Kconfig metamodel: a first approach

GT LMV

Jolan Philippe

Jan 12th, 2026

Outline

Background

Configuration and variability

Model Driven Engineering (MDE)

Kconfig metamodel: a first approach

Perspectives

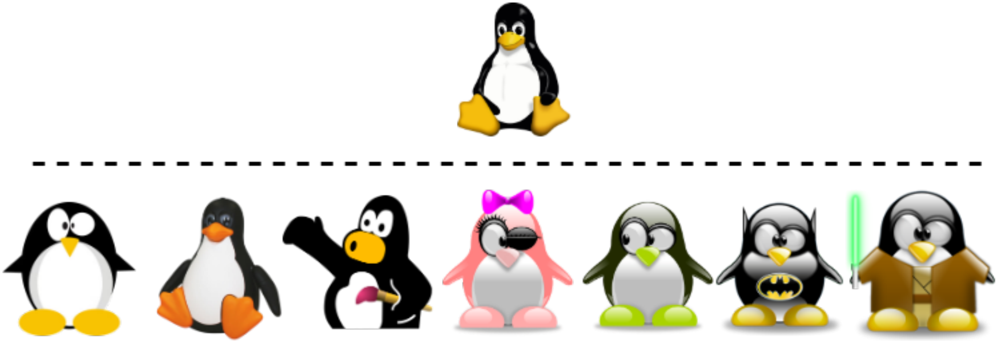
Background

Configuration and variability

- **Configuration** is the process of selecting values for a set of options to build a concrete system
- A **Software Product Line (SPL)** is an approach to:
  - Develop a family of related products
  - Share common assets while managing variability
- Each product corresponds to a **valid configuration** satisfying constraints and dependencies

Configuring the Linux Kernel

A motivating example: The Linux kernel



Configuring the Linux Kernel

A highly configurable kernel

- 19000 parameters
- Considering all boolean: 2^{19000} possible configuration
- Approximately: 10^{5720}

Configuring the Linux Kernel

A highly configurable kernel

- 19000 parameters
- Considering all boolean: 2^{19000} possible configuration
- Approximately: 10^{5720}

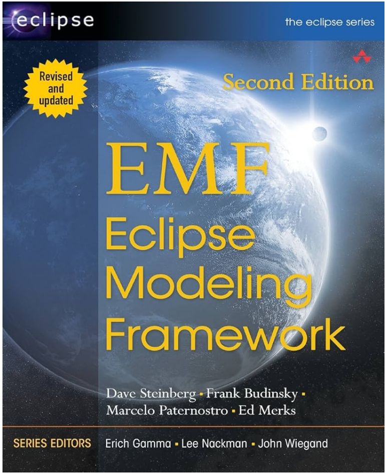
FYI, there are approx. 10^{80} atoms in the universe.

Community

- **1990s** Emergence of **Software Product Lines**
- **2001** Introduction of **Kconfig** in the Linux kernel
- **2010s** Renewed interest
  - Highly configurable systems
  - Variability modeling and automated analysis
- **2026** VARIABILITY
  - A merge from SPLC, VaMoS, ICSR

Model-Driven Engineering (MDE)

- **Models** are the central artifacts of the engineering process
- A model as an abstraction:
  - Describe
  - Analyze
  - Transform
- Models used at different levels:
  - **Software development models** (e.g., class models, relational models)
  - **Execution models** (e.g., Petri nets, sequence diagrams)


eclipse
the eclipse series
Revised and updated
Second Edition
EMF
Eclipse Modeling Framework
Dave Steinberg • Frank Budinsky • Marcelo Paternostro • Ed Merks
SERIES EDITORS
Erich Gamma • Lee Nackman • John Wiegand

Metamodeling

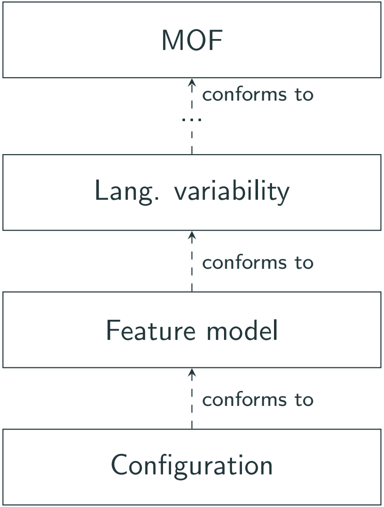
A **metamodel** gives structure rules for defining a frame for constructing models ... and is also a **model**.

