Introduction to Generative Software Development

Krzysztof Czarnecki
University of Waterloo
czarnecki@acm.org

www.generative-programming.org
Goals

• What is to be achieved?
  • Basic understanding of Generative Programming (GP)
  • Based on first impressions and discussion you should be able to judge what you can benefit from GP

• How can it be achieved?
  • Overview of the most important steps of GP
  • Demonstration of tools
  • Presentation of case studies
  • Discussion in the plenum
Overview

➢ Challenges
  • Introduction
  • First Example
  • Further Information
Complexity

• Software systems are engineering in new and demanding application domains
• Software systems support increasingly complex tasks
• Requirements concerning functionality and efficiency are constantly growing
Quality

- Software is deployed in life-critical application areas
- Software malfunction causes economic losses
- Usability is a major acceptance criterion
- Users expect a specific quality that is determined by varying criteria
Productivity

- Growing demand for new systems and variants
- Increase of productivity and development staff cannot close the gap between demand and supply
- Standard software becomes more and more important, but it cannot be adapted to every specific requirement
Maintenance & Legacy Systems

• Long lifetime of software systems (sometimes 20 years and more)
• Software must be continuously adapted to new and changing requirements
• Maintenance of legacy systems absorbs development capacity to a high degree
• Software systems of today are the legacy of tomorrow!
Example "Object Technology"

- **Theory**
  - Productivity and quality should increase thanks to reuse

- **Reality**
  - Classes are too small as units of reuse
  - Frameworks are sufficiently large as units of reuse, but frameworks from different vendors do not integrate well
  - Design patterns are pieces of reusable knowledge, but they do not exist as executable code
Example "Component Technology"

- Standards and platforms improve interoperability and the distributed execution of components
- Many components provide a special mode for customization
- Reusing small components does not have a large impact on software development while large components require high customization efforts
One of the Causes ...

Currently dominating methods and tools are designed for developing *single systems*
Overview

• Challenges

➢ Introduction

• First Example

• Further Information
Single System Development

• Focus on analyzing, modeling, and implementing a single system
• Important goal: decomposition into components, i.e. modularization
  • Individual components are easier to maintain, to replace, and to reuse
• But, what are the criteria for decomposition?
Experiment - Part I
Experiment - Part II
Exemplar Systems of a Family
The "Right" Modules
An Additional System ...
... Can Cause Surprise!
How To Order Systems?

• By Components
  • Downward arrow on top
  • Large triangle below with its 90° angle at the lower left point

• By Specification
  • A cat
  • Sitting upright
  • Waiving its tail
From Order To Product

Creating the order
• Expert language
• Form, catalogue
• By example

Processing the order
• Completion
• Validation
• Construction rules
• Optimization

Product generation
• Assembling elementary, reusable components
Generative Software Development...

• is a software family approach
• automates the creation of family members
• generate a family member based on a specification in a *domain-specific language*
Domain-Specific Languages (DSLs)

• Scoped and designed for a specific purpose
  • Domain-specific abstractions
  • Domain-specific concrete syntax
  • Domain-specific error checking
  • Domain-specific optimizations
  • Domain-specific debugging, version control, etc.

• Many forms possible
  • Text, diagrams, forms, tables, wizards, conventional library, etc.
Different Forms of DSLs

- **Wizard**
- **Configuration tool**
- **Library in a programming language**

- **Textual DSL**

```
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() }
    }
}
```

- **Visual DSL**
Two Processes

• 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.
System Family Approach

Domain Engineering

Domain Knowledge

Domain Analysis

Domain Design

Architecture & Production Plan

Domain Implementation

Application Engineering

Customer Requirements

Requirements Analysis

Product Features

Product Configuration

Components, DSLs, Generators, Infrastructure

Integration and Test

Product

Iterative Process

New Requirements
Maturity Levels in Domain Engineering

- Domain Engineering can be applied at different levels
  - Domain analysis
  - Reference architectures and architectural patterns
  - Design patterns and OO frameworks
  - Components and component frameworks
  - Generation of parts of application code
- Generative Programming aims at the highest level of automation of application engineering
  - Domain-specific languages and automatic configuration of components
Generative Domain Model

Problem Space
- Domain-specific abstractions

Solution Space
- Implementation-oriented abstractions

Mapping

Specification

Target platform
Transformational View

Problem Space
- Domain-specific language

Transformation

Solution Space
- Implementation language
Configuration View

Problem Space
- Domain-specific concepts and
- Features

Configuration Knowledge
- Illegal feature combinations
- Default settings
- Default dependencies
- Construction rules
- Optimizations

Generator(s) / product configurators
Also: Reflection, Object Factory, ...

Solution Space
- Elementary components
- Maximum combinability
- Minimum redundancy

Components + Product-Line Architecture

Domain Specific Language(s)
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
Each subdomain is implemented as a generative domain model
Higher order components
Federated Generators

Concrete Components
Specification in DSL
Components
- elementary
- generic
- DSL

Concrete Components
Specification in DSL
Components
- elementary
- generic
- DSL

DSL
Generator

Components
- elementary
- generic
- DSL
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

Problem Space
- Domain-specific concepts and features

Configuration Knowledge
- Illegal feature combinations
- Default settings
- Dependencies
- Construction rules
- Optimizations

Solution Space
- Elementary components
- Maximum combinability
- Minimum redundancy

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

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

Small Components
- Usage of an XML configuration file
- Static and dynamic 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

(a) Model
   Generated Code
   Platform

(b) Model
   Generated Code
   Platform

(c) Model
   Generated Code
   Framework
   Pattern Language
   Platform

(d) Model
   Generated Code
   Platform
In this presentation

- Domain Analysis is part of Domain Engineering
- A product family comprises the variants of a product
- A product line comprises multiple products that are targeted for a specific group of customers
- A product line can be based on system families

Different uses

- Domain Analysis precedes Domain Engineering
- Product family and product line are synonyms
Essential Terms

• Generic
  • “relating to or characteristic of a whole group or class” (Merriam-Webster Online)
  • Solution space technique for developing parameterizable 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
Scope

• Generative programming can be applied at any level of granularity – from individual functions and classes to large software systems

• Generative programming is not limited to implementing application code, testing, and creating GUIs

• System family engineering also includes
  • documentation
  • system installation
  • user training
  • maintenance, and many more
And ... Watch Out For These Cats!