On the Diversity of Capturing Variability at the Implementation Level

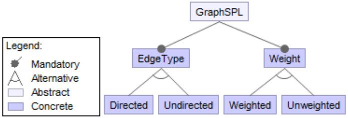
Xhevahire Tërnava and Philippe Collet

Tuesday 26th September, 2017

Université Côte d'Azur, CNRS, I3S, Sophia Antipolis, France

Motivation

Specification Level



mapping

Implementation Level

```
object Conf {
  final val WEIGHTED: Boolean = true
}
abstract class Graph { /* Common part */ }
class ConcreteGraph extends Graph {
  def adddirectededge(s: Vertex, d: Vertex, w: Int) = {
    val edge = new Edge(s, d)
    if (Conf.WEIGHTED) {
      edge.weight = w
    }
    edges = edge :: edges
    addtoadjacencymatrix(edge)
  }
  def addundirectededge(s: Vertex, d: Vertex, w: Int) = {
    val edge1 = new Edge(s, d)
    val edge2 = new Edge(d, s)
    if (Conf.WEIGHTED) {
      edge1.weight = w
      edge2.weight = w
    }
    edges = edge1 :: edges
    edges = edge2 :: edges
    addtoadjacencymatrix(edge1)
    addtoadjacencymatrix(edge2)
  }
  def addedge(callback: (Vertex, Vertex, Int) => Unit,
      x: Vertex, y: Vertex, w: Int = 1) = callback(x, y, w)
```

core-code assets with traditional techniques

- During evolution, their mapping may deteriorate
- It's needed to reconstruct the FM, or part of it
  - Code is not shaped in terms of features

Motivation

Reverse engineering approaches abstract from the implementation technique

- Feature locations
- Reconstructing the FM from the propositional formula, ...
- When a single technique is used (e.g., preprocessors in C)

Motivation

Reverse engineering approaches abstract from the implementation technique

- Feature locations
- Reconstructing the FM from the propositional formula, ...
- When a single technique is used (e.g., preprocessors in C)

The addressed issues:

I1. Capturing the variability implementation technique

I2. Capturing features and variation points is not the same

I3. The importance of techniques in a reverse engineering process

Variability Realization

Core-code assets consist of: core, commonalities, variabilities

Variability Realization

Core-code assets consist of: core, commonalities, variabilities

Variable Part: abstractions

commonalities: variation points (*vp*-s)

variabilities: variants

variability implementation techniques

(e.g., inheritance, generic types, design patterns)

Dimensions of Diversity

Features in a feature model:

- Parent-child hierarchy
- Logical relations
- Cross-tree constraints

Variation points with variants:

- A richer set of Characteristic Properties
- Realized by diverse Techniques

Dimensions of Diversity

Features in a feature model:

- Parent-child hierarchy
- Logical relations
- Cross-tree constraints

```
1  object Conf {
2    final val WEIGHTED: Boolean = true
3  }
4  abstract class Graph { /* Common part */ }
5  class ConcreteGraph extends Graph {
6    def adddirectededge(s: Vertex, d: Vertex, w: Int) = {
7      val edge = new Edge(s, d)
8      if (Conf.WEIGHTED) {
9        edge.weight = w
10     }
11     edges = edge :: edges
12     addtoadjacencymatrix(edge)
13   }
14   def addundirectededge(s: Vertex, d: Vertex, w: Int) = {
15     val edge1 = new Edge(s, d)
16     val edge2 = new Edge(d, s)
17     if (Conf.WEIGHTED) {
18       edge1.weight = w
19       edge2.weight = w
20     }
21     edges = edge1 :: edges
22     edges = edge2 :: edges
23     addtoadjacencymatrix(edge1)
24     addtoadjacencymatrix(edge2)
25   }
26   def addedge(callback: (Vertex, Vertex, Int) => Unit,
27     x: Vertex, y: Vertex, w: Int = 1) = callback(x, y, w)
```

Variation points with variants:

