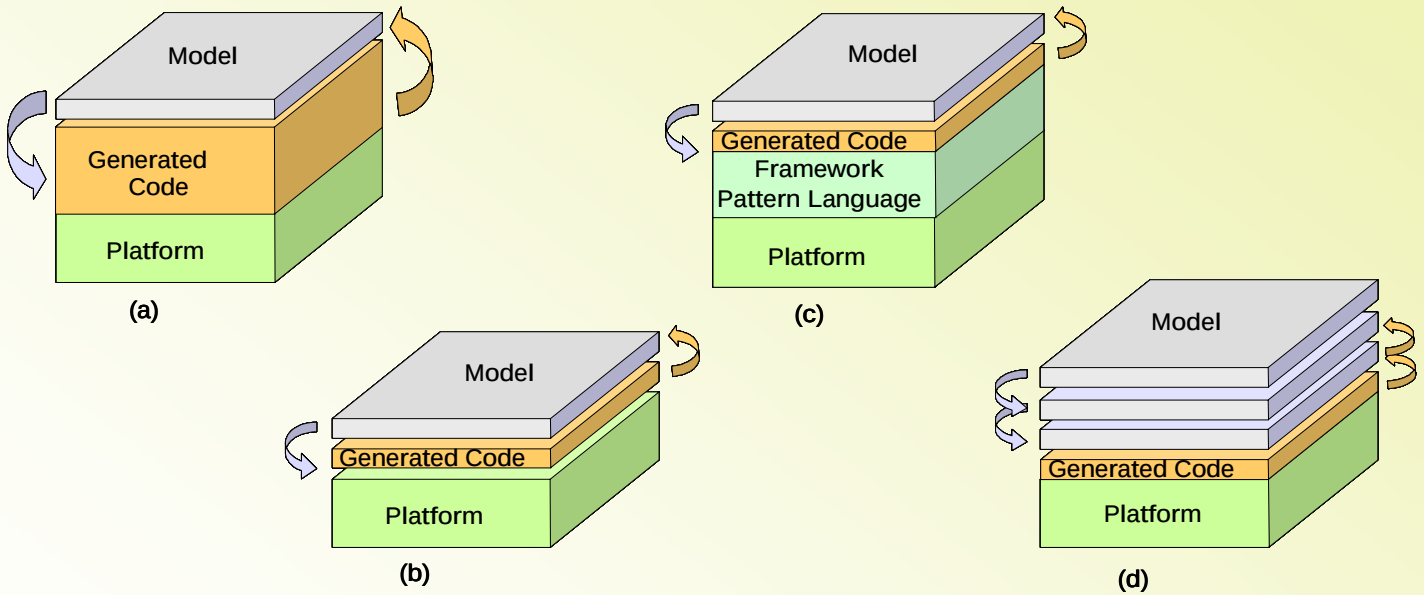
Small Components
- ⌘Usage of an XML configuration file
- ⌘Static and dynamic configuration
- ⌘Separate preprocessing step

XML, TL and Ada 83
- ⌘Graphically-interactive configuration
- ⌘Static configuration
- ⌘Separate preprocessing step

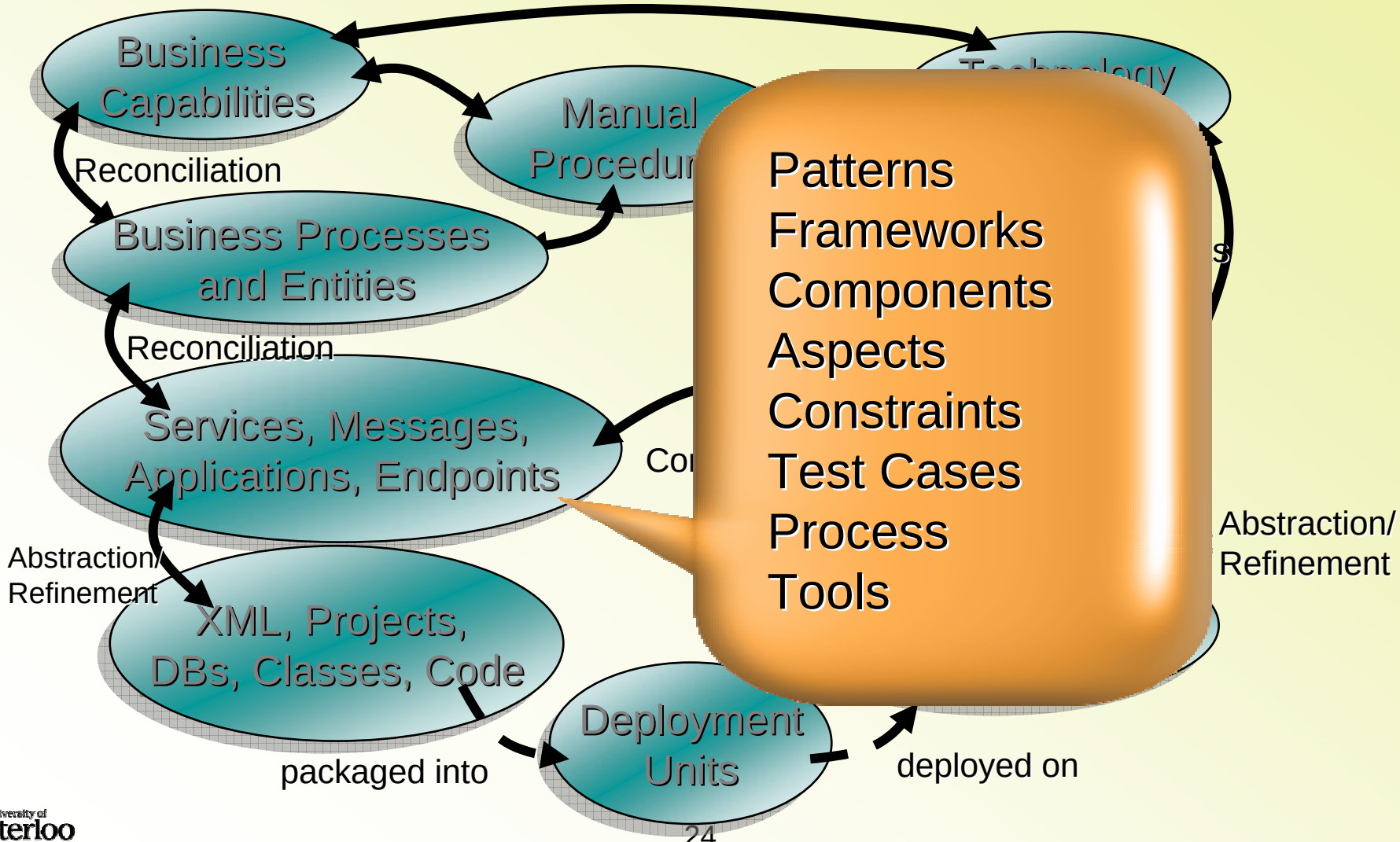
Many Mapping Technologies to Chose From…

- Template-based code generation
  - TL, Velocity, XVCL, Angie, …
- Metaprogramming
  - C++ Template Metaprogramming, Template Haskell, …
- Transformation systems
  - DMS, Stratego/XT, TXL, …
- Program specialization
  - Tempo, …
- Model transformation
  - ATL, UMLAUT, …
- Product configurators
- …

