About this Lecture

Problem Space
- Domain Expert
- Features and Dependencies

Solution Space
- Architect / Developer
- Architecture and Implementation

Implementation Techniques: Classification
- Decompositional Approaches
  - Text-based filtering (untyped)
  - Preprocessors
- Compositional Approaches
  - Language-based composition mechanisms (typed)
  - OOP, AOP, Templates
- Generative Approaches
  - Metamodel-based generation of components (typed)
  - MDD, C++ TMP, generators

Real-world software uses them all!
Agenda

5.1 Motivation
5.2 Variability in Linux
5.3 Configuration Consistency
5.4 Configuration Coverage
5.5 Automatic Tailoring
5.6 Summary
5.7 References

Typical Configurable Operating Systems...

320 optional, independent features

more variants than atoms in the universe!

1,250 features

one individual variant for each human being
Typical Configurable Operating Systems...

1,250 features

Challenges:
- How to maintain this?
- How to test this?
- Why so many features anyway?

12,000 features

The Linux Configuration and Generation Process

1. Configuration with an `CONFIG` frontend
2. Compilation of a subset of files
3. Selection of a subset of `CPP` Blocks
4. Linking of the kernel and loadable kernel modules

Dominancy and Hierarchy of Variability

- Feature Modelling: 12,000 features
- Coarse-grained: `KBUILD`: 31,000 source files
- Fine-grained: `CPP`: 89,000 `ifdef` blocks
- Language-level: `GCC`: `if(CONFIG_SMP) ...`
- Link-time: `LD`: branches in linker scripts
- Run-time: `INSMOD`, `MODPROBE`, ...

Variability Management in Operating Systems
Challenges with Implemented Variability

Central declaration of configurability: `CONFIG`
Distributed implementation of configurability: `MAKE`, `CPP`, `GCC`, `LD`

Problem Analysis: Configuration Consistency

Configuration

Implementation

Consistency?
Coverage?

Problem Analysis: Symbolic Inconsistency

Configuration

Implementation

Symbolic
Logic

config HOTPLUG_CPU
bool "Support for hot-pluggable CPUs"
depends on SMP & HOTPLUG:

static int hotplug_cfd(struct notifier_block *nfb, unsigned long action, void *hcpu)
{
    // [...
    switch (action) {
    case CPU_UP_PREPARE:
    case CPU_UP_PREPARE_FROZEN:
        // [...
    #ifdef CONFIG_CPU_HOTPLUG
    case CPU_UP_CANCELED:
    case CPU_UP_CANCELED_FROZEN:
    case CPU_DEAD:
    case CPU_DEAD_FROZEN:
        free_cpumask_var(cfd->cpumask);
        break;
    #endif
    }
    return NOTIFY_OK;
}
Problem Analysis: Logic Inconsistency

Feature DISCONTIGMEM implies feature NUMA

Inner blocks are not actually configuration-dependent
- Block 2 is always selected ▸ undead
- Block 3 is never selected ▸ dead

〜 Linux contains superfluous #ifdef Blocks!

Solution Approach: Consistency Validation

Problem and solution space are analyzed for configuration points:

\[
C = (\text{FLATMEM} \rightarrow \text{MEMORY MODEL}) \land (\text{DISCONTIGMEM} \rightarrow \text{MEMORY MODEL}) \land (\text{SPARSEMEM} \rightarrow \text{MEMORY MODEL}) \land (\text{NUMA} \rightarrow \text{MEMORY MODEL}) \land (\text{DISCONTIGMEM} \rightarrow \text{NUMA})
\]

\[
I = (\text{Block} 1 \leftrightarrow \text{DISCONTIGMEM}) \land (\text{Block} 2 \leftrightarrow \text{Block} 1 \land \text{NUMA}) \land (\text{Block} 3 \leftrightarrow \text{Block} 1 \land \neg \text{Block} 2)
\]

Implementation: The UNDERTAKER

Job: Find (and eventually bury) dead #ifdef-code!

We have found 1776 configurability defects in Linux v2.6.35

Submitted 123 patches for 364 defects

20 are confirmed new bugs (affecting binary code)

Cleaned up 5129 lines of cruft code
Implementation: The UNDERTAKER

Job: Find (and eventually bury) dead #ifdef-code!

New and Fixed Configuration Defects over Linux Releases

How good is this, really?

Common Beliefs About Variability in Linux

1. Most variability is expressed by boolean (or tristate) switches.
2. arch-x86 is the largest and allyesconfig selects most features.
3. Variability is mostly implemented with the CPP.
4. The Linux kernel is highly configurable.