- A richer set of Characteristic Properties
- Realized by diverse Techniques

vp_edgetype (26-27), v_directed (6-13), v_undirected (14-25)

vp_weight (2), v_weighted (2, ‘true‘), v_unweighted (2, ‘false‘)

Characteristic Properties of Variable Parts

- Logical relation
- Binding time
- Defaults
- Granularity
- Evolution
- Quality criteria

- Mandatory
- Optional
- Multi-Coexisting (Or)
- Alternative

vp_edgetype with Alternative (v_directed, v_undirected), Strategy Pattern

Characteristic Properties of Variable Parts

- Logical relation
- Binding time
- Defaults
- Granularity
- Evolution
- Quality criteria

Binding	Values
Static binding (S)	(S) compilation / link (S) build / assembly (S) programming
Dynamic binding (D)	(S/D) configuration (S/D) (re) deploy (D) runtime (start-up) (D) pure runtime (operational mode)

vp_edgetype is bound during Runtime, e.g., to v_directed

Characteristic Properties of Variable Parts

- Logical relation
- Binding time
- Defaults
- Granularity
- Evolution
- Quality criteria

- Default variant

Some variability may not be subject to frequent variations among the majority of software products in an SPL

v_unweighted is a Default variant of vp_weight

Characteristic Properties of Variable Parts

- ☐ Logical relation
- ☐ Binding time
- ☐ Defaults
- ☐ Granularity
- ☐ Evolution
- ☐ Quality criteria

Granularity	Values
Coarse grained	► Component, framework with plug-ins as variants, file, package, class, interface, frame, feature module, etc.
Medium grained	► Method, field inside a class, aspect, delta module, frame, etc.
Fine grained	► Expression, statement, block of code within a method, frame, etc.

vp_weight at a Parameter level, or v_directed at a Method level

Characteristic Properties of Variable Parts

- ☐ Logical relation
- ☐ Binding time
- ☐ Defaults
- ☐ Granularity
- ☐ Evolution
- ☐ Quality criteria

- Open
- Closed

vp_edgetype is a closed vp, as it can take only ‘true‘ or ‘false‘ values

Characteristic Properties of Variable Parts

- Logical relation
- Binding time
- Defaults
- Granularity
- Evolution
- Quality criteria

- Preplanning effort
- Visibility of variation point
- Information hiding
- Uniformity
- Separation of Concerns (SoC)
- Traceability
- Scalability

Using strategy pattern requires more Preplanning effort than parameters

Classifications of Techniques

1. Based on the emergence time
   - Traditional (*e.g.*, inheritance, generic types, design patterns)
   - Emerging (*e.g.*, frames, feature modules, delta modules)
2. Based on language or tool support
   - Language-based (*e.g.*, inheritance, feature modules, aspects)
   - Tool-based (*e.g.*, frames)
3. Based on how the variability is represented and resolved
   - Annotative (*e.g.*, preproceessor directives, frames)
   - Compositional (*e.g.*, feature modules, delta modules, frames)
     - Positive or Negative variability (*e.g.*, delta modules)

Classifications of Techniques

1. Based on the emergence time
   - Traditional (*e.g.*, inheritance, generic types, design patterns)
   - Emerging (*e.g.*, frames, feature modules, delta modules)
2. Based on language or tool support
   - Language-based (*e.g.*, inheritance, feature modules, aspects)
   - Tool-based (*e.g.*, frames)
3. Based on how the variability is represented and resolved
   - Annotative (*e.g.*, preproceessor directives, frames)
   - Compositional (*e.g.*, feature modules, delta modules, frames)
     - Positive or Negative variability (*e.g.*, delta modules)

Classifications of Techniques

1. Based on the emergence time
   - Traditional (*e.g.*, inheritance, generic types, design patterns)
   - Emerging (*e.g.*, frames, feature modules, delta modules)
2. Based on language or tool support
   - Language-based (*e.g.*, inheritance, feature modules, aspects)
   - Tool-based (*e.g.*, frames)
3. Based on how the variability is represented and resolved
   - Annotative (*e.g.*, preproceessor directives, frames)
   - Compositional (*e.g.*, feature modules, delta modules, frames)
     - Positive or Negative variability (*e.g.*, delta modules)

Catalog Building Method

Covered techniques: Used in a closed-world SPLE process

Excluded techniques: Components, frameworks, ...

