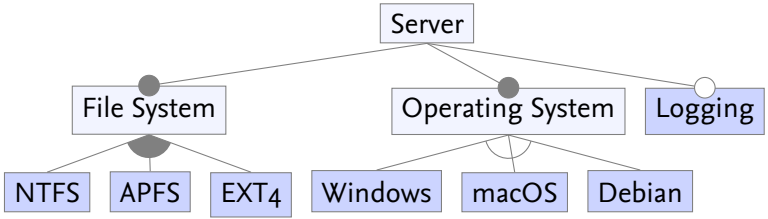
Technische Universität Braunschweig

Towards a Universal Variability Language: Design Tradeoffs

Dominik Engelhardt, Thomas Thüm, February 4, 2020

Institut für Softwaretechnik und Fahrzeuginformatik

Example



Legend:

- Abstract Feature
- Concrete Feature
- Mandatory
- Optional
- Or Group
- Alternative Group

Windows $\Longrightarrow$ NTFS
macOS $\Longrightarrow$ APFS

Feature Description Language (FDL)

```
Server: all (FileSystem, OperatingSystem, logging?)
FileSystem: more-of (ntfs, apfs, ext4)
OperatingSystem: one-of (windows, macOS, debian)

windows requires ntfs
macOS requires apfs
```

GUIDSL Grammars

```
// grammar
Server: FileSystem+ OperatingSystem [Logging];
FileSystem: NTFS | APFS | EXT4;
OperatingSystem: Windows | macOS | Debian;

// constraints
Windows implies NTFS;
macOS implies APFS;
```

Variability Specification Language (VSL)

```
featureModel FM {
    Server! (
        FileSystem! ( [1..*] (
            NTFS, APFS, EXT4
        )),
        OperatingSystem! ( [1] (
            Windows, macOS, Debian
        )),
        Logging?
    );
    link Windows = needs => NTFS;
    link macOS = needs => APFS;
}
```

Simple XML Feature Model (SXFM)

```
<feature_model name="FM"><feature_tree>
    :r Server (id_srv)
        :m FileSystem (id_fs)
            :g [1,*]
                : NTFS (id_ntfs)
                : APFS (id_apfs)
                : EXT4 (id_e4)
        :m OperatingSystem (id_os)
            :g [1,1]
                : Windows (id_win)
                : macOS (id_mac)
                : Debian (id_deb)
        :o Logging (id_log)
</feature_tree><constraints>
    c1: ~id_win or id_ntfs
    c2: ~id_mac or id_apfs
</constraints></feature_model>
```

FAMILIAR

```
FM (
    Server : FileSystem OperatingSystem [Logging];
    FileSystem : (NTFS | APFS | EXT4)+;
    OperatingSystem : (Windows | macOS | Debian);

    (!Windows | NTFS);
    (!macOS | APFS);
)
```

Text-based Variability Language (TVL)

```
root Server {
    group allOf {
        FileSystem {
            group someOf {
                NTFS, APFS, EXT4
            }
        },
        OperatingSystem {
            group oneOf {
                Windows, macOS, Debian
            }
            Windows requires NTFS;
            macOS requires APFS;
        },
        opt Logging
    }
}
```

μTVL

```
root Server {
    group allOf {
        FileSystem {
            group someOf {
                NTFS, APFS, EXT4
            }
        },
        OperatingSystem {
            group oneOf {
                Windows {require NTFS;}, macOS {require APFS;}, Debian
            }
        },
        opt Logging
    }
}
```

Clafer

```
Server
    or FileSystem
        NTFS
        APFS
        EXT4
    xor OperatingSystem
        Windows
        macOS
        Debian
        [Windows => NTFS;]
        [macOS => APFS;]
    Logging?
```

VELVET

```
concept Server {
    mandatory feature FileSystem {
        someOf { feature NTFS; feature APFS; feature EXT4;}
    }
    mandatory feature OperatingSystem {
        oneOf {feature Windows; feature macOS; feature Debian;}
    }
    feature Logging;
    constraint Windows -> NTFS;
    constraint macOS -> APFS;
}
```

INDENICA Variability Modeling Language (IVML)

```
project Server {
    compound Server {
        enum FileSystem {NTFS, APFS, EXT4};
        enum OperatingSystem {Windows, macOS, Debian};

        Boolean logging;

        FileSystem fileSystem;
        OperatingSystem operatingSystem;
        operatingSystem == Windows implies fileSystem == NTFS;
        operatingSystem == macOS implies fileSystem == APFS;
    }
}
```