Generating Code From Domain Specific Languages



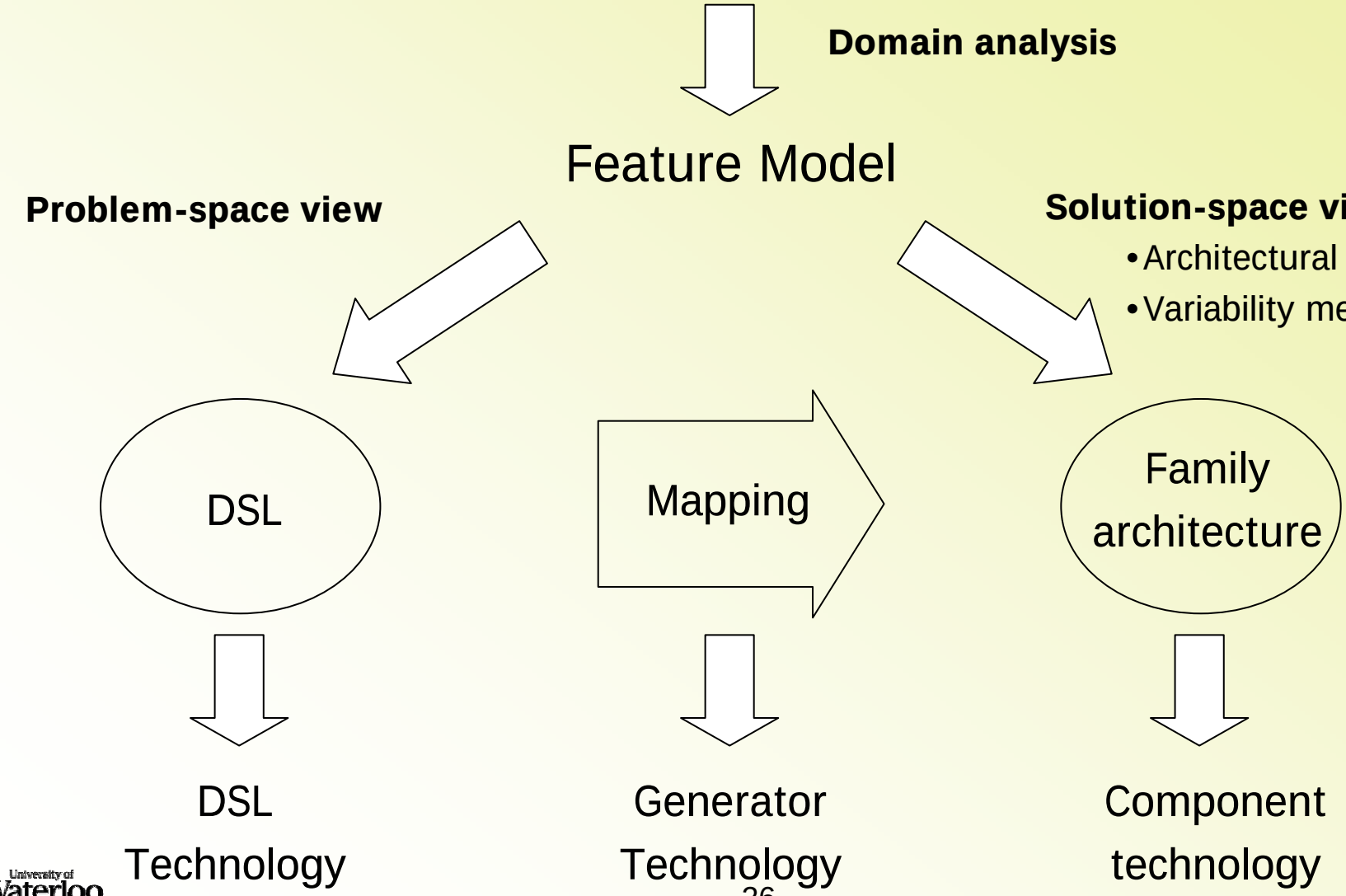
Software Factories (Greenfield et al.)



Software Factory Schema

- This is called a software factory schema
  - Like a recipe for a specific type of application
  - A set of viewpoints related by mappings that support transformation, validation and traceability
  - Lists artifacts required to build that type of application and explains how to combine them
- A software factory template is content
  - Configures a development environment to build that type of application
  - Projects, patterns, frameworks, guidance
- The configured development environment is the software factory
  - Integrates tools, process and content for that type of application
  - Domain specific editing, rendering, compilation, debugging, refactoring