Evaluation of techniques

- *First process:* Use 4 small case studies (in Scala)
- *Second process:* An Informed opinion from the existing research works (catalogs, taxonomies, studies,...)

Resulting catalog...

Legend A:
●: good support / belong
◐: possible support (difficult)
○: no support (not often applicable) / does not belong
✱: high; ◖ : average; ◁ : low
E: explicit; A: ambiguous

	Feature types			Binding time			Granularity																
	Optional	*Or*	*Alternative*	*Static (S)*	*Dynamic (D)*	Defaults	Open for evolution	*Coarse*	*Medium*	*Fine*	Preplanning effort	Visibility of *vp-s*	Information hiding	Uniformity	Sep. of concerns	Traceability	Scalability	Language paradigm	Annotative	Compositional	Language-based	Tool-based	Traditional	Emerging
AD-HOC REUSE																								
Cloning / Patching	●	●	●	● 7	○ 7	○ 7	● 7	● 7	●	●	◁	A/E 7	○ 7	●	○ 7	○ 7	○ 7	Not specific 7	●	●	●	○	●	○
Conditional Execution (Parameters)	● 7	○ 7	● 7	◐ 1,7	● 1,7	○ 7	○ 7	○ 1,7	◐ 1,7	● 1,7	◁ 1	A 7	◐ 1	○ 1	◐ 1,7	◐ 1,7	○ 1	Not specific 7	● 1	○ 1	● 1	○ 1	● 1	○ 1
METHODOLOGICAL REUSE																								
Preprocessor directives	● 2,3,4,7	◐ 2,4	● 1,2,3,4,7	● 1,2,7	○ 1,2,7	◐ 6,7	○ 7	● 1,3,4,6,7	● 1,3,4,6,7	● 1,3,4,6,7	◁ 1,7	A/E 7	○ 1	● 1	○ 2,7	◐ 1,2	◐ 2,7	Not specific 7	● 1,2,4,6	○ 1,4,6	○ 1,3,4,6	● 1,3,4,6	● 1,3	○ 1,3
Argument defaulting	● 4	○ 4	○ 4	●	○	● 6	○	○	●	○	◖	A	◐	○	○	◐	◐	Not specific	● 3,4,6	○ 3,4,6	●	○	● 3	○ 3
Overriding	◐	●	○	○	●	●	●	○	●	○	◁	A	●	○	●	●	●	O. Oriented	○	●	●	○	●	○
Aggregation / Delegation	● 2	◐ 2	◐ 2	● 2	● 2	●	◐	●	●	○	◖	E	◐	○	● 2	◐ 2	◐ 2	O. Oriented 2	● 2	● 2	●	○	●	○
Inheritance	● 2	● 2	◐ 2	● 2,6	● 2,6	●	●	● 6	● 6	○ 6	◖	A	● 6	○ 1	● 2	● 2	◐ 2	O. Oriented 6	◐ 2,6	● 2,6	● 6	○ 6	●	○