Examples:

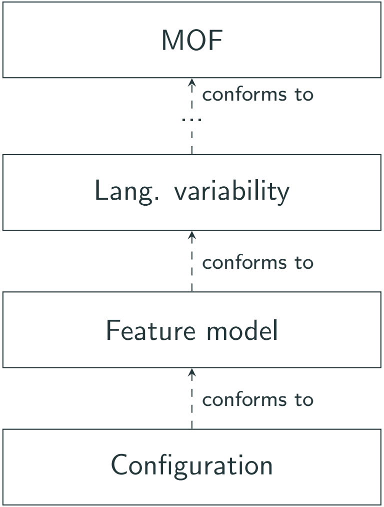
- A **variable** has a **type**. With a type: rules to write an instance
- A **program** written using a language follows a **grammar**
- A **class diagram** follows the **UML standard**

(And semantics)

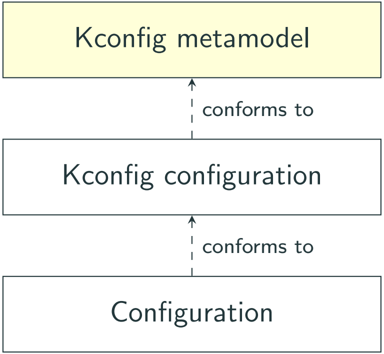
Variability metamodels



Variability metamodels



In the paper I will now talk about:



Kconfig metamodel: a first approach

Kconfig metamodel: a first approach

Context

- David Romero-Organvidez, Pablo Neira-Ayuso, José A. Galindo, David Benavides
- 28th ACM International Systems and Software Product Line Conference (SPLC 2024) - ranked B on CORE

Kconfig

- **Kconfig** stands for **Kernel + Config**
- Domain-specific language used to **configure the Linux kernel**
- Produces configuration files: `.config`
- Supports:
  - Boolean and tristate options (`y`, `n`, `m`)
  - Dependencies and conditional
  - Modular approach (with `menu`)
- Used by tools like:
  - `menuconfig`
  - `xconfig`
  - `nconfig`

Linux kernel configuration process

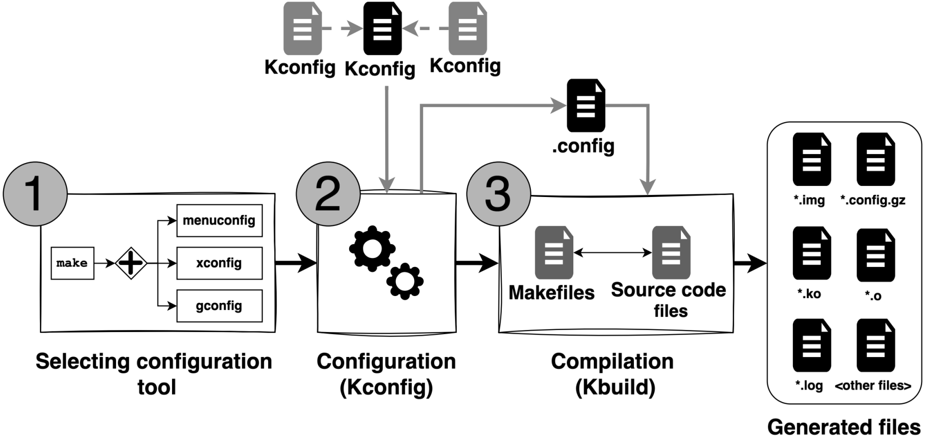


Figure 1: Steps in the Linux kernel configuration process

Config example

```
source "drivers/cpuidle/Kconfig"
...
config CLANG_SUPPORTS_DYNAMIC_FTRACE_WITH_ARGS
    bool
    default CC_IS_CLANG
    depends on AS_IS_GNU || (AS_IS_LLVM && (LD_IS_LLD || LD_VERSION >= 23600))
    select HAVE_DYNAMIC_FTRACE_WITH_ARGS
...
config NR_CPUS
    int "Maximum number of CPUs (2-4096)"
    range 2 4096
    default "512"
...
config ARCH_MMAP_RND_BITS_MAX
    int
    default 19 if ARM64_VA_BITS=36
    default 24 if ARM64_VA_BITS=39
```

