Introduction to Generative Programming

Michal Antkiewicz
mantkiew@swen.uwaterloo.ca

Overview

- What is Generative Programming?
- GP Process
- Technology Projections
- Object Technology
- Component Technology
- Feature Modeling
- Exercise
- Specialization
- MDA

What is Generative Programming?

- "... is a software engineering paradigm based on
- modeling ***software system families*** such that,
- given a particular ***requirements specification***, a
- highly customized and optimized intermediate or
- end-product can be ***automatically manufactured***
- on demand from elementary, reusable
- ***implementation components*** by means of
- ***configuration knowledge***" [Cza02]

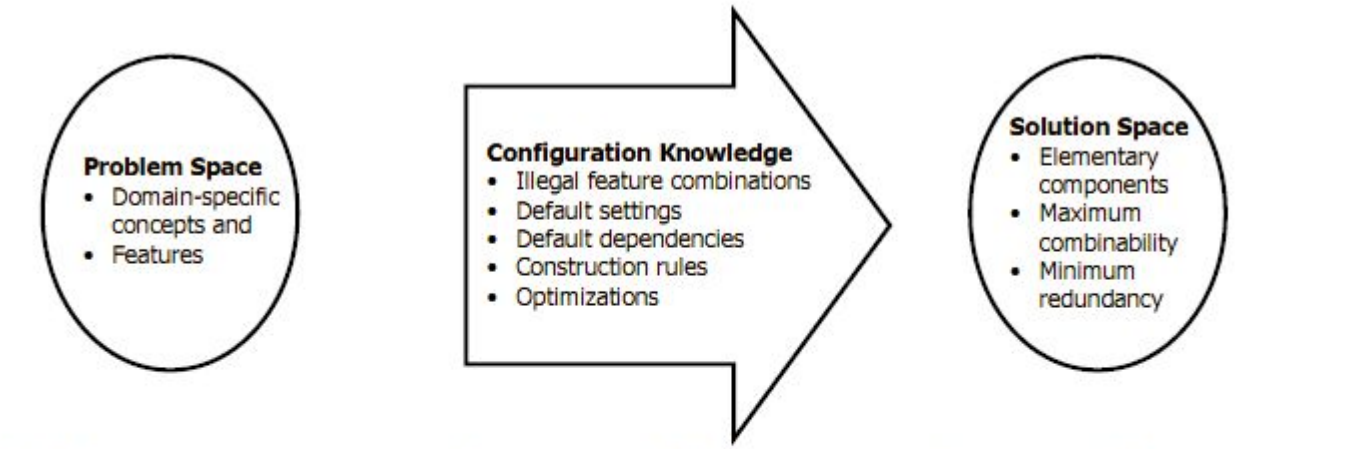
Generative Programming Process

- Two parallel processes:
  - development for reuse
  - development with reuse
- Introducing GP is not always profitable

Development For Reuse

- Create generative domain model (means of specification of members, implementation components and configuration knowledge)
- Capture the scope of system family
- Capture commonalities and variation points – feature modeling
- Design and implement a system family model
  - Choose common architecture,
  - Provide means of specifying family members,
  - Capture configuration knowledge in a generator,
  - Implement a model using generative technologies.

Technology Projections



DSL Technologies
- Programming language
- Extendible languages
- New textual languages
- Graphical languages
- Interactive wizards and GUIs
- Any mixture of the above

Generator Technologies
- Simple traversal
- Templates and frames
- Transformation systems
- Programming languages with metaprogramming support
- Extendible programming systems

Component Technologies
- Generic components
- Component models
- AOP approaches

2

This is a recursive process. One's solution space may be someone's else problem space.

Object Technology

- Why it does not suffice?
  - classes are too small units of reuse,
  - frameworks are sufficiently large 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.
- GP supports better software and knowledge reuse.

