Understanding Software Variability in Software Ecosystems

Krzysztof Czarnecki
University of Waterloo
Canada

http://gsd.uwaterloo.ca
Joint work with

Thorsten Berger (U Leipzig)
Steffen Dienst (U Leipzig)
Rafael Lotufo (U Waterloo)
Marko Novakovic (U Waterloo)
Leonardo Passos (U Waterloo)
R.-Helge Pfeiffer (ITU)
Steven She (U Waterloo)
Reinhard Tartler (U Erlangen)
Andrzej Wasowski (ITU)
Yingfei Xiong (U Peking)
Variability is everywhere
Variability adds complexity
Variability Handling

Build independently

Clone & own

Share assets
Software Product Lines

- Variability modeling
- Product configuration
- Components
- Services
- DSLs
- Generators
- Design patterns
- …
Which variability techniques are effective in practice?
Under what conditions?
Variability Handling in Open Source Software Ecosystems
Software ecosystems are communities of developers and users supported by a shared platform and building on each others solutions
Introductions...

Linux Kernel
- General-purpose OS kernel

eCos
- Embedded OS

Debian
- Complete OS plus apps

Eclipse
- Customizable IDEs

Android
- Mobile OS plus apps
Each one has vast variability

- Linux Kernel: 10k Config options
- eCos: 2.8k Config options
- Debian: 28k Packages
- Eclipse: >7k Bundles
- Android: >400k Apps
Each handles variability differently

Linux Kernel

eCos

Debian

Eclipse

Android

Feature models

Manifests-based packages

Service-oriented apps
Key Findings
Two Opposing Approaches to Variability

Variability management

Controlling scope
Eliminating variants if no significant business value added

Variability encouragement

Unleashing community innovation
Encouraging competition
Letting community decide scope
Variability Management vs. Encouragement

- Mainline Kernel
- Free edition
- Debian Archive
- Platform release
- Android OS
- Eclipse Market-place
- Google Play store

Centralized Variability Management
Variability Encouragement
Feature Models

Proposed by Kang et al., 1990

A. van Deursen and P. Klint. Domain-Specific Language Design Requires Feature Descriptions, JCIT, 2002

Google Scholar has over 3k papers on feature modeling
Feature Models in Practice

Developed independently in open source – Linux, eCos

- Journaling Flash File System 2
- Debug Level: int
- Compress Data
  - Support ZLIB
  - Default Compression
    - None
    - Priority
    - Size
  - option
  - alternatives

Support ZLIB $\Rightarrow$ ZLIB Inflate
JFFS2 $\Rightarrow$ CRC & MTD
$0 \leq$ Debug Level $\leq 2$
CDL Snippet

cdl_component KERNEL_SCHED_MLQUEUE {
    display "Multi—level queue scheduler"
    default_value 1
    implements KERNEL_SCHEDULER
    description "The multi—level queue scheduler supports multiple priority levels and multiple threads at each priority level..."
}

cdl_option TRACE_TIMESLICE {
    display "Output timeslices when tracing"
    active_if USE_TRACING
    requires !DEBUG_TRACE_ASSERT_SIMPLE
    ...
    }
}

cdl_option AT91_CLOCK_SPEED {
    display "CPU clock speed"
    calculated { AT91_CLOCK_OSC_MAIN * AT91_PLL_MULTIPLIER / AT91_PLL_DIVIDER / 2 }
    legal_values { 0 to 220000000 }
    flavor data
    }

eCos Configurator

Compression and decompression are entirely handled by the file system and are fully transparent to applications. However, selecting this option increases the amount of RAM required and slows down read and write operations considerably if you have a slow CPU.
Beyond Feature Modeling

• Concepts for scalability
  – Visibility
  – Modularization
  – Derived defaults / derived features

• Expressive constraints
  – Kconfig: Three-state logic (follows Kleene’s rules) for binding mode
  – CDL: Comparison, arithmetic and String operators

• Domain-specific vocabulary
  – E.g., package, component, option
Feature Types in Systems Software

The diagram illustrates the distribution of feature types among various software projects. The x-axis represents the different software projects, and the y-axis represents the percentage of features. The features are categorized into three types:

- **switch**: Represented by the light blue section of the bars.
- **data (number)**: Represented by the dark blue section of the bars.
- **data (string)**: Represented by the green section of the bars.
- **none**: Represented by the yellow section of the bars.

From the diagram, we can observe the relative proportion of each feature type across the different software projects.
Non-Boolean Constraints in eCos

\[
(1 \leq \left( \left( \left( RTC\_NUMERATOR\_data \times \left( \left( OSC\_MAIN\_data \times PLL\_MULTIPLIER\_data \right) / PLL\_DIVIDER\_data \right) / 2 \right) \right) / (TIMER\_TC\_enabled \ ? 32 \ : 16) \right) / RTC\_DENOMINATOR\_data \right) / 10000000000
\]

=> Implications for configuration and analysis tools
Dependency Structures

Dependencies grow linearly with model size (eCos, Linux, and 10 other Kconfig-based systems)

Indicates that the feature-based architectures scale well
Correlations

Highly technical

Static, closed config

Variability Mgmt.

End user

Dynamic, open config

Variability Egmnt.
Feature modeling works well for static variability in engineering domains
Variability Management vs. Encouragement
See papers for details...


See gsd.uwaterloo.ca
Summary

Feature models seem to scale well in the embedded domain.

Open and dynamic ecosystems grow fast!

They rely on dynamic binding, runtime-service lookup, and easy download and installation.

Variability encouragement complements variability management as a future direction.