Generative Software Development - Examples

Krzysztof Czarnecki
University of Waterloo
czarnecki@acm.org
Matrix Library Example
Matrix Example: Overview

- Feature Models
  - Implementation Components
  - Family Architecture
  - Generator
Domain Engineering: Analysis

- Stakeholder analysis and domain scoping
  - balancing market, economic, and strategic needs
  - GQM method, product maps, and economic model
  - linear or real options models (e.g., TrueScope method)

- Analysis of common and variable features and their dependencies

- Documentation using feature models (FODA)
  - Feature diagrams representing commonalities and variabilities in an implementation-independent and easy-to-understand manner
  - Additional information documenting feature dependencies, short descriptions, origin, rationale, defaults, binding time, usage examples, etc.
Feature Diagram

No decision regarding the mechanism for implementing variability!
Feature Combinations

- The feature diagram contains

\[ 4 \text{ (shape)} \times 3 \text{ (format)} \times 2 \text{ (bounds checking)} = 24 \text{ combinations} \]

- Additional constraints (requires & implies) are documented separately

- Not considered: density, memory management, blocking, index base, etc.
Variability is implemented using concrete mechanisms, e.g. inheritance, aggregation, associations, and templates.
Feature-Oriented Approach

Domain analysis

Problem-space view

Feature Model

Solution-space view

• Architectural patterns
• Variability mechanisms

DSL

Mapping

Family architecture

Generator Technology

Component technology
Matrix Example: Overview

- Feature Models
- Implementation Components
- Family Architecture
- Generator
Domain Engineering: Design

- Identifying categories of implementation components
  - feature diagrams as a starting point
    - concrete features map to concrete components
    - aspectual features map to aspects
    - abstract features to configurations of components and aspects
  - a category is a set of plug-compatible components and corresponds to a concrete variation point

- Dependency analysis
  - analyzing use and refinement dependencies

- Architecture for the system family
  - GenVoca layered architecture
    - lower layer parameterizes the layer above it
  - GenVoca grammar
Implementation Components

Matrix: Matrix
BoundsChecking: BoundsChecker
Symmetry: Symm
Format: ArrFormat
LoTriVecFormat
UpTriVecFormat
Container: D1Container
D2CContainer
D2FContainer
Dependency Analysis

Matrix
  └── BoundsChecking
    ├── Symmetry
    └── Format

Container

Matrix
  └── BoundsChecking
    ├── Symmetry
    └── Format
        └── Container
Matrix Example: Overview

- Feature Models
- Implementation Components
- Family Architecture
- Generator
# GenVoca Layered Architecture

<table>
<thead>
<tr>
<th>MatrixType: Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoundsChecking: BoundsChecker</td>
</tr>
<tr>
<td>Symmetry: Symm</td>
</tr>
<tr>
<td>Format: ArrFormat</td>
</tr>
<tr>
<td>Container: D1Container</td>
</tr>
<tr>
<td>Configuration: ElementType, Matrix, Format, ...</td>
</tr>
</tbody>
</table>

Homepage of the GenVoca Project,
www.cs.utexas.edu/users/schwartz

Notation from Czarnecki&Eisenecker, Generative Programming, 2000
## GenVoca Grammar

<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MatrixType:</strong></td>
<td><code>Matrix[OptBoundsCheckedMatrix]</code></td>
</tr>
<tr>
<td><strong>OptBoundsCheckedMatrix:</strong></td>
<td>`OptSymmMatrix</td>
</tr>
<tr>
<td><strong>OptSymmetricMatrix:</strong></td>
<td>`Format</td>
</tr>
<tr>
<td><strong>Format:</strong></td>
<td>`ArrFormat[Array]</td>
</tr>
<tr>
<td><strong>Array:</strong></td>
<td>`D2CContainer[Config]</td>
</tr>
<tr>
<td><strong>Vector:</strong></td>
<td><code>D1Container[Config]</code></td>
</tr>
<tr>
<td><strong>Config:</strong></td>
<td><code>MatrixType, ElementType, IndexType, Format, ...</code></td>
</tr>
</tbody>
</table>
Matrix Components