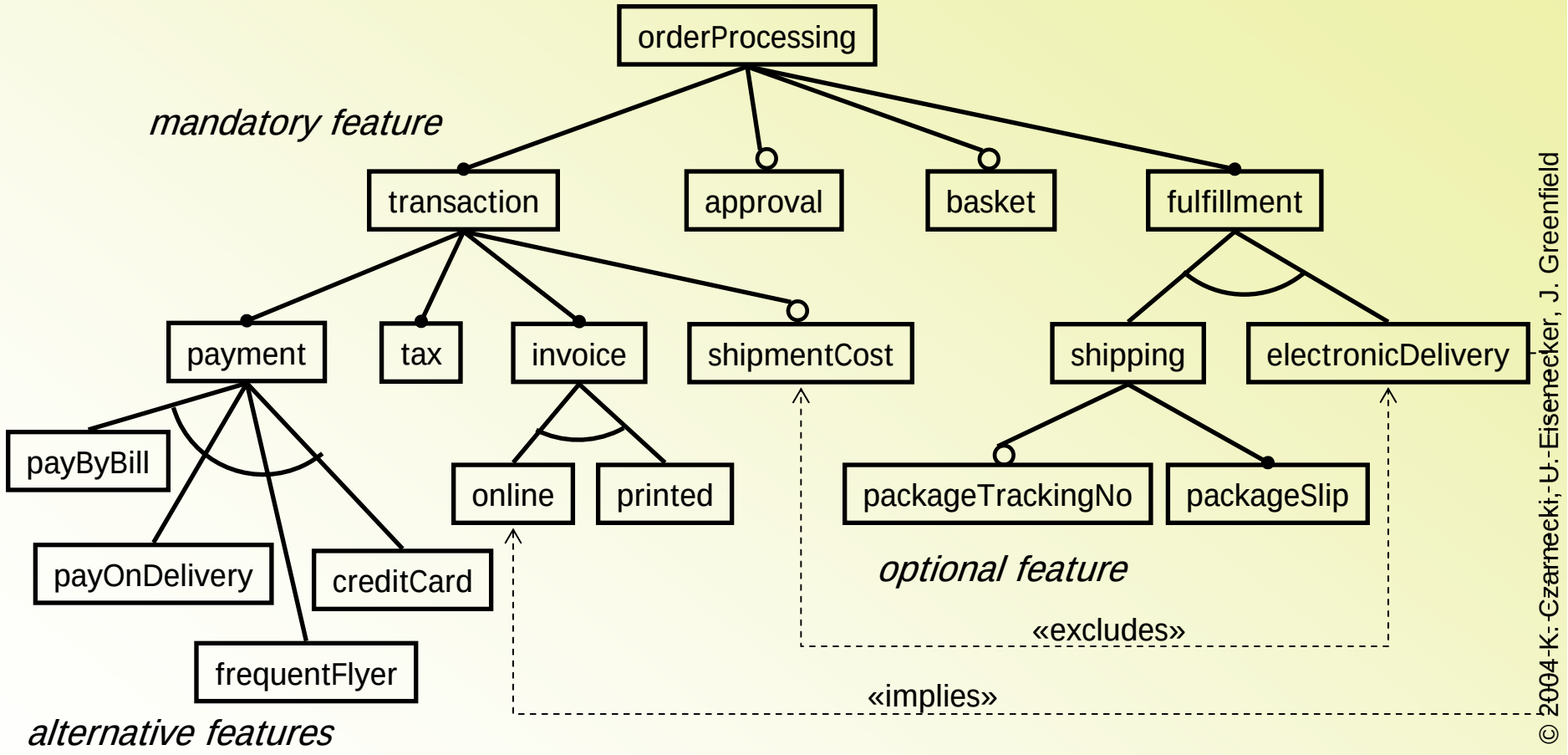
Feature-Oriented Approach



Overview

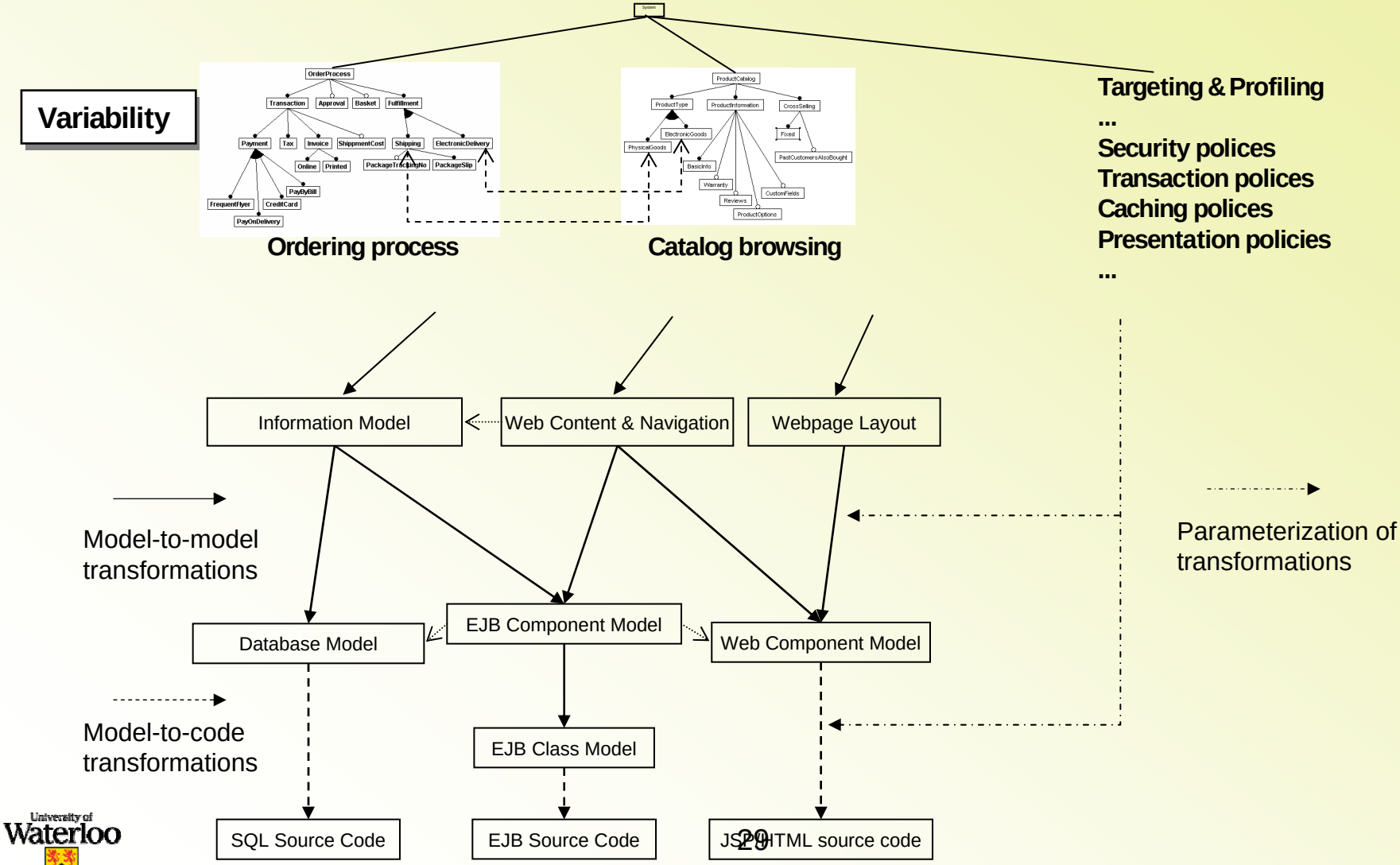
- Motivation
- Generative Software Development
- ➡Examples
- Summary & Outlook

