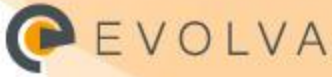
GP: AspectJ and GUI form generators

Daniel Strmečki | Web and mobile developer

10.05.2015.

BUSINESS WEB APPLICATIONS



Content

1. Generative programming (GP)
   - Feature-oriented modelling
   - Generic vs Generative programming
   - Aspect-oriented programming (AspectJ)
   - Template metaprogramming
   - Generators

2. GUI form generators
   - GUI form component based development
   - GUI form generators design

3. Demo
   - GUI form generators
   - AspectJ

Generative programming (GP)

EVOLVA

Feature-oriented modelling

Domain engineering goals

- Develop a common architecture for a **system family**
- Devise a production plan for **family members**

Software product lines goals

- Intensive systems share a common set of **features**, make use of it
- Increase the productivity and quality
- Reduce the development time, costs and complexity

Feature

- An important property of a concept instance
- Represents an reusable and configurable requirement

Feature modelling

- The **creative** activity of modelling the **common** and the **variable properties** of concepts and their **interdependencies**

Feature-oriented modelling

Feature model

- Feature diagram + additional information
- Features organised in diagrams express the configurability aspects of concepts

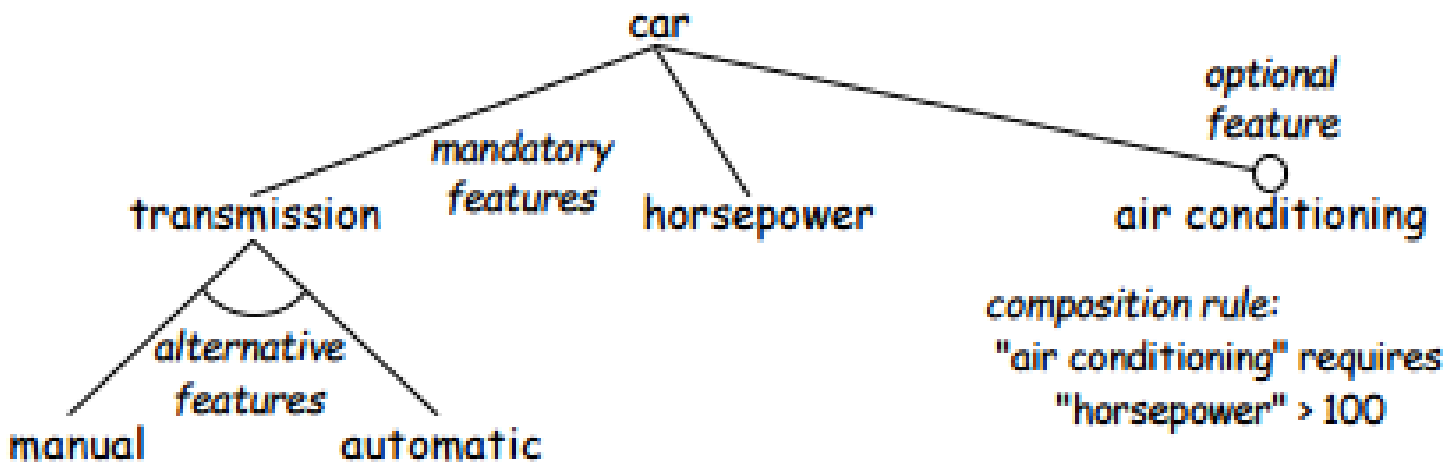


Figure copied from K. Czarnecki: Generative Programming, 1998.

Generic vs Generative programming

Generic programming

- Programming with **generic parameters**
- Programming by abstracting from concrete class
- Programming by parameterized components

Generative programming

- Generative – having the ability to **originate**, **produce** or **procreate**

Generic programming is focused on **parameterization**, whereas Generative programming additionally deploys **metaprogramming**.

Generative programming also includes the process of creating concrete instances of concepts.

Generic vs Generative programming

Generic parameters

- The goal is to avoid code duplication in statically typed languages
- Origin: STL library for C++

```
public static <E> void printArray(E[] inputArray) {
     for (E element : inputArray){
        System.out.printf( "%s ", element );
     }
     System.out.println();
}

public static void main(String[] args) {
    Integer[] intArray = { 1, 2, 3, 4, 5 };
    printArray(intArray);
    Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
    printArray(charArray);
}
```

Aspect-oriented programming

AOP base idea

- Provide support in the programming language for defining aspects along with the already present support for defining components
- An **aspect** is a common feature that is typically scattered across methods, classes, object hierarchies, or even entire object models
- AOP focuses on **crosscutting concerns** (logging, access rights…)
- It allows aspects to be cleanly separated and placed into modules that can be composed with other components