template<class Config_>
class D1Container
{
public:
    typedef typename Config_ Config;
    typedef typename Config::ElementType ElementType;
    typedef typename Config::IndexType IndexType;
    //...
    const ElementType& getElement(const IndexType&) const;
    void setElement(const IndexType&, const ElementType&);
    const IndexType& size() const;

private:
    IndexType size_;
    ElementType *pData_; 
};
Matrix Components - Forwarding Adapter

template<class Vector>
class LoTriangVecFormat
{
   public:
      typedef typename Vector::Config Config;
      typedef typename Config::ElementType ElementType;
      typedef typename Config::IndexType IndexType;

   private:
      const IndexType order_;  
      Vector elements_; 

   public:
      //...
      void setElement(
               const IndexType & i, const IndexType & j, const ElementType & v) 
      { if (i >= j) elements_.setElement(getIndex(i, j), v); 
         else assert(v == ElementType( 0 )); 
      }
      //...
   
};
Matrix Components - Mixin

template<class OptSymmMatrix>
class BoundsChecker : public OptSymmMatrix
{public:
    typedef typename OptSymmMatrix::Config Config;
    typedef typename Config::ElementType ElementType;
    typedef typename Config::IndexType IndexType;
    //...
    ElementType getElement(const IndexType& i, const IndexType& j) const
    {
        checkBounds(i, j);
        return OptSymmMatrix::getElement(i, j);
    }
private:
    void checkBounds(const IndexType & i, const IndexType & j) const;
};
Configuration

// lower symmetric matrix of floats stored in a vector
// with bounds checking
struct C1
{
  typedef float ElementType;
  typedef unsigned long IndexType;
  typedef CCommaInitializer CommaInitializer;

  // MatrixType
typedef Matrix<
    BoundsChecker<
      Symm<
        LoTriVecFormat<
          D1Container<C1> > > > > > MatrixType;
};
Manual Configuration

• Manually writing 24 configuration repositories is tedious and error prone
• Not all combinations of components are valid
• How to automatically produce valid configurations?

⇒ Configuration generators
Matrix Example: Overview

- Feature Models
- Implementation Components
- Family Architecture
  ➔ Generator
Deriving a Configuration DSL

Derivation process

• feature diagrams as a starting point
• user-specific view
• opportunity for introducing new abstract features
• determining feature defaults
• ordering according to frequency of use
• renesting to maximize specification efficiency and safety
Simulating Keyword-Template-Parameters

• Keyword-Template-Parameters can be simulated using wrapper templates
  • Parameters can be specified in any order
  • Can use the default value of any parameter (not only the last ones)
• Example
  • MatrixGenerator< ElemType<double>, Shape<lower_triang> >::RET
  • MatrixGenerator<Optimize<speed>, Shape<upper_triang>, ElemType<double> >::RET
• See implementation details at http://www.generative-programming.org/namedparams/
Configuration Generator

template<   class P1= Nil,class P2= Nil,class P3= Nil,
            class P4= Nil,class P5= Nil,class P6= Nil >
struct MatrixGenerator
{
    // define a short name for the complete generator
    typedef MatrixGenerator<P1,P2,P3,P4,P5,P6> Generator;

    // compute parameter defaults (and check combinations)
    typedef ComputeMatrixParamDefaults<P1,P2,P3,P4,P5,P6> Params;
    typedef Params::ElementType ElementType_;
    typedef Params::IndexType IndexType_
    enum{
        shape = Params::shape,
        format = Params::format,
        boundsChecking = Params::boundsChecking,
        isSymmetric = shape==symm,
    };
}
Configuration Generator

// assemble components
typedef SWITCH<format,
    CASE<cArray, D2CContainer<Generator> >,
    CASE<fArray, D2FContainer<Generator> >,
    CASE<vector, D1Container<Generator> >
> > > > ::RET Container_