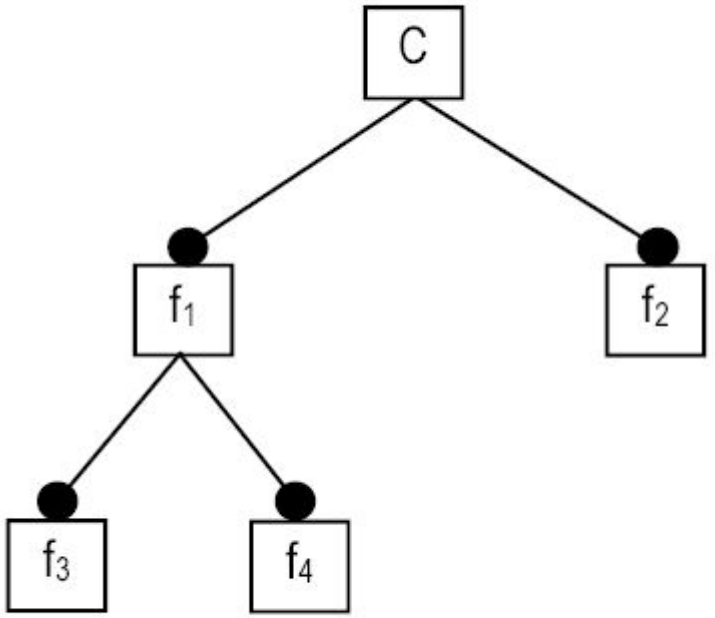
Component Technology

- Ongoing development improves component interoperability
- Reusing small components does not have a large impact on software development, and large components require high customization efforts,
- Problem with “fat components”,
- In GP, rather than having to search for needed components by name, they are generated to support required features.

Feature Modeling

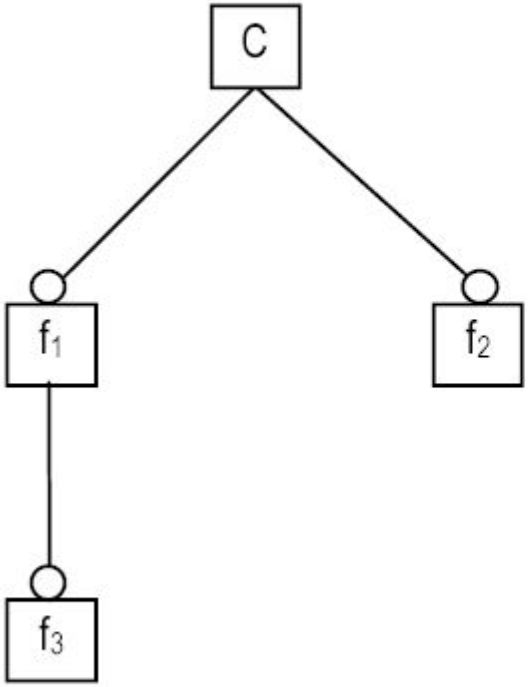
- Part of the Development for Reuse process,
- The goal is to find commonalities and variation points in system family,
- Feature diagrams are the basis for deriving the categories of implementation components,
- Choosing a concrete member of system family is called specialization and provides input for generator.

Feature Modeling: Mandatory Feature



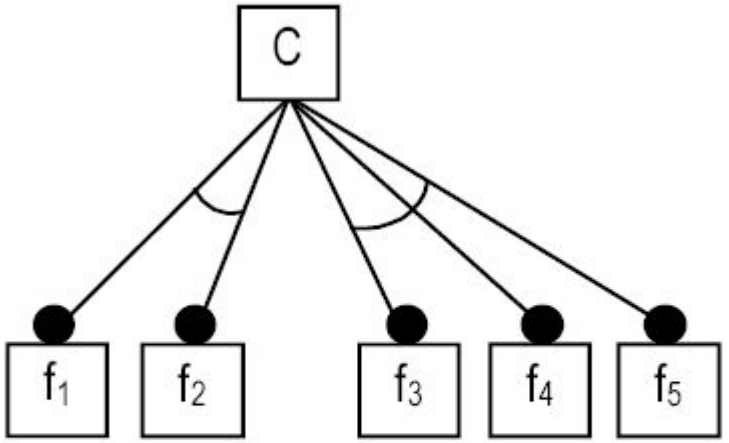
- A mandatory feature is part of a concept instance description only if its parent is also part of the description
- Mandatory features are pointed to by edges with a filled circle, e.g. f1,f2,f3, and f4
- All instances of C are described by the feature set {C,f1,f2,f3,f4}