AspectJ

- Java implementation of AOP
- Aspect weavers operate by taking instructions specified by aspects, known as **advice**, and distributing it throughout the various classes in the program automatically (precompile)
- The place where a weaver inserts aspect code is called a **join point**

Aspect-oriented programming

AspectJ

- Join points (pointcuts)
  - method call
  - method execution
  - constructor call
  - constructor execution
  - field get
  - field set
  - pre-initialization
  - initialization
  - static initialization
  - handler
  - advice execution

```
pointcut onClickFunctions() : execution(* hr.evolva.modules.fom.forms.*.*onClick(..));

pointcut doCalculateFunctions() : call(* hr.evolva.modules.fom.forms.*.doCalculate(..));
```

„An aspect is not a something. It is a something about a something.”
(Highley, Lack, Myers)

Aspect-oriented programming

AspectJ

- Annotations
  - @Annotation1
  - !@Annotation1
  - @Annotation1 @Annotation2
  - @(Foo || Goo)
  - @Annotation1 (hr.evolva.* || com.evolution_framework.*)
  - @Annotation1 List hr.evolva..*.*

```
pointcut formEntryAnnotation() : execution(@FormEntry * *(..)) ;

pointcut secureMethodAnnotation() : execution(@SecureMethod *
                                    hr.evolva.modules.fom.forms.*.*()) ;
```

Aspect-oriented programming

AspectJ

- Errors and warnings
  - declare error
  - declare warning

```
declare error : execution(* hr.evolva.modules.fom.forms.*.doCalculate(..)) &&
                !within(@TrustedCode *) :
                "Untrusted code should not call calculate methods";

declare warning : execution(* hr.evolva.modules.fom.forms.*.*onClick(..)) &&
                !within(@TrustedCode *) :
                "Untrusted code should not call click methods";
```

```
27      public int doCalculate(int a, int b) {
28          return a * b;
29      }
```

Aspect-oriented programming

AspectJ

- Fields / variables
  - get(* hr.evolva.gui.controls.GLabelFloat.value)
  - set(* hr.evolva.gui.controls.GLabelEdit.text)

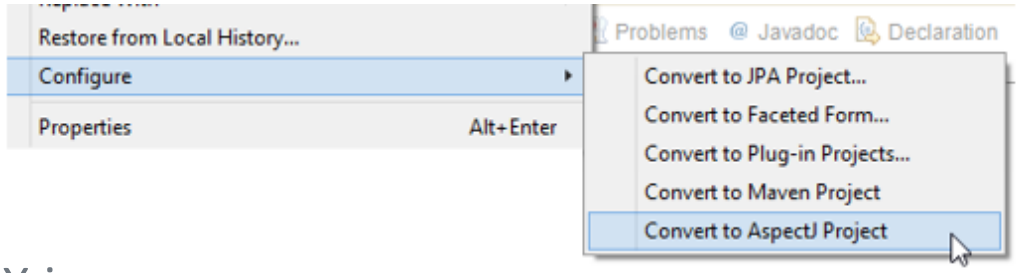
- Generics support
  - call(* *.*(List<?>))
  - execution(* C.*(List<? extends Number>))

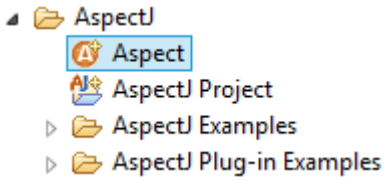
- Variable parameters
  - call(* hr.evolva.*.*(int, String...))

Aspect-oriented programming

AspectJ

- Getting started
  - Eclipse
  - JARs
    - aspectj-X.X.X.jar
    - aspectjrt-X.X.X.jar
  - AspectJ Development Tools
    - https://eclipse.org/ajdt/
    - http://download.eclipse.org/tools/ajdt/43/update
  - Your project
    - Configure -> Convert to AspectJ project
  - Create your first Aspect





Template metaprogramming

Metaprogramming refers to developing programs designed to read, generate, analyse or transform other programs, and even modify itself while running.

Template metaprogramming is a metaprogramming technique in which templates are used to generate source code.