// ...
typedef IF<isSymmetric,
    Symm<Format> ,
    Format> ::RET OptSymmetricMatrix;
typedef IF<boundsChecking,
    BoundsChecker<OptSymmetricMatrix> ,
    OptSymmetricMatrix> ::RET OptBoundsCheckedMatrix;
typedef Matrix<OptBoundsCheckedMatrix> RET; // complete matrix type
typedef EVAL_DEPENDENCY_TABLE //Format (see Table 14-7)

恃 typedef EVAL_DEPENDENCY_TABLE //Format (see Table 14-7)

< CELL< shape, CELL< format >>

, ROW< CELL< lowerTriang, CELL< vector, RET< LoTriangVecFormat<Vector>> >>

, ROW< CELL< lowerTriang, CELL< vector, RET< LoTriangVecFormat<Vector>> >>

, ROW< CELL< upperTriang, CELL< vector, RET< UpTriangVecFormat<Vector>> >>

, ROW< CELL< anyValue, CELL< anyValue,RET< ArrFormat<Array>> >>

> > > > > : RET Format;
C++ Template Metaprogramming

- C++ templates define a Turing-complete, functional language executed at compile time (Erwin Unruh, Todd Veldhuizen)
- C++ as a heterogeneous two-level language
- C-like control structures IF, SWITCH, DO, WHILE, FOR available
  - www.boost.org
- Comparison to other languages supporting metaprogramming
Configuration Generator

// generating configuration repository
struct Config
{
    //DSL features
    typedef Params DSLFeatures;
    //features required by components
    typedef ElementType_ ElementType;
    typedef IndexType_ IndexType;
    typedef CommaInitializer_ CommaInitializer;
    //MatrixType
    typedef RET MatrixType;
};

// the configuration repository is a part of each product; i.e., it
// can be accessed by other generators (e.g. for generating
// customized algorithms)
A Small Change Still Needed...

template<class Generator> // Generator instead of Config
class D1Container
{
  public:
    typedef typename Generator::Config Config;
    typedef typename Config::ElementType ElementType;
    // ...

};

// D2CContainer
...

// D2FContainer
...
typedef MatrixGenerator<>::RET M1; // give me a default one
typedef MatrixGenerator<ElementType<float>, Shape<upperTriang>, OptFlag<speed> >::RET M2;
typedef MatrixGenerator<BoundsChecking<false>, ElementType<long double> >::RET M3;

// ...
M1 m1(3,3); M2 m2(3,3); M3 m3(3,3);
// ...
Usage

//initialization using a comma-separated list
m1 = 1, 0, 3,
    0, 1, 4,
    2, 0, 1

//...

//evaluate a matrix expression
m1 = m2 + m3 + m2;

//print the result
cout << "m1 = " << endl << RectMatrix1 << endl;
Applications

- Matrix Template Library (MTL)
  - University of Notre Dame, USA
  - currently the fastest generic matrix library in C++
  - incorporated the generative design of GMCL as an extra layer
    - [http://www.lsc.nd.edu/](http://www.lsc.nd.edu/)
- Adapter Iterator Library
  - contains a set of generators for creating STL-conforming iterator adapters
    - [http://www.boost.org](http://www.boost.org)
Technology Projection - C++ Templates

Problem Space
- Domain-specific concepts and
- Features

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

Solution Space
- Elementary components
- Maximum combinability
- Minimum redundancy

DSL encoded in C++ itself using
- constants
- types
- templates

Template metaprogramming in C++

Generic programming in C++
On-Board Satellite Software Example
Problem

• Satellite On-Board Software
  • Several functions, e.g., attitude and orbit control, thermal control, power management, data handling
  • Organized as a set of applications that communicate through packets
  • Services for monitoring/housekeeping, transmission control, storage control etc.
  • On-Board Data-Handling System (OBDHS)
  • Packet Utilization Standard (PUS) – European Space Agency

• Goal
  • Software system family engineering instead of one-of-a-kind development
  • Implementation of PUS as a generative toolkit
Proposed Technology Projection