Listing 1: `Kconfig` example for config types

General purpose

- The authors define a **metamodel** to formalize how Kconfig configurations are **described**
- It is **not** an instance of a kernel configuration
  - It specifies the *structure* and *allowed values*
  - Not a concrete `.config` file
- The metamodel provides a foundation to:
  - Leverage **Model-Driven Engineering (MDE)** tools (e.g., EMF)
  - Enable **bidirectional model transformations** between variability models
- Plan to support biderctional transformations:
  - Kconfig ↔ KFeatures
  - Kconfig ↔ UVL

Overall metamodel

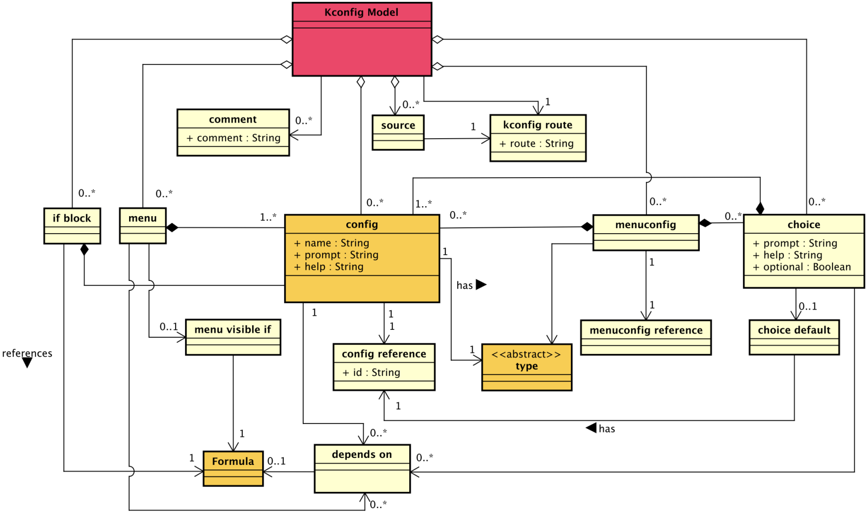


Figure 2: `Kconfig` metamodel

Config metamodel

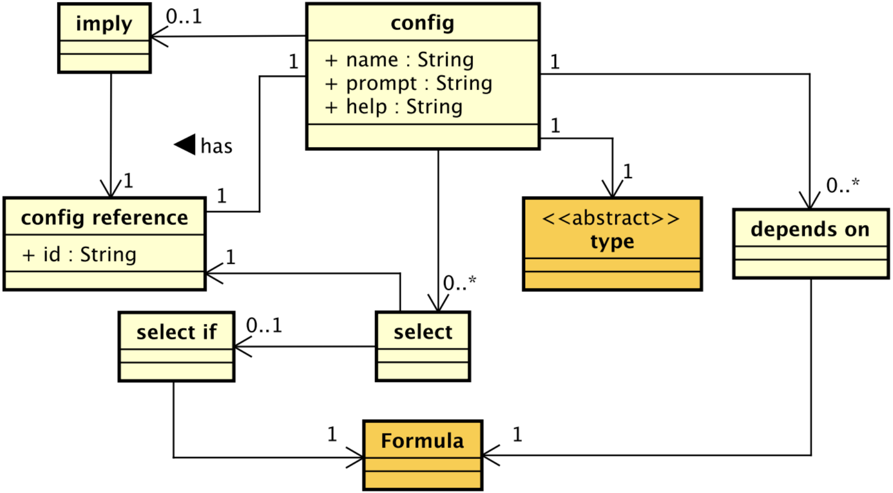


Figure 3: *Config* element metamodel

Dependencies

- *Depends on*: Pre-condition (with an optional boolean formula)
- *Select*: Activation of another config if current activated (with an optional boolean formula)
- *Imply*: Necessarily implication of another configuration