Feature Diagram

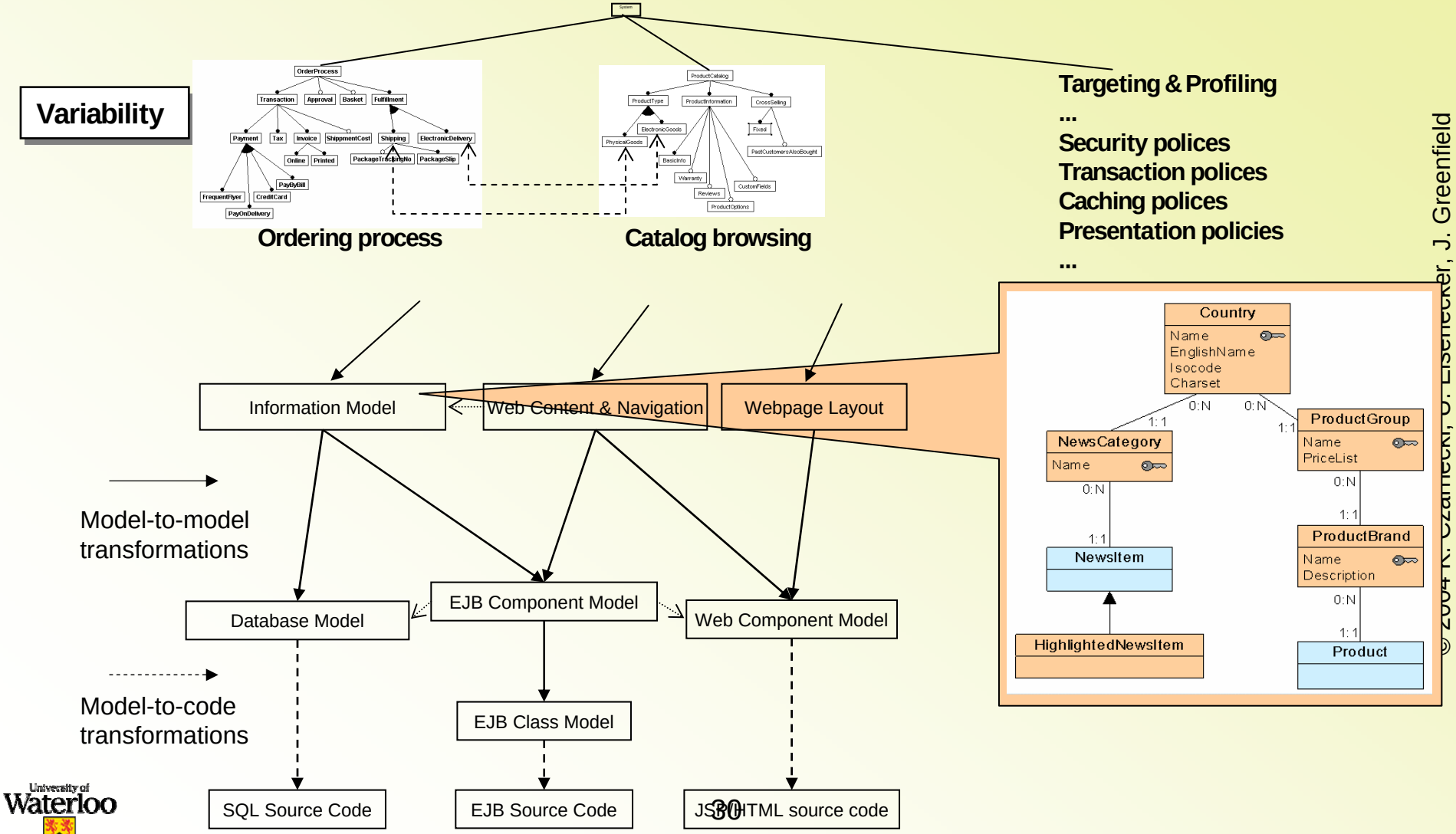


No decision regarding the mechanism for implementing variability!

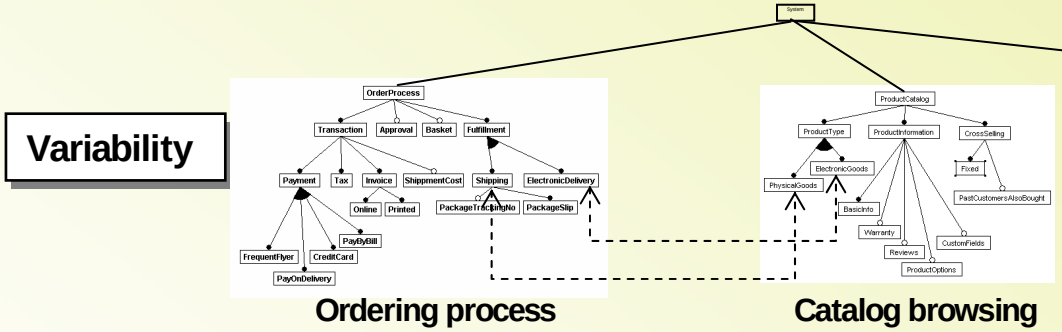
Mapping Feature Variations To Software



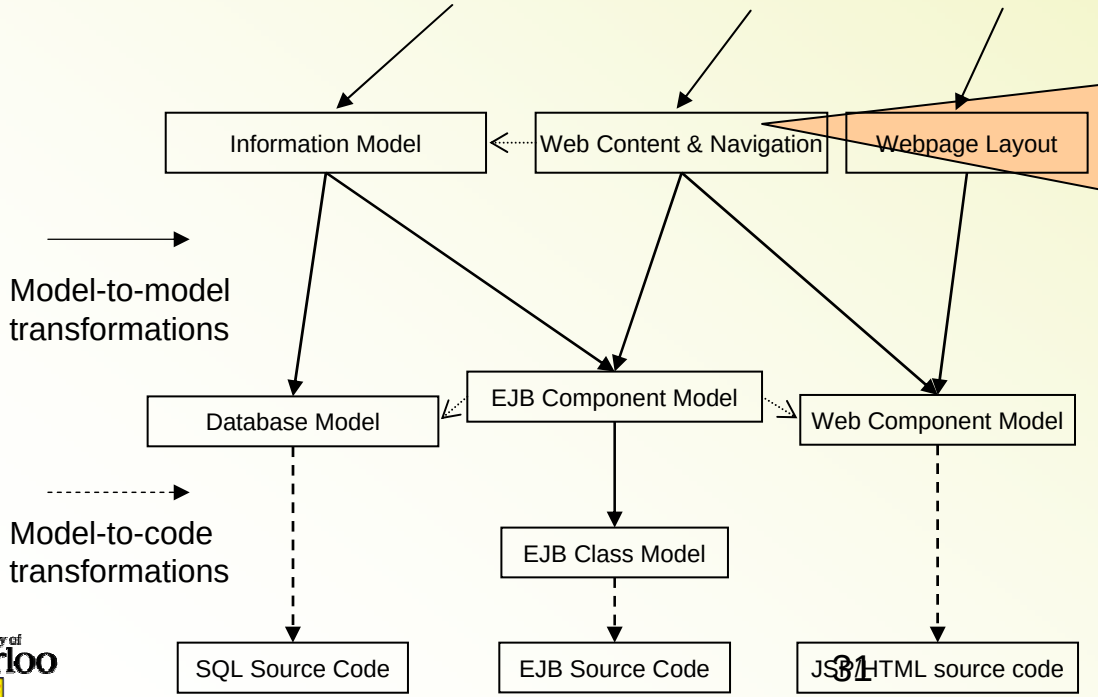
Mapping Feature Variations To Software

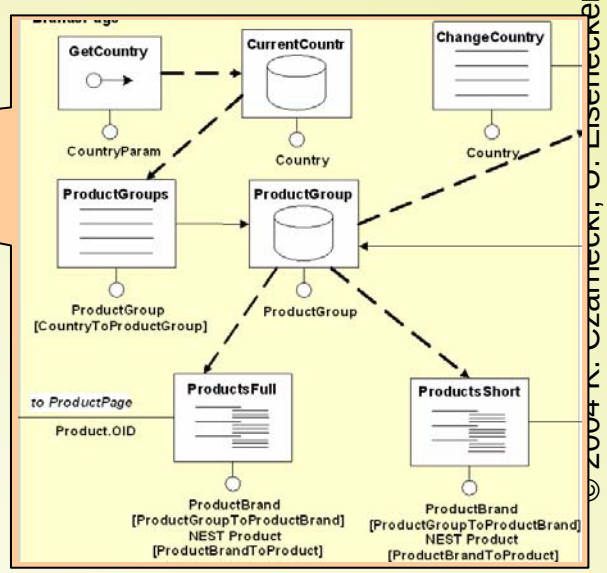


Mapping Feature Variations To Software



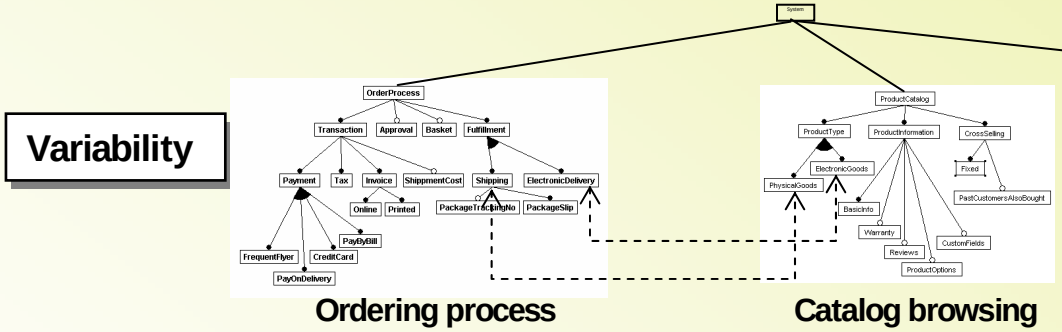
Targeting & Profiling
...
Security polices
Transaction polices
Caching polices
Presentation policies
...