**Problem Space**
- Domain-specific concepts and
- Features

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

**Solution Space**
- Elementary components
- Maximum combinability
- Minimum redundancy

Specification of a system configuration using a **graphical editor** (checks and completes the specification)

Configuration definition in **XML**

Generation of variable parts of the architecture in Ada 83 using the template language **TL**

Architecture framework in **Ada83**
Technology Projection - Overview

- General-purpose configuration editor
- Satellite configuration in XML
- Satellite Product Line Architecture (Ada83 + Templates)
- Concrete Satellite
- Ground station

Product line definition (feature model in XML format)

load

save

Statically configure

Dynamically configure

Generate

Communication
Overview - Feature Model

Product line definition (feature model in XML format)

General-purpose configuration editor

Satellite configuration in XML

Generate

Satellite Product Line Architecture (Ada83 + Templates)

Statically configure

Dynamically configure

Concrete Satellite

Communication

Ground station
Satellite Feature Model Using AmiEddi
Overview - Configuration Editor

General-purpose configuration editor

Product line definition (feature model in XML format)

Satellite configuration in XML

Satellite Product Line Architecture (Ada83 + Templates)

Concrete Satellite

Ground station

Generate

Statically configure

Dynamically configure

load

save

Communication
ConfigEditor

Adding features
Overview - Product Line Architecture

Product line definition (feature model in XML format) -> Satellite configuration in XML

Statically configure

Dynamically configure

Generate

Communication

Concrete Satellite

Ground station

General-purpose configuration editor

save

load

Satellite Product Line Architecture (Ada83 + Templates)
Satellite Product Line Architecture - Overview

Application $x_1$

service implementation $x_1$

interface

... service implementation $x_n$

Application $x_n$

main computer e.g. 1750 processor

on-board bus

Telemetry packet

Telecommand packet
Template Language

- Templates - source code instrumented with preprocessor directives
- The directives accesses information in a XML configuration file
- Automatic generation of a program generator form the templates

Craig Cleaveland, Program Generators with XML and Java.
Template Language - Selection

#if "/satellite/definition/transmissioncontrol" = 'true'#
    with transmissioncontrol;
#fi#

separate(app)

procedure send(o :ptr; p :in out pus.ptr) is
    transmission :boolean := true;

begin
#if "/satellite/definition/transmissioncontrol" = 'true'#
    transmission := transmissioncontrol.exec(o.transControl, p);
#fi#

    if transmission then
        packetrouter.send(p);
    end if;
end send;
Template Language - Expressions

• Getting a constant from the XML configuration

MAX_STORAGE_OBJECTS: constant integer
  :=
  #"//Attribute[@Name='MAX_STORAGE_OBJECTS']/attribute::Value"# ;

• Counting nodes

MAX_APPS: constant integer:=
  #"count(//App)"#;
Template Language - Iteration

```plaintext
#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;
  ...
end decode;
```

```plaintext
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;
```
Overview – Generated System

General-purpose configuration editor

Product line definition (feature model in XML format)

Satellite configuration in XML

Satellite Product Line Architecture (Ada83 + Templates)

Statically configure

Generate

Concrete Satellite

Dynamically configure

Communication

Ground station
Satellite Simulation Environment

Satellite monitoring panel

Ground station sender

Ground station receiver
Automotive Embedded Software Example
Software Technology Progress in the Automotive Industry

- Electronic control units (ECUs)
  - Low-resource microprocessors (8/16/32 bit, kB RAM/ROM)
- In the past: ECUs programming in assembly
- Around 1985: switch to C
  - Initially developers studied the assembly produced by the compiler
  - No one does this anymore
  - Compiler errors are still found from time to time
- Model-Based Development using Matlab/Simulink/Stateflow
  - Simulink - time-continuous, signal flow
  - Stateflow - reactive parts, state charts
  - simulation (SIL, HIL)
  - code generation (“autocoding”)
Control Systems: Visual Formalisms: Statecharts
Control Systems: Visual Formalisms: Block Diagrams