PyFML

```
FM = Server (1..1): all [
        FileSystem (1..1): moreof [
            NTFS (0..1),
            APFS (0..1),
            EXT4 (0..1)
        ],
        OperatingSystem (1..1): oneof [
            Windows (0..1),
            macOS (0..1),
            Debian (0..1)
        ],
        Logging (0..1)
    ];
Windows implies NTFS;
macOS implies APFS;
```

Variability Modeling

```
Relationships:
Server {
    FileSystem {
        someOf {
            NTFS APFS EXT4
        }
    }
    OperatingSystem {
        oneOf {
            Windows macOS Debian
        }
    }
    ? Logging
}
Constraints:
Windows requires NTFS
macOS requires APFS
```

Summary

```
root Server {
    group allOf {
        FileSystem {
            group someOf {
                NTFS, APFS, EXT4
            }
        },
        OperatingSystem {
            group oneOf {
                Windows, macOS, Debian
            }
            Windows requires NTFS;
            macOS requires APFS;
        },
        opt Logging
    }
}
```

```
concept Server {
    mandatory feature FileSystem {
        someOf { feature NTFS; feature APFS; feature EXT4;}
    }
    mandatory feature OperatingSystem {
        oneOf {feature Windows; feature macOS; feature Debian;}
    }
    feature Logging;
    constraint Windows -> NTFS;
    constraint macOS -> APFS;
}
```

```
featureModel FM {
    Server! (
        FileSystem! ( [1..*] (
            NTFS, APFS, EXT4
        )),
        OperatingSystem! ( [1] (
            Windows, macOS, Debian
        )),
        Logging?
    );
    link Windows = needs => NTFS;
    link macOS = needs => APFS;
}
```

```
Relationships:
Server {
    FileSystem {
        someOf {
            NTFS APFS EXT4
        }
    }
    OperatingSystem {
        oneOf {
            Windows macOS Debian
        }
    }
    ? Logging
}
Constraints:
Windows requires NTFS
macOS requires APFS
```

```
FM = Server (1..1): all [
        FileSystem (1..1): moreof [
            NTFS (0..1),
            APFS (0..1),
            EXT4 (0..1)
        ],
        OperatingSystem (1..1): oneof [
            Windows (0..1),
            macOS (0..1),
            Debian (0..1)
        ],
        Logging (0..1)
    ];
Windows implies NTFS;
macOS implies APFS;
```

```
root Server {
    group allOf {
        FileSystem {
            group someOf {
                NTFS, APFS, EXT4
            }
        },
        OperatingSystem {
            group oneOf {
                Windows {require NTFS;}, macOS {require APFS;}, Debian
            }
        },
        opt Logging
    }
}
```

```
<feature_model name="FM"><feature_tree>
    :r Server (id_srv)
        :m FileSystem (id_fs)
            :g [1,*]
                : NTFS (id_ntfs)
                : APFS (id_apfs)
                : EXT4 (id_e4)
        :m OperatingSystem (id_os)
            :g [1,1]
                : Windows (id_win)
                : macOS (id_mac)
                : Debian (id_deb)
        :o Logging (id_log)
</feature_tree><constraints>
    c1: ~id_win or id_ntfs
    c2: ~id_mac or id_apfs
</constraints></feature_model>
```

```
project Server {
    compound Server {
        enum FileSystem {NTFS, APFS, EXT4};
        enum OperatingSystem {Windows, macOS, Debian};

        Boolean logging;

        FileSystem fileSystem;
        OperatingSystem operatingSystem;
        operatingSystem == Windows implies fileSystem == NTFS;
        operatingSystem == macOS implies fileSystem == APFS;
    }
}
```

```
// grammar
Server: FileSystem+ OperatingSystem [Logging];
FileSystem: NTFS | APFS | EXT4;
OperatingSystem: Windows | macOS | Debian;

// constraints
Windows implies NTFS;
macOS implies APFS;
```

```
FM (
    Server : FileSystem OperatingSystem [Logging];
    FileSystem : (NTFS | APFS | EXT4)+;
    OperatingSystem : (Windows | macOS | Debian);

    (!Windows | NTFS);
    (!macOS | APFS);
)
```

```
Server: all (FileSystem, OperatingSystem, logging?)
FileSystem: more-of (ntfs, apfs, ext4)
OperatingSystem: one-of (windows, macOS, debian)

windows requires ntfs
macOS requires apfs
```

```
Server
    or FileSystem
        NTFS
        APFS
        EXT4
    xor OperatingSystem
        Windows
        macOS
        Debian
        [Windows => NTFS;]
        [macOS => APFS;]
    Logging?
```

Concepts and Design Dimensions

Expressivity (not Expressive Power)

Expressivity: How much meaning is conveyed? How much verbosity is there? How concise is the code?

