Understanding Linux Feature Distribution
FOSD Meeting 2012, Braunschweig

Reinhard Tartler

March 22, 2012
VAMOS Research Questions

- What’s the cause of variability and variability implementations
- In Systems Software, variability
  - ... stems from application requirements
  - ... is mostly implementation driven

- What are the challenges for development and maintenance?
  - How to support legacy systems, software and tools?
  - Closing the gap between problem and solution domain

- How can we support developers with tools to manage variability
  - Context: implementation driven, highly configurable software
    → Linux as a perfect subject for investigation
Challenges with Implemented Variability

- **Central Declaration of Variability**: `KCONFIG`
- **Heterogeneous**: Languages, Tools & Concepts

Consistency?

Configuration

Implementation

- `KCONFIG`
- `MAKE`
- `CPP`
- `GCC`
- `LD`
Dominancy and Hierarchy of Variability

1. Feature Modelling: Kconfig in Linux
2. Coarse-grained: MAKE → KBUILD in Linux
3. Fine-grained: CPP → #ifdef-Blocks
4. Language-level: GCC → if(CONFIG_SMP) ...
5. Link-time: LD → branches in linker scripts
6. Run-time: INSMOD, MODPROBE, ...
Configuration and Variability in Linux

1. Configuration with an \texttt{KCONFIG} frontend

2. Compilation of a subset of files

3. Selection of a subset of \texttt{CPP} Blocks

4. Linking of the kernel and loadable kernel modules
Linux v3.1: Feature Distribution by Type

Kconfig features
11,691 [100%]

Option-like
10,907 [93.3%]

Boolean
6,024 [51.5%]

Tristate
4,883 [41.8%]

Value-like
784 [6.7%]

String
87 [0.7%]

Integer/Hex
697 [6%]

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

Kconfig features
11,691 [100 %]

arch-x86
7,776 [66.5 %]

allyesconfig
5,482 [46.9 %]

non-allyesconfig
2,294 [19.6 %]

non-arch-x86
3,915 [33.5 %]

not considered by x86, allyesconfig
6,209 [53.1 %]

⇒ arch-x86/allyesconfig is not nearly a full configuration
Linux v3.1: Coverage of arch-x86 / allyesconfig

Kconfig features
11,691 [100 %]

arch-x86
7,776 [66.5 %]

allyesconfig
5,482 [46.9 %]

non-allyesconfig
2,294 [19.6 %]

non-arch-x86
3,915 [33.5 %]

not considered by x86, allyesconfig
6,209 [53.1 %]

⇒ arch-x86/allyesconfig is not nearly a full configuration
Linux v3.1: Distribution by Granularity

⇒ \texttt{Kbuild} implements more than two thirds of all variation points
⇒ **Kbuild** implements more than two thirds of all variation points
Variability Extraction from \textsc{Kbuild} with \textsc{golem}

Basic idea: \textit{Systematic Testing} and \textit{Inferring of implications}

submitted to (SPLC '2012)
Variability Extraction from KBUILD with GOLEM

Basic idea: Systematic Testing and Inferring of implications
submitted to (SPLC ’2012)

- Dancing Makefiles
- Identification of KCONFIG references

```
obj-y += fork.o
obj-$(CONFIG_SMP) += spinlock.o
obj-$(CONFIG_APM) += apm.o
```
Variability Extraction from `KBUILD` with `GOLEM`

Basic idea: *Systematic Testing* and *Inferring of implications*

submitted to *(SPLC ’2012)*

- **Dancing Makefiles**
- Identification of `KCONFIG` references
- Recursion into subdirectory while considering constraints

```makefile
obj-y += fork.o
obj-$(CONFIG_SMP) += spinlock.o
obj-$(CONFIG_APM) += apm.o
obj-$(CONFIG_PM) += power/
```

Robust with respect to architecture and version

⇒ no adaptations on or for `Kbuild`!

Kernelversion found inferences

- v2.6.25 6, 2256 (92.6%)
- v2.6.28.6 6, 2256 (92.6%)
- v2.6.33.3 8, 2256 (92.9%)
- v2.6.37 9, 2256 (93.4%)
- v3.2 10, 2256 (94.1%)

Readers@cs.fau.de Understanding Linux Feature Distribution (March 22, 2012) 9–13
Variability Extraction from KBUILD with GOLEM

Basic idea: Systematic Testing and Inferring of implications
submitted to (SPLC '2012)

- 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!

obj-y += fork.o
obj-$ (CONFIG_SMP) += spinlock.o
obj-$ (CONFIG_APM) += apm.o
obj-$ (CONFIG_PM) += power/