Preview:

```
Where ${w1} = new Where(wz.sql);
${w1}.add("1=1");
String ${q1} = "SELECT * FROM table" + ${w1};
ResultSet ${rs1} = wz.sql.executeQuery(${q1});
while(${rs1}.next()) {
    //TODO
}
wz.sql.close(${rs1});
```

Generators

„A generator is a program that take a highel-lever specification of a piece of software and produces its implementation"
(Czarnecki, Eisenecker)

Generator tasks

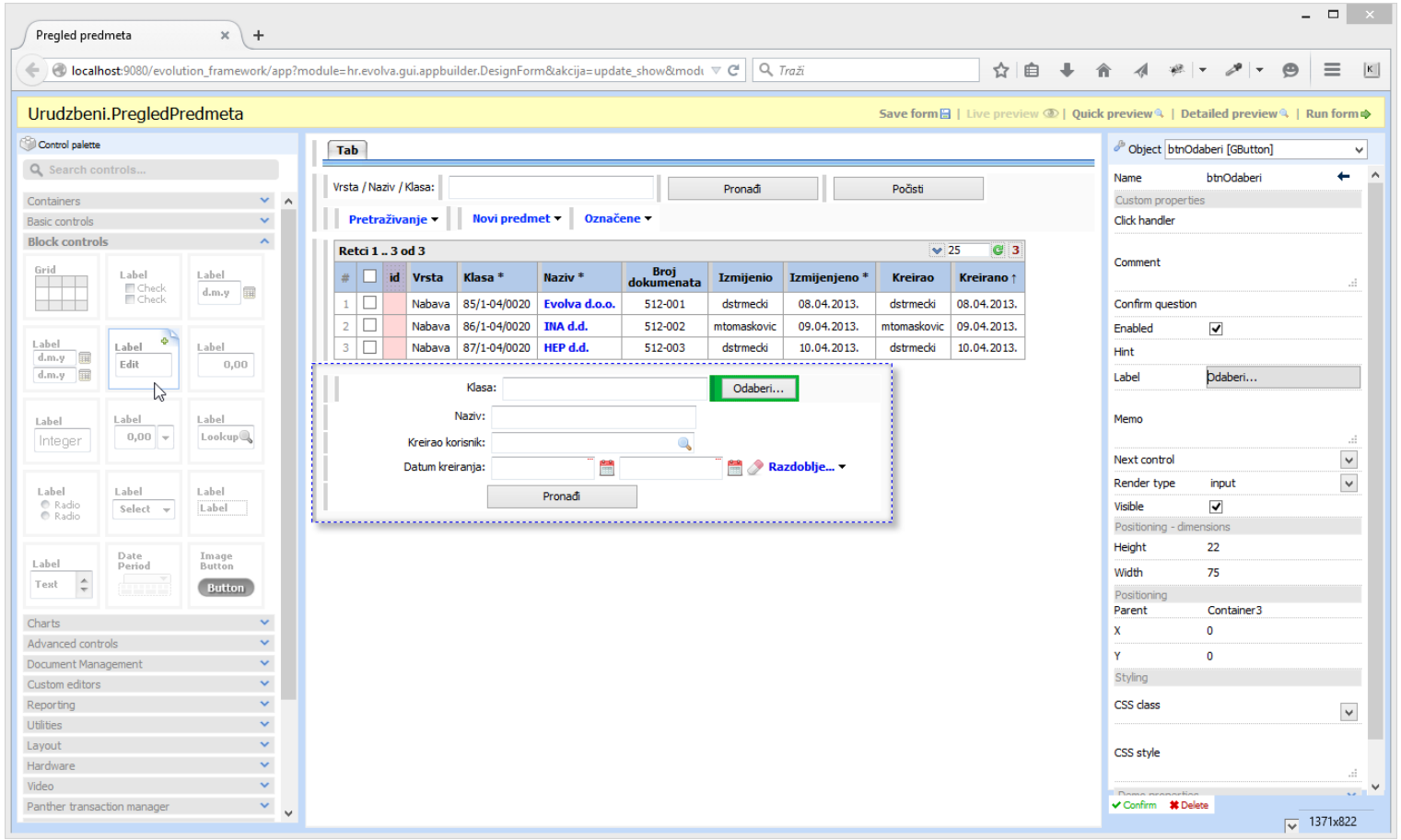
- Checks the validity of input specification
- Reports errors and warnings
- Completes the specification using the defaults
- Performs optimizations
- Generates the implementation

Generators that implement complex specification can become complex themselves. In that case: **modularize their design**.