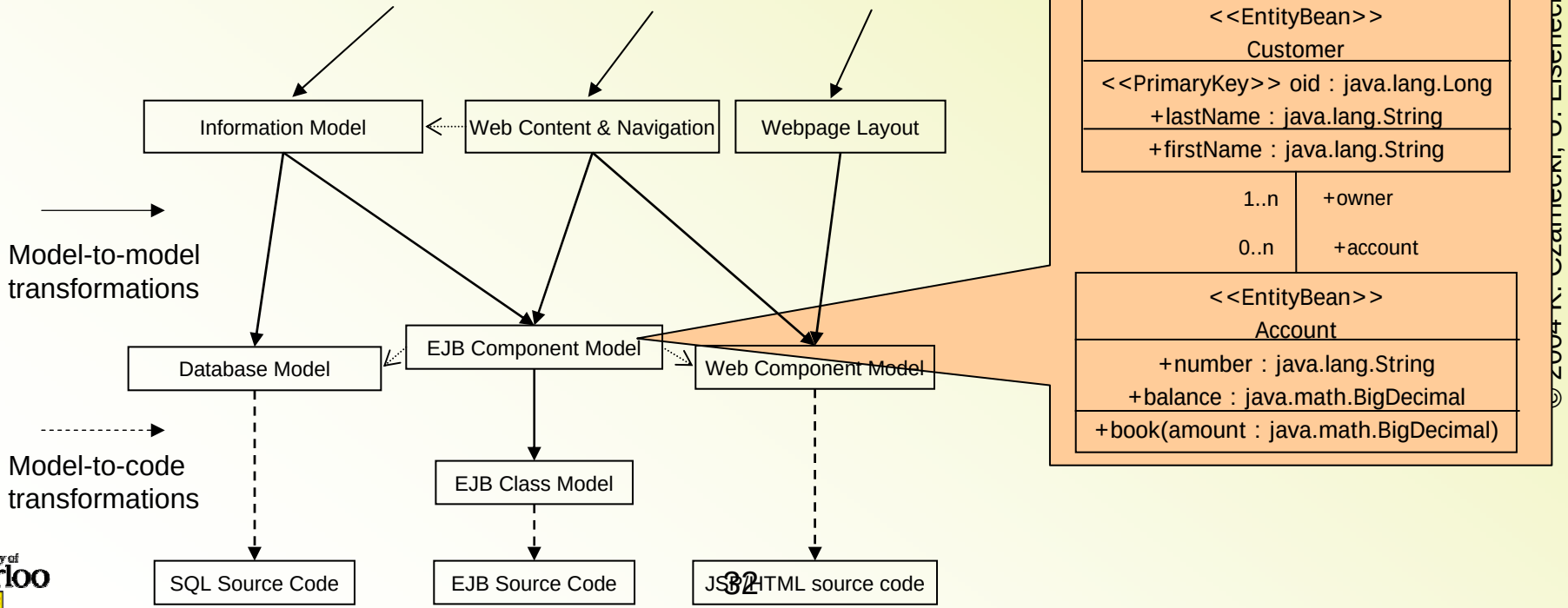




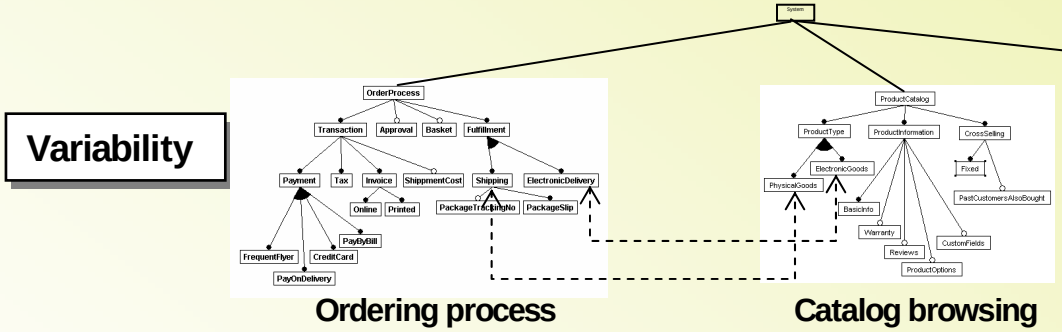
See webml.org

Mapping Feature Variations To Software

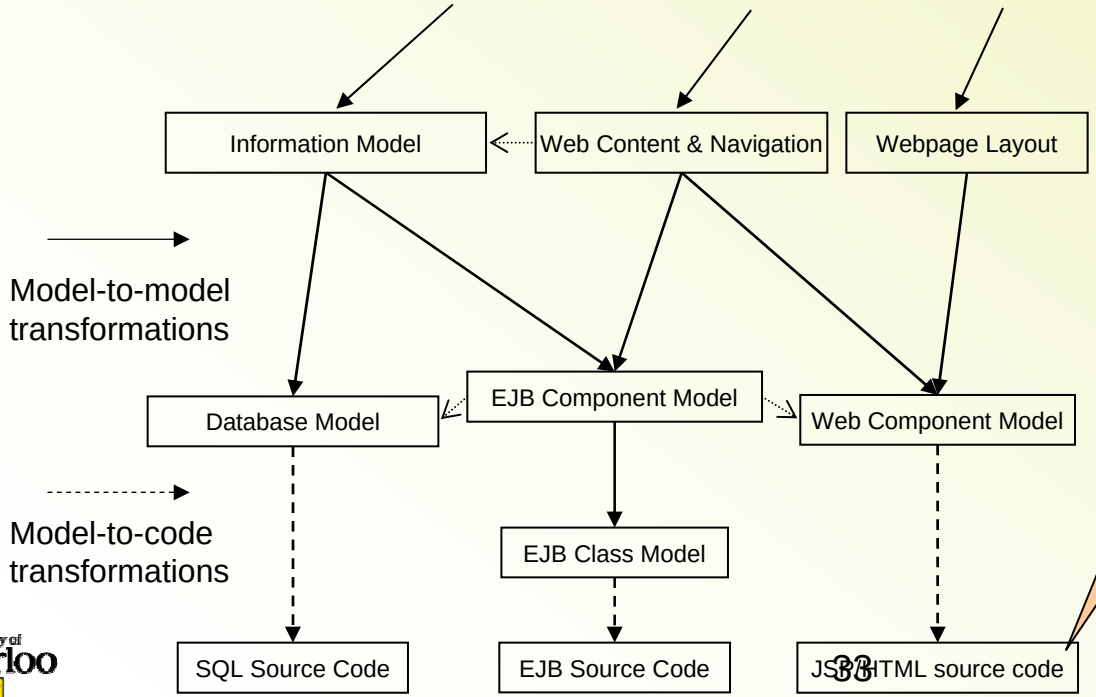


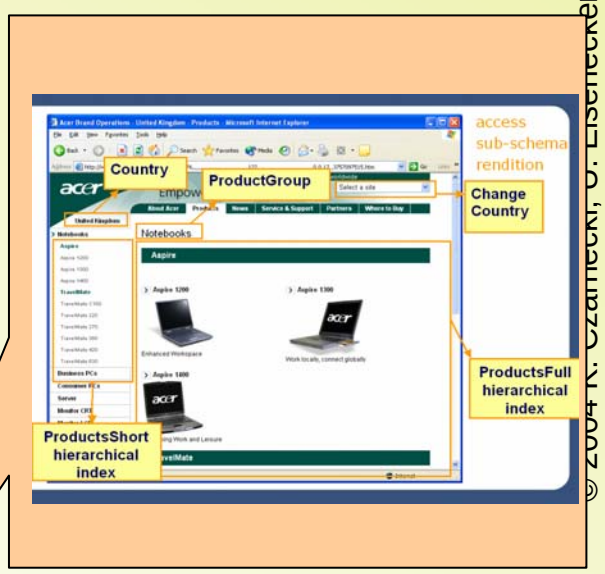


Mapping Feature Variations To Software

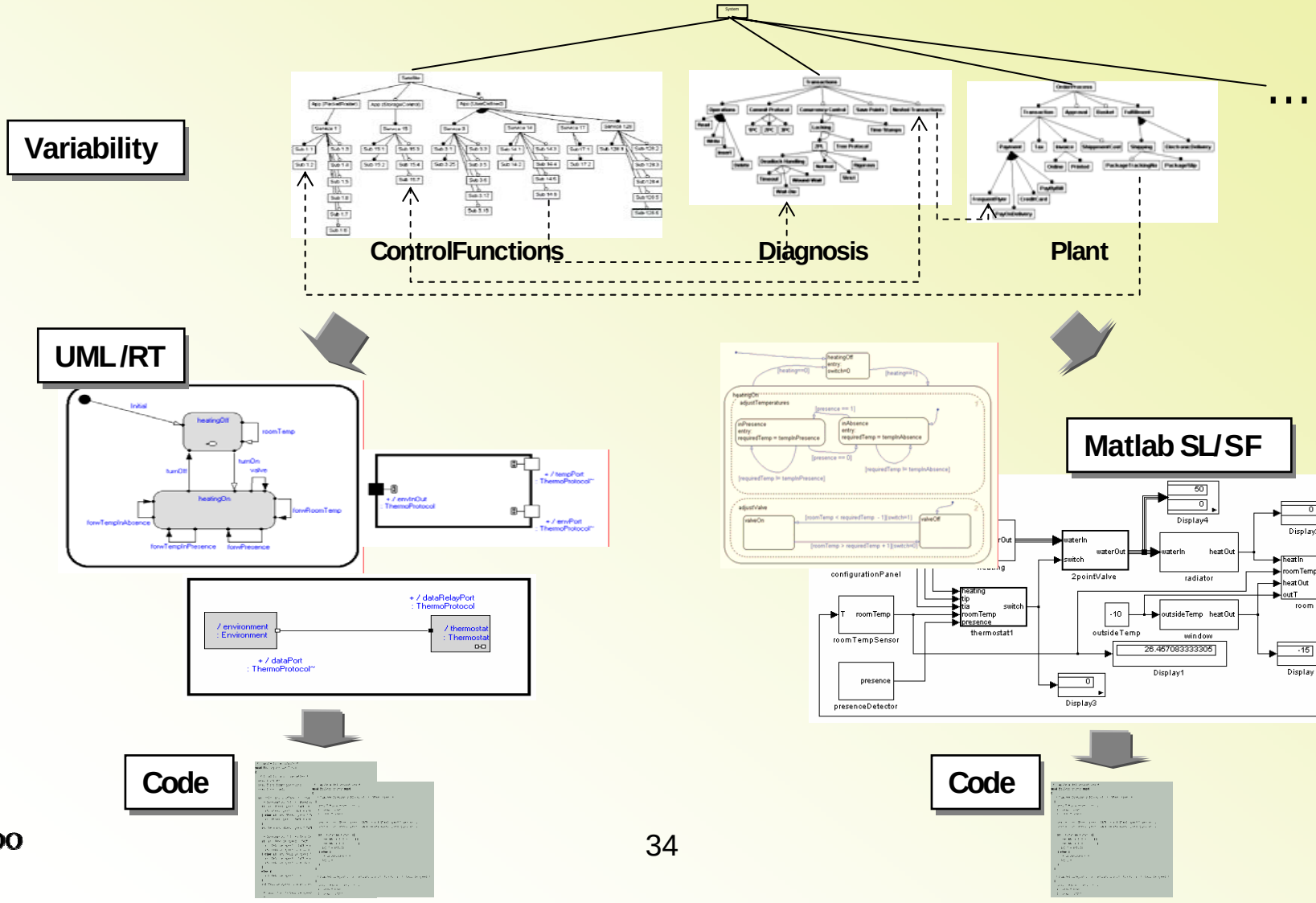


Targeting & Profiling
...
Security polices
Transaction polices
Caching polices
Presentation policies
...



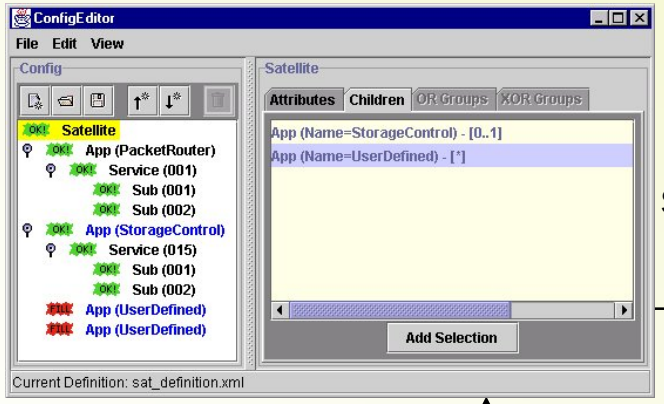