Feature Modeling: Optional Feature



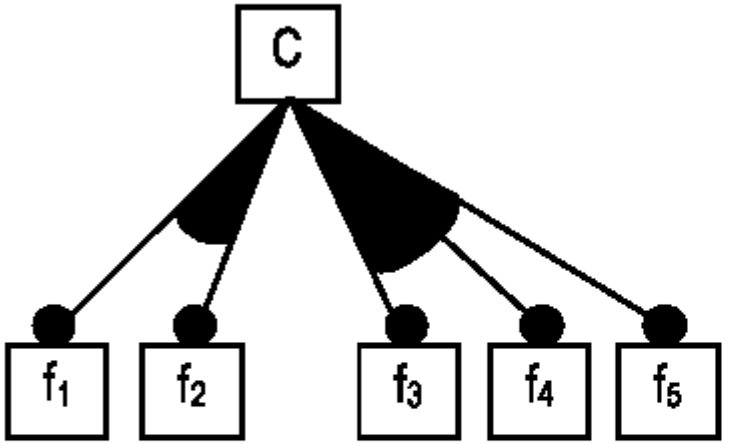
- An optional feature can be part of a concept instance description only if the parent node is also part of the description
- Optional features are pointed to by edges with an empty circle (e.g., f1,f2, and f3)
- The following sets describe instances of C: {C}, {C,f1}, {C,f1,f3}, {C,f2}, {C,f1,f2}, {C,f1,f3,f2}

Feature Modeling: Exclusive-Or



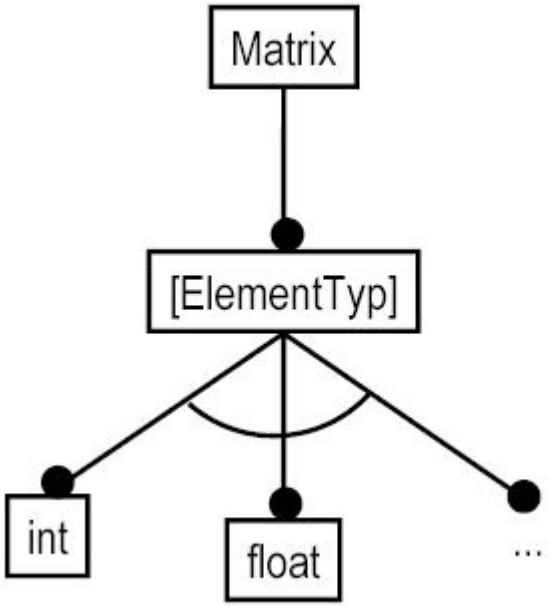
- Exactly one from a set of exclusive-or features is part of a concept instance description if its parent node is also part of the description
- Edges pointing to exclusive-or features of one set are connected by an empty arc
- The following sets describe instances of C:
  {C,f1,f3}, {C,f1,f4},
  {C,f1,f5}, {C,f2,f3},
  {C,f2,f4}, {C,f2,f5}

Feature Modeling: Inclusive-Or



- Any non-empty subset from a set of inclusive-or features can be part of a concept instance description if the parent node is also part of it
- Edges pointing to inclusive-or features of one set are connected by a filled arc
- The diagram denotes ((2*2) – 1) * ((2 * 2 * 2) -1) = 21 different concept instances