Reflections	● 2	● 2	● 2	◐ 2	● 2	●	◐	●	●	○	✱	E	●	○	● 2	● 2	● 2	O. Oriented	○	●	●	○	●	○
Aspects	◐ 2,4	● 2,4	◐ 2,4	● 1,2,7	◐ 1,2,7	● 7	● 7	◐ 1,7	● 1,7	○ 1,7	◖ 1,7	A 1,7	○ 1	◐ 1	● 1,3,7	● 1	● 2,7	Aspect Ori. 1,3	◐ 2	● 1,2,3	● 1,3,7	○ 1,3,7	○ 1,3	● 1,3
Polymorphism																								
Coercion (Casting)	◐ 4	◐ 4	◐ 4	●	○	● 4	●	●	○	○	✱	A	●	○	●	●	●	O. Oriented 3,4	○	●	●	○	● 3	○ 3
Overloading	○ 2,4	○ 2,4	● 2,4	● 2,6	○ 2,6	●	●	○ 6	● 6	○ 6	◖	A	●	○	◐ 2	◐ 2	◐ 2	Procedural 3,4	● 2,6	● 2,6	●	○	● 3	○ 3
Subtype polymorphism	◐ 4,7	◐ 3,4	● 3,4,7	◐	● 3,4,7	● 6	● 7	● 7	● 7	○ 7	✱	A 7	●	○	● 7	●	●	O. Oriented 3,4	○	● 7	● 7	○ 7	● 3	○ 3
Parametric polymorphism (generics)	◐ 3,4,7	● 3,4	● 3,4,7	● 3,4,6,7	○ 3,4,6,7	○ 7	○/●	● 6	● 6,7	○ 6,7	✱	E 6,7	●	○	◐	●	◐	Generic prog. 3,4	● 6	● 7	● 4,7	○ 4,7	● 3	○ 3
Design patterns																								
Strategy pattern	● 1	● 1	● 1	◐ 6	● 1,6	● 1	●	●	● 1	○	✱ 1	A	● 1	○ 1	● 1	● 1	●	O. Oriented	○ 1	● 1	● 1	○ 1	● 1	○ 1
Decorator pattern	● 1	● 1	○	◐	● 1	●	●	●	●	○	✱ 1	A	● 1	○ 1	● 1	● 1	●	O. Oriented	○	●	●	○	●	○
Observer pattern	●	●	●	◐	● 1	●	●	●	●	○	✱ 1	A	● 1	○ 1	● 1	● 1	●	O. Oriented	● 1	● 1	● 1	○ 1	● 1	○ 1
Template method pattern	● 1	◐ 1	● 1	◐ 6	● 1,6	● 1	●	●	● 6	○	✱ 1	A	◐ 1	○ 1	● 1	● 1	●	O. Oriented	○ 1	● 1	● 1	○ 1	● 1	○ 1
Visitor pattern	◐	●	◐	◐	●	●	◐	●	●	○	✱	A	●	○	●	●	●	O.Oriented	○	●	●	○	●	○
Emerging techniques																								
Frames	● 2,5	● 2,5	● 2,5	● 2,5,7	○ 2,5,7	● 5,7	○/● 7	● 5,7	● 5,7	● 5,7	◖ 2,7	E 5,7	○	◐ 1	● 2,7	● 2	◐ 2,7	Not specific 5,7	● 2,5	● 2,5	○ 5,7	● 5,7	○	●
Feature Modules	◐ 4	● 1,3,4	◐ 4	● 1	◐ 1	●	●	● 1	● 1	○ 1	◖ 1	A	○ 1	● 1	● 1,3	● 1	●	Feature Ori. 1	○ 1	● 1,3	● 1,3	○ 1,3	○ 1,3	● 1,3
Delta Modules	◐	●	◐	● 1	○	●	●	● 1	● 1	○ 1	◖	A	○	◐	●	●	●	Delta Ori. 1	● 1	● 1	● 1	○ 1	○ 1	● 1