- Implement larger generators as a set of cooperating, smaller generators

GUI form generators

GUI form component based development



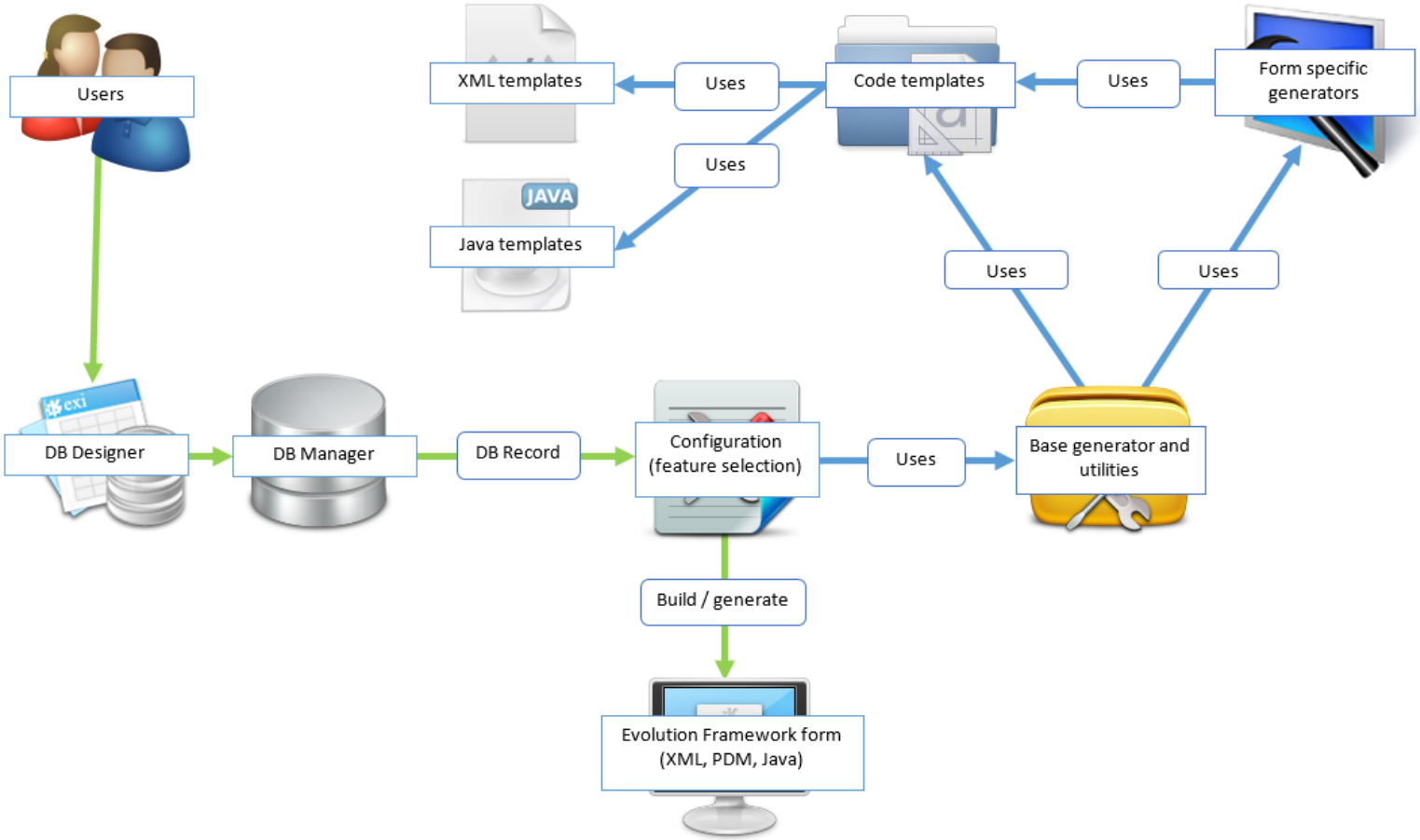
GUI form generators design

"Programmers often spend time on GPL programming tasks that are tedious and follow the same pattern." (M.Mernik)

Repetitive tasks

- View form
  - Grid data (dates, integers, floating-point numbers…)
  - Basic search
  - Advanced search
  - Add new record
  - Delete selected
- Edit form
  - Input fields (dates, integers, floating-point numbers…)
  - Delete record
  - Close form
  - Save record

GUI form generators design



DEMO

Literature

- Czarnecki K.: Generative Programming - Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment Based Component Models, PhD dissertation, 1998.
- Czarnecki K., U. Eisenecker: Generative Programming: Methods, Tools, and Applications, Paperback, 2000.
- Eclipse: AspectJ official documentation, 2005., https://eclipse.org/aspectj/doc/released/adk15notebook/index.html
- Highley T.J., Lack M., Myers P.: Aspect Oriented Programming, Technical Report, 1999.
- Kang K., Lee J., Donohoe P.: Feature-Oriented Product Line Engineering, IEEE Software, 2002.
- Magdalenić I., Radoševic D., Orehovački T.: Autogenerator: Generation and execution of programming code on demand, Expert Systems with Applications, 2013.
- Mernik M.: When and how to develop domain-specific languages, ACM Computing Surveys, 2005.
- Radošević D., Magdalenić I: Source Code Generator Based on Dynamic Frames, JIOS, 2011.
- Zhanga H., Jarzabek S.: XVCL: A Mechanism for Handling Variants in Software Product Lines, Science of Computer Programming, 2004.

Thanks for your attention

www.evolva.hr

daniel.strmecki@evolva.hr