- Make assumptions about target domain
- Encapsulate knowledge about the domain in the language

```
feature Server
```

vs.

```
Feature
{
    string {"Server"}
}
```

Separation of Concerns (SOC)

2 extremes:

all information in one place

```
Server
    or FileSystem
        NTFS
        APFS
        EXT4
    xor OperatingSystem
        Windows
        macOS
        Debian
        [Windows => NTFS;]
        [macOS => APFS;]
    Logging?
```

vs.

separate files or sections within a file for everything

```
// feature list
Server
FileSystem
NTFS
APFS
...
```

```
// containtment
Server -> FileSystem
Server -> OperatingSystem
FileSystem -> NTFS
...
```

```
// constraints
Windows => NTFS;
macOS => APFS;
...
```

...

Completeness

- Can all existing formats (+extra information) be expressed in the language?
- How to deal with incompleteness?
  - Transformation into simpler constructs
  - Mix of languages?

Scope

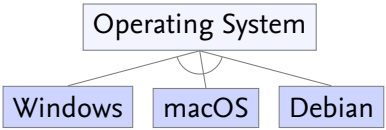
- Which language features should be available?
- Language-levels? Which features per level?
- Modularity of the language?
- How to control growth/evolution?

Composability

- Real-world systems are huge
- FMs that are too big hard to understand, evolve and analyze
- Interfaces and composition for models?

Examples and Feedback

Keyword Length



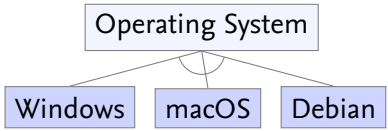
long	abbreviated	symbols
`feature OperatingSystem`	`f OperatingSystem`	`OperatingSystem`
`alternative NTFS`	`a NTFS`	`: NTFS`
`alternative APFS`	`a APFS`	`: APFS`
`alternative EXT4`	`a EXT4`	`: EXT4`

Line-breaks and long lines

semicolon	continued lines

`abstract`
`  mandatory`
`  feature`
`  OperatingSystem;`
`  alternative`
`  feature`
`  NTFS;`
	`abstract \`
	`  mandatory \`
	`  feature \`
	`  OperatingSystem`
	`  alternative \`
	`  feature \`
	`  NTFS`

Structuring



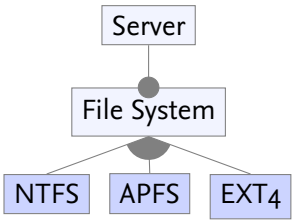
whitespace	curly braces	parentheses

`feature OS`
`  alternative`
`  feature Windows`
`  feature macOS`
`  feature Debian`
	`feature OS`
	`  alternative {`
	`  feature Windows {},`
	`  feature macOS {},`
	`  feature Debian {}`
	`}`
		`(feature OS`
		`  (alternative`
		`  (feature Windows)`
		`  (feature macOS)`
		`  (feature Debian)))`

Hierarchy – Embedded or by Reference?

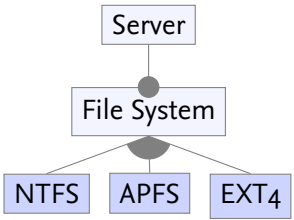
embedded	by reference

`Server`
`  or FileSystem`
`  NTFS`
`  APFS`
`  EXT4`
	`Server: FileSystem`
	`FileSystem: (NTFS | APFS | EXT4)+`



Location of Groups

at parent	on children	in-between
Server or FileSystem NTFS APFS EXT4	Server FileSystem or NTFS or APFS or EXT4	Server FileSystem or NTFS APFS EXT4



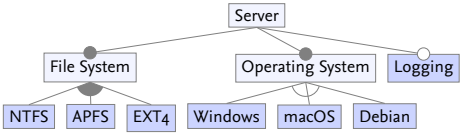
Multiple Groups Beneath One Parent?