Linux v3.1: Feature Distribution by Type

Most variability is expressed by boolean (or tristate) switches

Option-like

<table>
<thead>
<tr>
<th>Feature Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>KCONFIG</td>
<td>11.691 [100%]</td>
</tr>
<tr>
<td>Boolean</td>
<td>6.024 [51.5%]</td>
</tr>
<tr>
<td>Tristate</td>
<td>4.883 [41.8%]</td>
</tr>
<tr>
<td>String</td>
<td>87 [0.7%]</td>
</tr>
<tr>
<td>Integer/Hex</td>
<td>697 [6%]</td>
</tr>
</tbody>
</table>

⇒ Almost all features in Linux are option-like
Linux v3.1: Coverage of arch-x86 / allyesconfig

- arch-x86 is the largest and allyesconfig selects most features

<table>
<thead>
<tr>
<th>Feature</th>
<th>CONFIG features</th>
<th>BUILD interpreted</th>
<th>BUILD only</th>
<th>KBUILD/CPP</th>
<th>CPP only</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch-x86</td>
<td>11,691 [100%]</td>
<td>7,749 [66.3%]</td>
<td>1,925 [16.5%]</td>
<td>3,916 [33.5%]</td>
<td></td>
</tr>
<tr>
<td>non-arch-x86</td>
<td>3,915 [33.5%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allyesconfig</td>
<td>5,482 [46.9%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>non-allyesconfig</td>
<td>2,294 [19.6%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

⇒ arch-x86/allyesconfig is not nearly a full configuration

Linux v3.1: Distribution by Granularity

- Variability is mostly implemented with the CPP

<table>
<thead>
<tr>
<th>Feature</th>
<th>CONFIG features</th>
<th>BUILD interpreted</th>
<th>BUILD only</th>
<th>KBUILD/CPP</th>
<th>CPP only</th>
</tr>
</thead>
<tbody>
<tr>
<td>arch-x86</td>
<td>11,691 [100%]</td>
<td>7,749 [66.3%]</td>
<td>1,925 [16.5%]</td>
<td>3,916 [33.5%]</td>
<td></td>
</tr>
<tr>
<td>non-arch-x86</td>
<td>3,915 [33.5%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>allyesconfig</td>
<td>5,482 [46.9%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>non-allyesconfig</td>
<td>2,294 [19.6%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

⇒ KBUILD implements more than two thirds of all variation points

Linux v3.2: Distribution by HW/SW

- The Linux kernel is highly configurable

<table>
<thead>
<tr>
<th>Feature</th>
<th>CONFIG features</th>
<th>BUILD interpreted</th>
<th>BUILD only</th>
<th>KBUILD/CPP</th>
<th>CPP only</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software related</td>
<td>1,487 [12.4%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware related</td>
<td>10,551 [87.6%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>net</td>
<td>530 [4.4%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>misc</td>
<td>447 [3.7%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivers</td>
<td>5,330 [44.3%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sound</td>
<td>536 [4.5%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arch</td>
<td>4,685 [38.9%]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

⇒ Software features account for only twelve percent of all variation points

Linux Feature Growth over Time (#Features, 2007–2012)

- All features

⇒ HW features (arch/ drivers/ sound/)

⇒ SW features (everything else)
Linux Feature Growth over Time

Results: Where Have all the Features Gone?

1. Most variability is expressed by boolean (or tristate) switches
   - more than 93 percent of all features are option-like
   ~ it is acceptable for tools to ignore value-type features

2. arch-x86 is the largest and allyesconfig selects most features
   - more than 53 percent are not covered by this configuration
   ~ other parts of Linux are probably less tested and error-prone!

3. Variability is mostly implemented with the CPP
   - more than 66 percent of all features are handled
     by the build system, only 17 percent are handled by CPP only
   ~ variability extraction from KBUILD is necessary

4. The Linux kernel is highly configurable
   - only 12 percent of all features configure software only
   - variability is mostly induced by advances in hardware
   ~ complexity will increase further

Challenges: Variability Extraction from the Build System

- Variability extraction \(\mapsto\) which file is selected by which feature?

- Usual approach for variability extraction [6, 9] (KCONFIG, CPP, ...):
  source \(\mapsto\) parse \& transform \(\mapsto\) propositional formula

- Parsing does not work well for MAKE-languages
  - declarative and Turing-complete languages
  - special features, like shell, foreach, eval, addprefix, ...