Feature Modeling: Open Feature

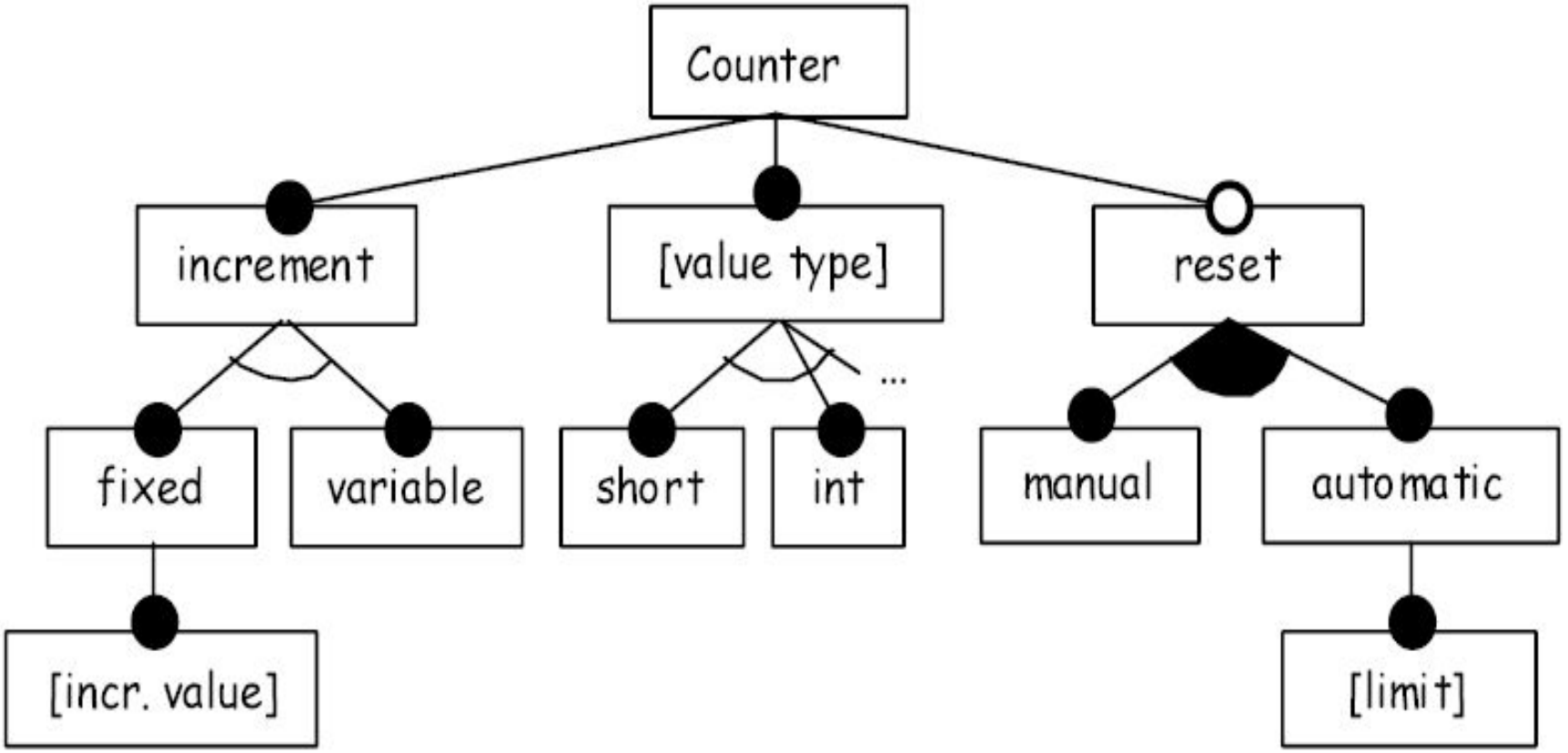


- An open feature is expected to be refined with further sub-features
- In a feature diagram, brackets [] are used to indicate openness
- We can also show selected examples of sub-features (not part of the formal notation)

Exercise: Family of Counters

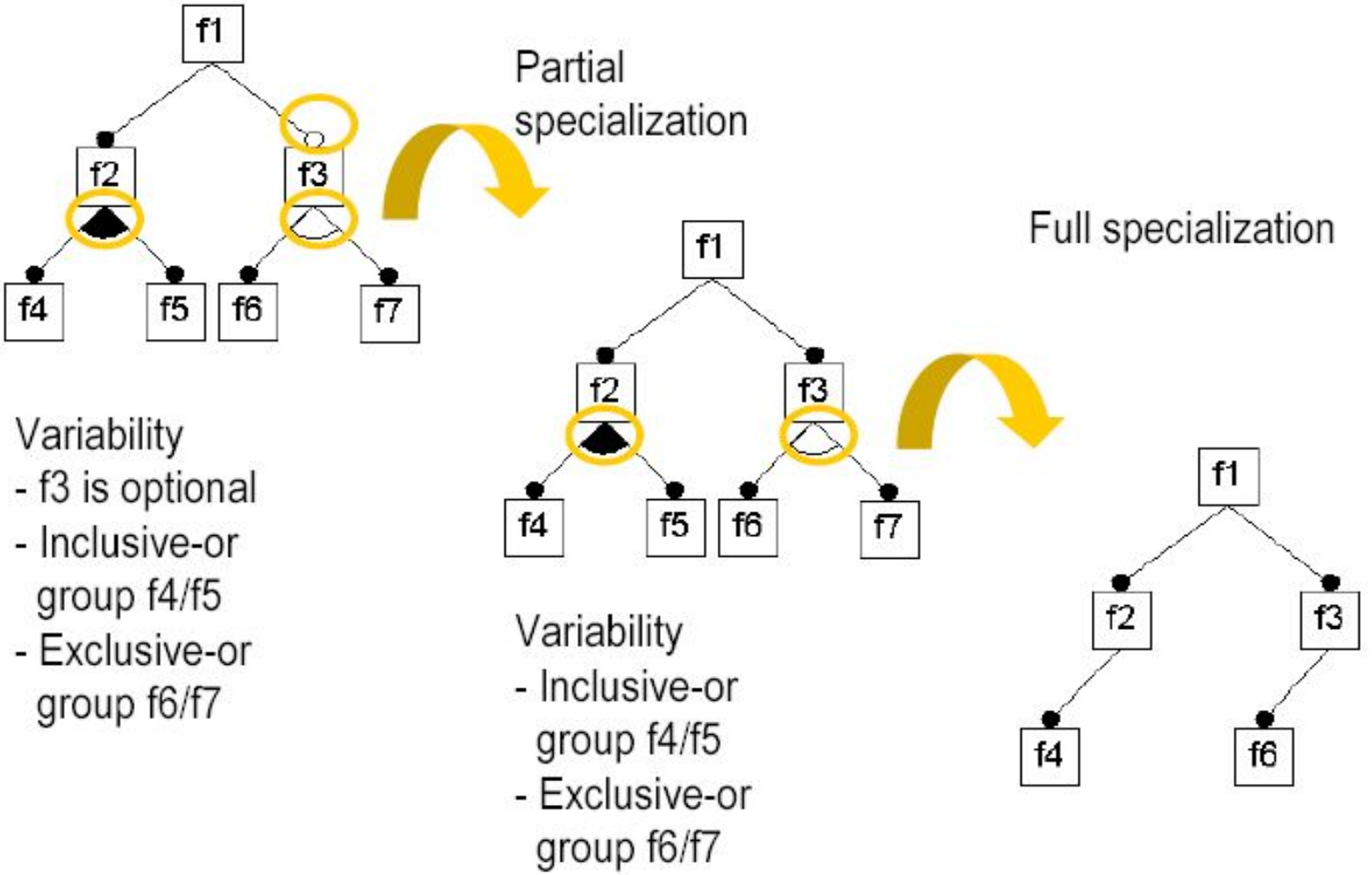
- Detailed requirements:
  - support a fixed and variable increment,
  - the value of fixed increment can be 'statically' specified,
  - support different counter value types (short, int, long),
  - assume, that more value types can be added,
  - may optionally support manual or automatic reset (or both); automatic reset is activated, when the counter value exceeds a reset limit,
  - the reset limit can be 'statically' specified.
- Draw feature diagram for family of counters and count the number of valid family members.