Configuration and Generation For Embedded Systems



Feature-Based Configuration of Satellite Software

General-purpose ConfigEditor



save

Satellite configuration in XML

Statically configure

Family Architecture of Satellite Com Systems (Ada83 templatized using TL)

Generate

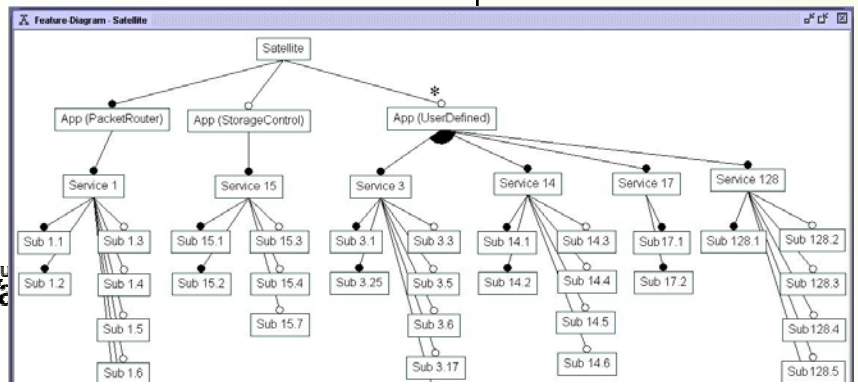
Concrete Satellite

Communi-cation

Ground station

Dynamically configure

DSL definition (FM in XML format)