<table>
<thead>
<tr>
<th>Kernelversion</th>
<th>found inferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>v2.6.25</td>
<td>6,200 (92.6%)</td>
</tr>
<tr>
<td>v2.6.28.6</td>
<td>6,959 (92.6%)</td>
</tr>
<tr>
<td>v2.6.33.3</td>
<td>8,886 (92.9%)</td>
</tr>
<tr>
<td>v2.6.37</td>
<td>9,959 (93.4%)</td>
</tr>
<tr>
<td>v3.2</td>
<td>10,908 (94.1%)</td>
</tr>
</tbody>
</table>

tartler@cs.fau.de Understanding Linux Feature Distribution (March 22, 2012)
Case Study: Variability Consistency Analysis

Eurosys 2011: *The Linux 10000 Feature Nightmare*

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

\[ I = (\text{Block}_1 \leftrightarrow \text{DISCONTIGMEM}) \]
\[ \wedge (\text{Block}_2 \leftrightarrow \text{Block}_1 \wedge (\text{NUMA})) \]
\[ \wedge (\text{Block}_3 \leftrightarrow \text{Block}_1 \wedge \neg \text{Block}_2) \]

**dead?** sat\((C \land I \land \text{Block}_N)\)

**undead?** sat\((C \land I \land \neg \text{Block}_N \land \text{parent}(\text{Block}_N))\)

**Result:** > 100 patches
Case Study: Configuration Consistency Analysis

Configuration defects in Linux v3.2:

*Without `KBUILD` constraints*

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code defects</td>
<td>1873</td>
</tr>
<tr>
<td>Referential defects</td>
<td>486</td>
</tr>
<tr>
<td>Logical defects</td>
<td>97</td>
</tr>
<tr>
<td><strong>Sum:</strong></td>
<td><strong>Σ 2456</strong></td>
</tr>
</tbody>
</table>

*With `KBUILD` constraints*

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code defects</td>
<td>1873</td>
</tr>
<tr>
<td>Referential defects</td>
<td>506</td>
</tr>
<tr>
<td>Logical defects</td>
<td>512</td>
</tr>
<tr>
<td><strong>Sum:</strong></td>
<td><strong>Σ 2891</strong></td>
</tr>
</tbody>
</table>

Result: +15%

Technical Challenges

- `#define/#undef` Handling (STTT ’12)
- `KBUILD`-Inferences (SPLC ’12)
Case Study: Configuration Consistency Analysis

Configuration defects in Linux v3.2:

**Without **\texttt{Kbuild} **constraints**

- Code defects: 1873
- Referential defects: 486
- Logical defects: 97
- Sum: $\sum 2456$

**With **\texttt{Kbuild} **constraints**

- Code defects: 1873
- Referential defects: 506
- Logical defects: 512
- Sum: $\sum 2891$  
  Result: $+15\%$

Technical Challenges

- \texttt{#define/#undef} Handling \hspace{2cm} (\textit{STTT '12})
- \texttt{Kbuild}-Inferences \hspace{2cm} (\textit{SPLC '12})
Case Study: Configuration Coverage

How to ensure that each single line is at least compiled?

Configuration Coverage

Making static analysis *variability aware*

- Option 1: Variability-aware parsing
  
  *(TypeChef, @OOPSLA’11)*

- Option 2: **Configuration Coverage**
  
  *(published @ACMOSR 45.3)*

Basic idea:

1. (ideally) small set of configurations
2. (legacy) static analysis tools on each configuration

Finding hard to detect bugs
Case Study: Configuration Coverage

How to ensure that each single line is at least compiled?
⇒ Configuration Coverage
⇒ Making static analysis *variability aware*

- Option 1: Variability-aware parsing
- Option 2: **Configuration Coverage** *(published @ACMOSR 45.3)*

Basic idea:
1. (ideally) small set of configurations
2. (legacy) static analysis tools on each configuration

⇒ Finding hard to detect bugs

Case study in SPLC’12: Improvement: 46.6% → 95.5% *(allyesconfig: 90.1%)*
Case Study: Configuration Coverage

- How to ensure that each single line is at least compiled?
- Configuration Coverage
- Making static analysis variability aware
  - Option 1: Variability-aware parsing
  - TypeChef, @OOPSLA’11
  - Option 2: Configuration Coverage
  - published @ACMOSR 45.3
- Basic idea:
  1. (ideally) small set of configurations
  2. (legacy) static analysis tools on each configuration
- Finding hard to detect bugs
- Case study in SPLC’12: Improvement: 46.6% → 95.5% (allyesconfig: 90.1%)
Conclusion

- Variability has to be seen as source of bugs on its own respect
- Option-like configuration types clearly dominates value-like types
- Hierarchy of variability that shows how variability points, which are managed by different tools, concepts and languages related to and dominate each other
- Most of the Linux variability is managed by KBUILD (as opposed to KCONFIG)
- Extracting Variability from KBUILD improves variability analyses significantly.