ABS Braking Model

Simulation

Vehicle speed and wheel speed

Vehicle speed ($\omega_v$)

Wheel speed ($\omega_w$)

Code Generation

```c
/* Compute block outputs */
void MdlOutputs(int_T tid)
{
    /* local block i/o variables */
    real_T rtb_Yt;
    real_T rtb_Brake_pressure;
    real_T rtb_Suml;

    if (ssIsContinuousTask(rts, tid)) { /* Sample time: [0.0, 0.0] */
        /* Integrator: '<Root>/Wheel Speed' */
        if (rtX.Wheel_Speed_CSTATE >= rtP.Wheel_Speed_UpperSat) {
            rtX.Wheel_Speed_CSTATE = rtP.Wheel_Speed_UpperSat;
        } else if (rtX.Wheel_Speed_CSTATE <= rtP.Wheel_Speed_LowerSat) {
            rtX.Wheel_Speed_CSTATE = rtP.Wheel_Speed_LowerSat;
        }
        rtB.Yw = rtX.Wheel_Speed_CSTATE;

        /* Integrator: '<Root>/Vehicle speed' */
        if (rtX.Vehicle_speed_CSTATE >= rtP.Vehicle_speed_UpperSat) {
            rtX.Vehicle_speed_CSTATE = rtP.Vehicle_speed_UpperSat;
            rtB.Vehicle_speed_o2 = 1.0;
        } else if (rtX.Vehicle_speed_CSTATE <= rtP.Vehicle_speed_LowerSat) {
            rtX.Vehicle_speed_CSTATE = rtP.Vehicle_speed_LowerSat;
            rtB.Vehicle_speed_o2 = -1.0;
        } else {
            rtB.Vehicle_speed_o2 = 0.0;
        }
        rtB.Vehicle_speed_o1 = rtX.Vehicle_speed_CSTATE;
    }
/* Gain: '<Root>/Vehicle speed (angular)' */

    /* Compute model derivatives */
    void MdlDerivatives(void)
    {
        /* Limited Integrator Block: '<Root>/Wheel Speed' */
        real_T *dx = ssGetdx(rts);
        boolean_T lsat;
        boolean_T usat;

        lsat = (rtX.Wheel_Speed_CSTATE <= rtP.Wheel_Speed_LowerSat);
        usat = (rtX.Wheel_Speed_CSTATE >= rtP.Wheel_Speed_UpperSat);

        if ((lsat && !usat) ||
            (lsat && (rtB.I > 0)) ||
            (usat && (rtB.I < 0))) {
            dx[0] = rtB.I;
        } else {
            /* in saturation */
            dx[0] = 0.0;
        }

        /* Limited Integrator (w/ Saturation Port) Block: '<Root>/Vehicle speed' */
        real_T *dx = ssGetdx(rts);
        boolean_T lsat;
        boolean_T usat;
```
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)

Electronic Control Unit

Application Code (Control Functions)

Standard Components (OS, Network Management, Diagnosis, Flashing, etc.)

Firmware

Hardware

Static configuration

Dynamic configuration (Calibration)

CAN Bus
Structure Spectrum of DSLs

Routine configuration  Creative construction

Wizards

Feature-based configuration

Graph-like language
(with user-defined elements)

Path through a decision tree  Subtree of a feature tree  Subgraph of an (infinite) graph
Summary & Discussion
Key Concepts in Generative Software Development

- Software system families
  - Cornerstone of systematic software reuse
- Feature modeling
  - Family scoping, DSL & architecture development
- Domain-specific languages
  - Optimal support for application developers
- Mappings
  - Design knowledge capture
Relation To Other Fields

Problem Space

- Feature modeling, feature interactions
- Domain-specific languages
- Aspect-oriented modeling

Configuration Knowledge

- System Families, Product Lines
- Generators/Transformers
- Aspect-oriented programming

Solution Space

- Components
- Architectures
- Generic programming
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
Discussion