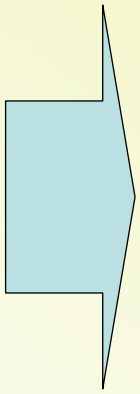
Generation Using Template Language

```
#for i "/Satellite/PUS/Service/Sub"#
with service#"$i/../id"#_#"$i/id"#;
#end#

separate (app)
  procedure decode(o :in out ptr ; p :in ...) is
    no_service : exception;
    ...
  begin
    for i in o.service'range loop
      ...
    end loop;
    ...

    case s.typ is
#for i "/Satellite/PUS/Service"#
    when #"$i/id"# =>
      case s.sub is
      #for j "$i/Sub"#
        when #"$j/id"# => service#"$j/../id"#_#"$j/id"#(o, p);
      #end#
      when others => null;
      end case;
#end#
      when others => null;
    end case;

  exception
    ...
end decode;
```

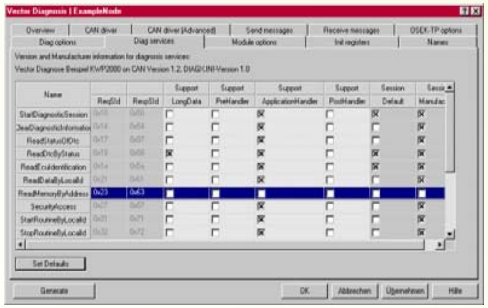


```
with service014_001;
with service014_002;
with service001_001;
with service001_002;
with service001_007;
with service001_009;
with service004_023;
with service004_024;
with service004_025;
with service004_026;
separate (app)
  procedure decode(o :in out ptr ; p :in
...) is
    no_service : exception;
    ....
  begin
    for i in o.service'range loop
      ...
    end loop;
    ...
    case s.typ is
      when 014 =>
        case s.sub is
          when 001 => service014_001(o, p);
          when 002 => service014_002(o, p);
          when others => null;
        end case;
      when 001 =>
        case s.sub is
          when 001 => service001_001(o, p);
          when 002 => service001_002(o, p);
          when 007 => service001_007(o, p);
          when 009 => service001_009(o, p);
          when others => null;
        end case;
      when 004 =>
        case s.sub is
          when 023 => service004_023(o, p);
          when 024 => service004_024(o, p);
          when 025 => service004_025(o, p);
          when 026 => service004_026(o, p);
          when others => null;
        end case;
      when others => null;
  end case;
exception
  ...
end decode;
```

Model-Based Development of Automotive Embedded Software

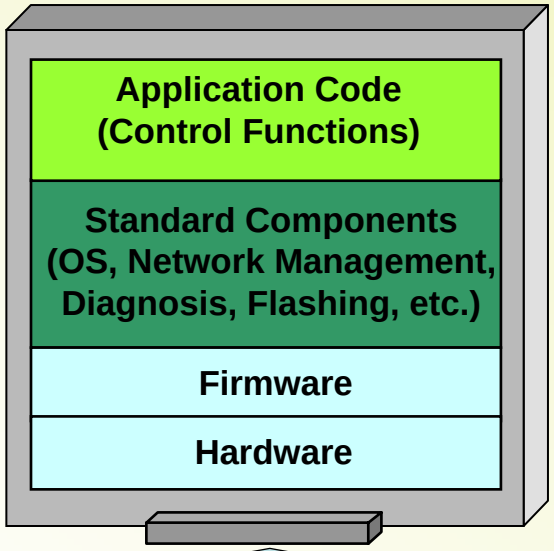
Standard Components
- Configuration using a GUI-based editor
- Template-based code generation

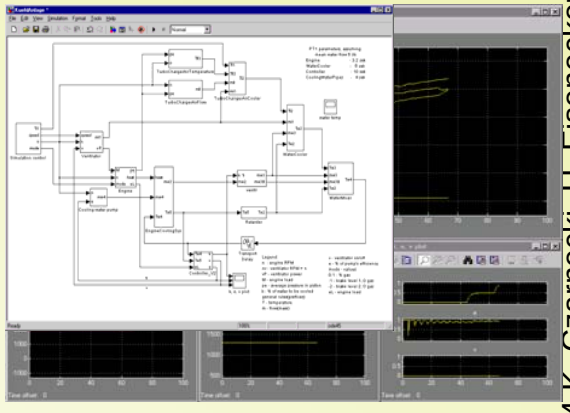
Control Functions
- Modeling and Simulation (Matlab/Simulink/Stateflow)
- Production code generation (TargetLink, Real Time Workshop)



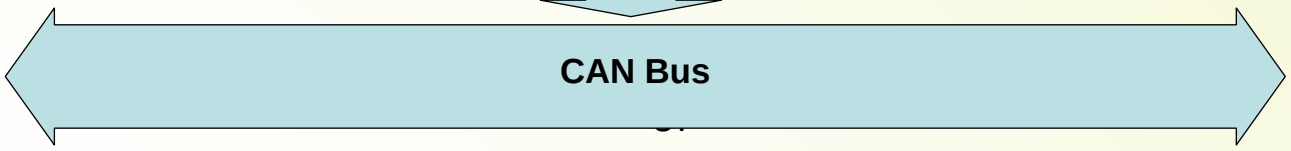
Static configuration

Electronic Control Unit





Dynamic configuration (Calibration)