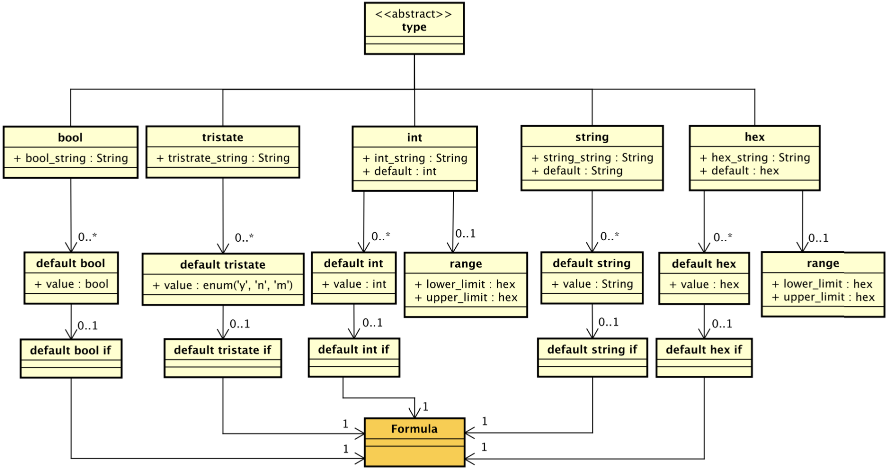
Config example

```
config A
    bool "You can select me as long as you have selected B"
    depends on B

config B
    bool "You can select me as long as you have not selected C"
    depends on !C
    imply D
    select E if F > 20
```

Listing 2: `Kconfig` example for config types

Types metamodel



<<abstract>>
type

bool
+ bool_string : String

tristate
+ tristrate_string : String

int
+ int_string : String
+ default : int

string
+ string_string : String
+ default : String

hex
+ hex_string : String
+ default : hex

default bool
+ value : bool

default tristate
+ value : enum('y', 'n', 'm')

default int
+ value : int

range
+ lower_limit : hex
+ upper_limit : hex

default string
+ value : String

default hex
+ value : hex

range
+ lower_limit : hex
+ upper_limit : hex

default bool if

default tristate if

default int if

default string if

default hex if

Formula

Figure 4: *Config type* metamodel

Formula metamodel

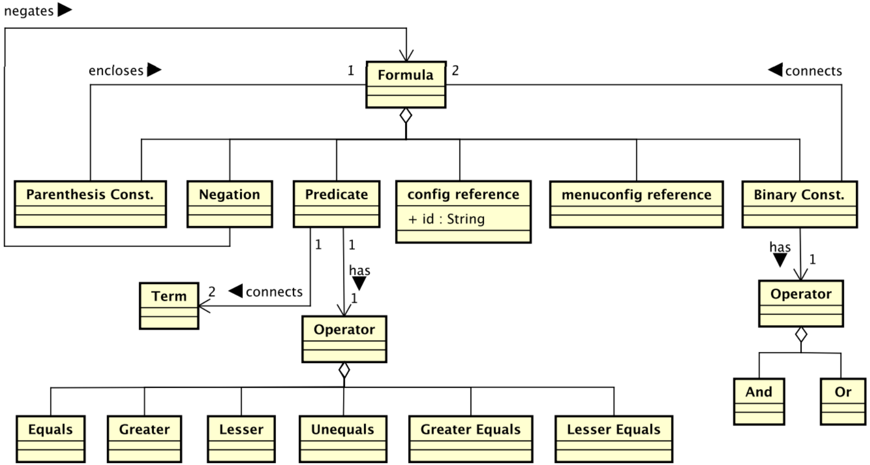


Figure 5: *Formula* metamodel

Perspectives

Perspectives

- Enable **reasoning about variability** in the context of **system reconfiguration**
- During reconfiguration:
  - A **target configuration** is selected
  - The system must ensure **properties** on the new configuration (consistency, constraints, safety)
- Leverage the metamodel to:
  - Define a **grammar** coupled with a formal **semantics**
  - Support automated reasoning and verification
- Initial focus on **UVL-oriented** semantics. Extend to:
  - A formal semantics for **Kconfig**
  - A correctness argument or proof for **Kconfig** → **UVL** transformations