Counter Family Feature Diagram



This diagram denotes 2*2*4=16 different counter configurations.

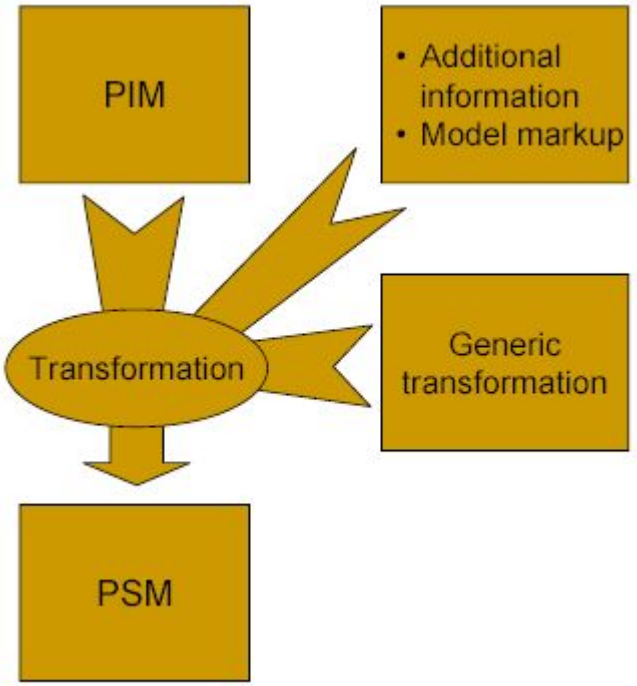
Specialization



Model Driven Architecture

- MDA is a significant, emerging part of GP
- MDA is about transformations of models
  - PIM – platform independent model
  - PSM – platform specific model
- Transformations:
  - PIM to PIM
  - PIM to PSM – changing the level of abstraction
  - PSM to PSM -> End-product

MDA Pattern


PIM
• Additional information
• Model markup
Transformation
Generic transformation
PSM

Benefits of MDA

- Preserving the investment in knowledge
  - Independent of implementation platform
  - Tacit knowledge is made explicit
- Speed of development
  - Most of the code is generated
- Quality of implementation
  - Experts provide transformation templates
- Maintenance and documentation
  - Design and analysis models are not abandoned after writing
  - 100% of traceability from specification to implementation