With the last two options, the model could be restructured:

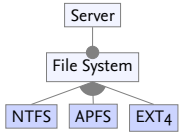
on children	in-between

```
Server
  or NTFS
  or APFS
  or EXT4
  alt Windows
  alt macOS
  alt Debian
  opt Logging
```

```
Server
  or
    NTFS
    APFS
    EXT4
  alt
    Windows
    macOS
    Debian
  opt
    Logging
```



Groups vs. Cardinality vs. Constraints

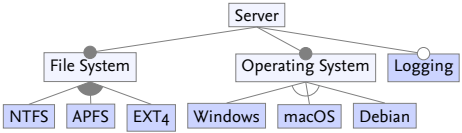


groups	cardinality	constraints

Server
  or FileSystem
  NTFS
  APFS
  EXT4
	Server
	  [1..*] FileSystem
	  NTFS
	  APFS
	  EXT4
		Server
		  FileSystem
		  NTFS
		  APFS
		  EXT4
		  [children.count > 0]

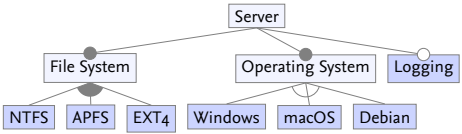
Example 1 - dense syntax, structure + data by reference

```
Server: (FileSystem!, OperatingSysetm!, Logging);
FileSystem: [NTFS, APFS, EXT4];
OperatingSystem: {Windows, macOS, Debian};

Windows => NTFS;
macOS => APFS;

defaults Debian, EXT4;
abstract Server, FileSystem, OperatingSystem;
```



Example 2 - explicit structure, data in one place

```
abstract feature Server {
    abstract mandatory feature FileSystem {
        alternative feature NTFS {}
        alternative feature APFS {}
        alternative default feature EXT4 {}
    }
    abstract mandatory feature OperatingSystem {
        or feature Windows {
            constraint {Windows requires NTFS}
        }
        or feature macOS {
            constraint {macOS requires APFS}
        }
        or default feature Debian {}
    }
    optional feature Logging {}
}
```



Scope

Which language features should be supported?

- default selection for configurations?
- abstract features?
- save entire configurations?
- attributes of features?
- composition of feature models? namespaces? interfaces or visibility modifiers?
- expressive power of constraints? Propositional, first-order, higher-order?

Separation of Concerns (SOC)

Which information should be in-line vs. in separate files/fragments?

- structure
- abstract?
- constraints
- configurations
- default selection
- arbitrary additional data, e.g. layouting

What is a reasonable tradeoff? Where to cut?

Use or adapt existing serialization format?

- Parsers and basic tool integrations exist
- Made for generic data, often unnecessarily verbose
- No checks for correct application (unless schemas are used)
- No syntactic support for specific concepts like constraints
- Examples: yaml, json, edn, openddl

IDE support? Who wants to use a text editor?

- Tradeoff usability vs. tool lock-in
- Projectional editors offer representations for different concerns, but no portability
- Classic parser can have EBNF spec for theoretical portability and small default library
- Most language workbenches allow to export small independent language core

What did we miss? More comments?

Questionnaire

https://tinyurl.com/touvala

Backup Slides

Composition of Feature Models

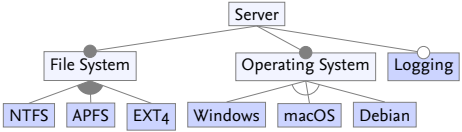
references	refs + visibility modifiers	refs + interface FM

`import Server.fm`
`Application`
`  Logging?`
`  ref Server as Host`
	`Server`
	`  private FileSystem`
	`  public OS`
	
	`import Server.fm`
	`Application`
	`  Logging?`
	`  ref Server.OS`
		`import Server.interface`
		`Application`
		`  Logging?`
		`  ref Server.OS`
		
		`//interface FM`
		`Server`
		`  OS`

Example 3

```
import FileSystem as FS
features
  abstract Server
    mandatory
      include FS
      abstract OperatingSystem
        or
          Windows
          macOS
          default Debian
    Logging

constraints
  Windows => NTFS
  macOS => APFS
```

```
// FileSystem.uvl
features
  abstract FileSystem
    alternative
      NTFS
      APFS
      default EXT4
```



Example 4

```
import FileSystem as FS

features
Server: include FS! OperatingSystem! Logging
abstract OperatingSystem: (Windows | macOS | Debian)+
default Debian:

constraints
Windows => NTFS
macOS => APFS

// FileSystem.uvl
features
FileSystem: NTFS | APFS | EXT4
default EXT4
```