Legend B: 1 → Apel [2]; 2 → Gacek [13]; 3 → Muthig [22]; 4 → Patzke [25]; 5 → Patzke [26]; 6 → Coplien [9]; 7 → Patzke [24]

Capturing Variability

Feature Modules

```
layer BasicGraph;
class Graph {
  Vector nodes = new Vector();
  Vector edges = new Vector();
  Edge add(Node n, Node m) {
    Edge e = new Edge(n, m);
    nodes.add(n);
    nodes.add(m);
    nodes.add(e);
    return e;
  } /* ... */
class Edge {   /* ... */ }
class Node {   /* ... */ }
}

layer Directed;
 class Graph { /* ... */ } /* ... */

layer Undirected;
class Graph { /* ... */ } /* ... */

layer Weighted;
class Graph { /* ... */ }
class Edge { /* ... */ }
class Weight { /* ... */ }
```

Strategy pattern with Parameters

```
object Conf {
  final val WEIGHTED: Boolean = true
}
abstract class Graph { /* Common part */ }
class ConcreteGraph extends Graph {
  def adddirectededge(s: Vertex, d: Vertex, w: Int) = {
    val edge = new Edge(s, d)
    if (Conf.WEIGHTED) {
      edge.weight = w
    }
    edges = edge :: edges
    addtoadjacencymatrix(edge)
  }
  def addundirectededge(s: Vertex, d: Vertex, w: Int) = {
    val edge1 = new Edge(s, d)
    val edge2 = new Edge(d, s)
    if (Conf.WEIGHTED) {
      edge1.weight = w
      edge2.weight = w
    }
    edges = edge1 :: edges
    edges = edge2 :: edges
    addtoadjacencymatrix(edge1)
    addtoadjacencymatrix(edge2)
  }
  def addedge(callback: (Vertex, Vertex, Int) => Unit,
      x: Vertex, y: Vertex, w: Int = 1) = callback(x, y, w)
```

Capturing Features

Logical Relation: BasicGraph (mandatory), Directed, Undirected, Weighted (optional)

Binding time: deployment; **Granularity**: feature module; No **Evolution** or **Default** concepts (Unweighted is default)

Capturing Variability

Capturing VP-s with Variants

VP-s	Lines	Granularity	Binding time	Logical Rl.	Evolution
`vp_edgetype`	26 – 27	method	runtime	alternative	Open
`vp_weight`	2	parameter	programming	alternative	Close

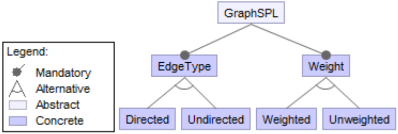
Variants	Lines	Granularity	Default	*VP*-s
`v_directed`	6 – 13	method	No	`vp_edgetype`
`v_undirected`	14 – 25	method	No	`vp_edgetype`
`v_weighted`	2	value	No	`vp_weight`
`v_unweighted`	2	value	Yes	`vp_weight`

Legend A:
●: good support / belong
◐: possible support (difficult)
○: no support (not often applicable) / does not belong
✱: high; ◖ : average; ◁ : low
E: explicit; A: ambiguous

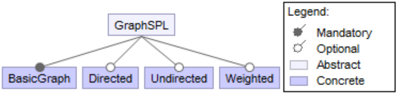
	Feature types			Binding time				Granularity																
	Optional	*Or*	*Alternative*	*Static (S)*	*Dynamic (D)*	Defaults	Open for evolution	*Coarse*	*Medium*	*Fine*	Preplanning effort	Visibility of *vp*-s	Information hiding	Uniformity	Sep. of concerns	Traceability	Scalability	Language paradigm	Annotative	Compositional	Language-based	Tool-based	Traditional	Emerging
Design patterns																								
Strategy pattern	●	●	●	◐	●	●	●	●	●	○	✱	A	●	○	●	●	●	O. Oriented	○	●	●	○	●	○
	1	1	1	6	1,6	1			1		1		1	1	1	1			1	1	1	1	1	1

Capturing Variability

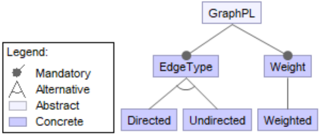
Reconstructing the Feature Model



Capturing Features



Capturing VP-s with Variants



Summary and Future Work

Addressed issues

- Capture the variability implementation techniques (**I1**)
- Both features and vp-s with variants can be used to capture the variability; their meaning overlap but is not the same (**I2**)
- We study the diverse properties that can be captured during reverse engineering (**I3**)

Summary and Future Work

Addressed issues

- Capture the variability implementation techniques (**I1**)
- Both features and vp-s with variants can be used to capture the variability; their meaning overlap but is not the same (**I2**)
- We study the diverse properties that can be captured during reverse engineering (**I3**)

Availability Case studies and a DSL:
https://github.com/ternava/variability-cchecking

Future Work

- Using vp-s with variants during the migration of some product variants as an SPL
- Demonstrate the usage of the catalog