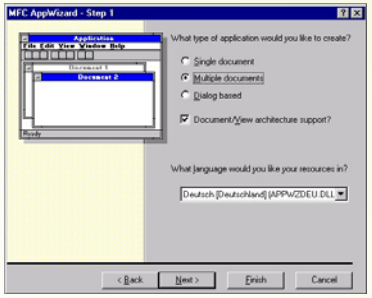


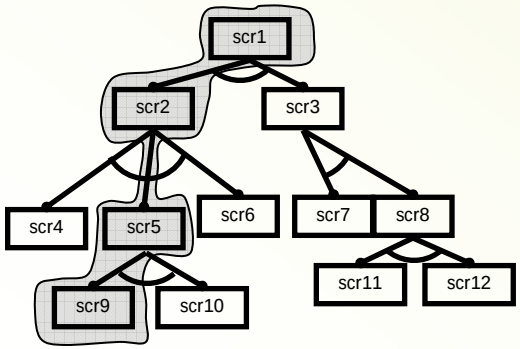
University of Waterloo

Structure Spectrum of DSLs

Routine configuration → Creative construction

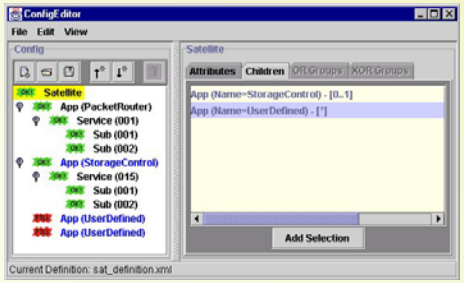
Wizards

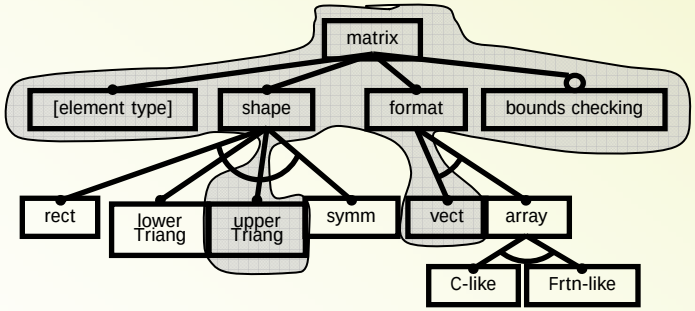




Path through a decision tree

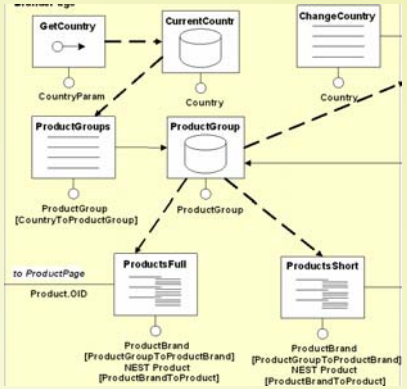
Feature-based configuration

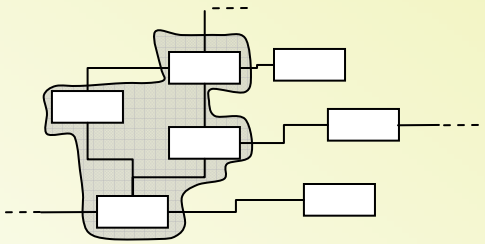




Subtree of a feature tree

Graph-like language
(with user-defined elements)





Subgraph of an (infinite) graph

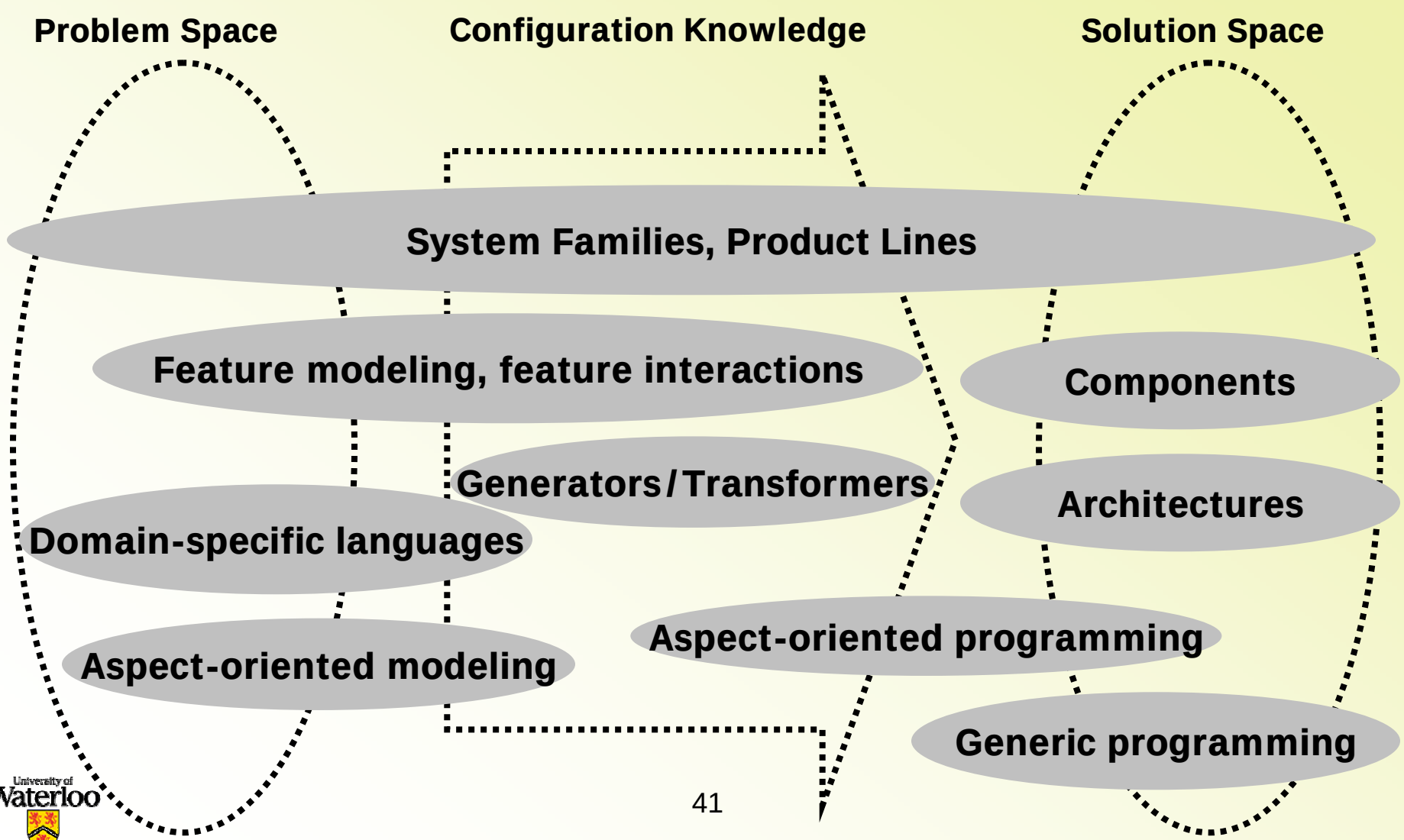
Overview

- Motivation
- Generative Software Development
- Examples
- ➡Summary & Outlook

Key Concepts in Generative Software Development

- Software system families
  - Cornerstone of systematic software reuse
- Domain-specific languages
  - Optimal support for application developers
- Mappings
  - Design knowledge capture
- Aspect-oriented development
  - Better separation of concerns & composition mechanisms
- Feature modeling
  - Family scoping, DSL & architecture development

Relation To Other Fields



Hot Topics in Generative Software Developments

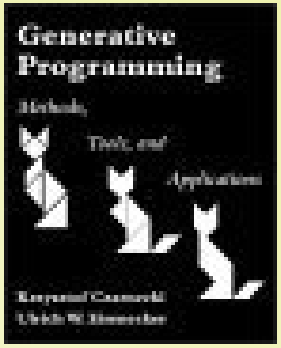
- Composition of DSLs
- Consistency management between views and reconciliation
- Systematic development of DSLs
- Tool support for DSL development
- Model editor generation

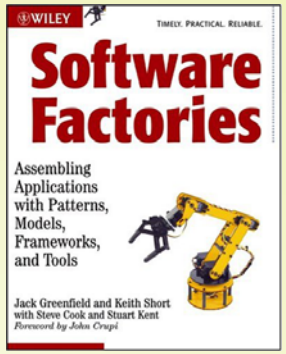
Generic vs. Generative

- Generic
  - *"relating to or characteristic of a whole group or class"* (Merriam-Webster Online)
  - Solution space technique for developing parametrized components
- Generative
  - *"having the power or function of generating, originating, producing, or reproducing"* (Merriam-Webster Online)
  - System for producing other systems; it comprises problem space, configuration knowledge, and solution space

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/
- 3rd Int. Conference on Generative Programming and Component Engineering (GPCE), October 24 -28, 2003, Vancouver, (co-located with OOPSLA) http://gpce.org/





Questions…