- Linux’s KBUILD is built on top of (GNU) MAKE
  - nevertheless, researchers have tried parsing to extract variability
    - KBUILDMINER by Berger, She, Czarnecki, et al. [1]
    - Nadi parser by Nadi and Holt [5]
  - resulting tools are too brittle at best
    - work for a (few) Linux version(s) only
    - each usage of a special feature requires manual tailoring

Linux Build Process Revisited
Variability Extraction from KBUILD with GOLEM [2]

Basic idea: Systematic probing and inferring of implications

**Dancing Makefiles**
- Identification of KCONFIG references
- Recursion into subdirectory while considering constraints
- Robust with respect to architecture and version

⇒ no adaptations on or for KBUILD!

### Case Study: Configuration Consistency

Configuration defects in Linux v3.2:

* Without KBUILD constraints
  - Code defects: 1835
  - Referential defects: 415
  - Logical defects: 83
  - Sum: Σ 2333

* With KBUILD constraints
  - Code defects: 1835
  - Referential defects: 439
  - Logical defects: 299
  - Sum: Σ 2573 Result: +10%

Case Study: Configuration Consistency

Agenda

5.1 Motivation
5.2 Variability in Linux
5.3 Configuration Consistency
5.4 Configuration Coverage
5.5 Automatic Tailoring
  - Idea
  - Results
5.6 Summary
5.7 References
Idea: Automated Tailoring of Linux

- Distribution kernels today come with a maximum configuration
- As side-effect, this maximizes the attack surface!
- Each use-case needs its specific, ideal configuration

→ Automatically derive an ideal configuration for a given use case.

Approach

Specific Scenario

- CONFIG_X86=y
- CONFIG_SCSI=n

[Tailored Configuration]

Automatic derivation

- CONFIG_X86=y
- CONFIG_SCSI=n

Evaluation

Ubuntu 12.04 with Linux 3.2 kernel; two use cases
- Web server setup with Apache, MySQL, PHP (LAMP)
- Workstation setup with NFS (Desktop)
- Trace time: 15 min, running defined workload
  - LAMP: Google Skipfish ≤ 5377 unique kernel functions
  - Desktop: iozone, bonnie++ ≤ 6933 unique kernel functions
- Black and whitelist for manual tailoring
  - Blacklist: CONFIG_FTRACE
  - Whitelist: CONFIG_UNIX, CONFIG_PACKET, CONFIG_DEVMTFS,
  CONFIG_DEVMTFS_MOUNT, CONFIG_ATA_PIIX, CONFIG_SATA_AHCI,
  CONFIG_ATA_GENERIC, CONFIG_DRM_I915_KMS, CONFIG_BLK_DEV_INITRD

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Tailored LAMP</th>
<th>Workstation/NFS</th>
<th>Tailored Workstation/NFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel size in Bytes</td>
<td>9,933,860</td>
<td>4,228,235 (44%)</td>
<td>4,792,508 (48%)</td>
<td></td>
</tr>
<tr>
<td>LKM total size in Bytes</td>
<td>62,987,539</td>
<td>2,139,642 (3%)</td>
<td>2,648,034 (4%)</td>
<td></td>
</tr>
<tr>
<td>Options set to ‘y’</td>
<td>1,537</td>
<td>452 (29%)</td>
<td>492 (32%)</td>
<td></td>
</tr>
<tr>
<td>Options set to ‘m’</td>
<td>3,142</td>
<td>43 (1%)</td>
<td>63 (2%)</td>
<td></td>
</tr>
<tr>
<td>Compiled source files</td>
<td>8,670</td>
<td>1,121 (13%)</td>
<td>1,423 (16%)</td>
<td></td>
</tr>
</tbody>
</table>
**Evaluation: Reduction for LAMP**

- **90%** less executable code
- **10%** less functions with known vulnerabilities (with published CVE issues)

**Evaluation: Performance Impact for LAMP**

No observable performance impact

**Results: Automatic Tailoring**

- **HotDep ’12:** Tartler, Kurmus, Ruprecht, Heinloth, Rothberg et al. [7]
- TCB is significantly smaller
- Easy to use: process is fully automated
- If necessary, the tailoring can be guided with whitelists and blacklists
- Going further: Dynamic ASR
  - Even if present: Who is allowed to call what → CFG analysis
  - At runtime: Block illegal invocations.
Summary

- Real-world system software offers thousands of features
  - eCos: 1,250 features
  - Linux: 12,000 features
  - mostly induced by hardware!
- This imposes great challenges for management and maintenance
  - how to ensure configurability consistency?
  - how to ensure configuration coverage?
  - how to keep pace with the constant feature increase?
- A strong call for adequate tool support → VAMOS
  - already found thousands and fixed hundreds of defects and bugs
  - more to come!

Referenzen (Cont’d)


