Konfigurierbare Systemsoftware (KSS)

VL 7 – Summary and Discussion

Daniel Lohmann

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität
Erlangen-Nürnberg

SS 16 – 2016-07-11

http://www4.informatik.uni-erlangen.de/Lehre/SS16/V_KSS
7.1 Summary
7.2 From Instance- to Interaction Tailoring
7.3 Evaluation und Diskussion
7.4 References
The Operating System – A Swiss Army Knife?
The Operating System – A Swiss Army Knife?

Commodity operating systems provide a rich set of features to be prepared for all kinds of applications and contingencies:

- Malicious or erroneous applications
  - preemptive scheduling, address space separation, disk quotas
- Multi-user operation
  - authentication, access validation and auditing
- Multi-threaded and interacting applications
  - Threads, semaphores, pipes, sockets
- Many/large concurrently running applications
  - virtual memory, swapping, working sets
Clearly, the operating system design must be strongly influenced by the type of use for which the machine is intended. Unfortunately it is often the case with ‘general purpose machines’ that the type of use cannot be easily identified; a common criticism of many systems is that in attempting to be all things to all men they wind up being totally satisfactory to no-one.

Clearly, the operating system design must be strongly influenced by the type of use for which the machine is intended. Unfortunately it is often the case with ‘general purpose machines’ that the type of use cannot be easily identified; a common criticism of many systems is that in attempting to be all things to all men they wind up being totally satisfactory to no-one.

Big is beautiful?

"Some applications may require only a subset of services or features that other applications need. These 'less demanding' applications should **not be forced to pay** for the resources consumed by unneeded features."

“Some applications may require only a subset of services or features that other applications need. These ‘less demanding’ applications should **not be forced to pay** for the resources consumed by unneeded features.”

Between a Rock and a Hard Place...

functional and nonfunctional requirements

Application
System Software

Hardware

functional and nonfunctional properties

tasks
sockets
file system
...
event latency
safety
...
ISA
IRQ handling
MMU / MPU
...
cache size
coherence
IRQ latency
...
Between a Rock and a Hard Place...

functional and nonfunctional requirements

- High variety of functional and nonfunctional application requirements
- High variety of hardware platforms
- High per-unit cost pressure

\( \Rightarrow \) System software has to be tailored for each concrete application

functional and nonfunctional properties
Configurable Software – Software Product Line

Problem Space

Domains Expert

Features and Dependencies

f1

f2

f3

f4

f5

f6

f7

Solution Space

Architect / Developer

Architecture and Implementation

Aspect

Class

Configuration

f1

f2

... Specific Problem

Specific Solution

System User

intentional side

extensional side

System User

intended properties

actual implementation

Variant

A

B

C

D

© dl KSS (VL 7 | SS 16)
7 Summary and Discussion | 7.1 Summary
Configurable Software – Software Product Line

- **Problem Space**
  - Domain Expert
  - Features and Dependencies: \( f_1, f_2, f_3, f_4, f_5, f_6, f_7 \)

- **Solution Space**
  - Architect / Developer
  - Classes, Aspects

- **KSS**
  - Architecture and Implementation

- **Configuration**
  - System User
  - Specific Problem: \( f_1, f_2, \ldots \)
  - Intended properties

- **Variant**
  - System User

- **Model Level**

- **Instance Level**

- **Intentional Side**

- **Extensional Side**

© dl KSS (VL 7 | SS 16) 7 Summary and Discussion | 7.1 Summary
Focus: solution space techniques
Implementation Techniques: Classification

### Decompositional Approaches
- Text-based filtering (untyped)
- Preprocessors

![Decompositional Approaches Diagram]

### Compositional Approaches
- Language-based composition mechanisms (typed)
- OOP, AOP, Templates

![Compositional Approaches Diagram]

### Generative Approaches
- Metamodel-based generation of components (typed)
- MDD, C++ TMP, generators

![Generative Approaches Diagram]
Feature vs. Instance-Based Configuration

- **OSEK**
- **eCos**
- **FreeRTOS**
- **Linux**
- **QNX**

**Static**
- Feature instantiation

**Dynamic**
- System object instantiation

© dl  KSS (VL 7 | SS 16)  7 Summary and Discussion | 7.1 Summary
Not only **features**, but also **object instances** are known at compile-time:

- Facilitates optimizations (static arrays instead of linked lists, . . . )
Real-world flight-control application (11 tasks, 3 alarms, 1 ISR)

Results with eCos and ERIKA Enterprise (open source OSEK)

<table>
<thead>
<tr>
<th></th>
<th>eCos</th>
<th>ERIKA</th>
<th>factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel code (bytes)</td>
<td>14763</td>
<td>6765</td>
<td>2.2x</td>
</tr>
<tr>
<td>kernel time (instructions)</td>
<td>88465</td>
<td>46087</td>
<td>1.9x</td>
</tr>
<tr>
<td>robustness ($10^9$ SDCs)</td>
<td>148</td>
<td>18</td>
<td>8.2x</td>
</tr>
</tbody>
</table>
Traditional Operating-System Design

Application

ISR1  ISR2  Task1  Task2  Task3

Feature  Feature  Feature  Feature  Feature

Hardware

© dl  KSS (VL 7 | SS 16)  7 Summary and Discussion | 7.1 Summary
Traditional Operating-System Design

Application

HAL

least common denominator

abstract

Feature

Feature

Feature

Feature

Hardware

ISR1

ISR2

Task1

Task2

Task3
Hardware-Centric Operating-System Design

Application

Kernel with Active HAL

ISR1  ISR2  Task1  Task2  Task3

Feature  Feature  Feature  Feature  Feature

Hardware

Infineon  ARM  Intel  AMD  Power  ARM  Cortex

© dl  KSS (VL 7 | SS 16)  7 Summary and Discussion | 7.1 Summary
Agenda

7.1 Summary
7.2 From Instance- to Interaction Tailoring
7.3 Evaluation und Diskussion
7.4 References
An extremely fault-tolerant OSEK implementation

Dependability by constructive measures

- Employ standard hardware memory protection
- Aggressive avoidance of indirections $\leadsto$ lots of inlining
- Arithmetic encoding of the kernel path (scheduler)
dOSEK: Dependability-Oriented Static Embedded Kernel

An extremely fault-tolerant OSEK implementation

- Dependability by constructive measures
  - Employ standard hardware memory protection
  - Aggressive avoidance of indirections \( \leadsto \) lots of inlining
  - Arithmetic encoding of the kernel path (scheduler)

Scenario: quadrotor flight-control application

- 11 tasks, 3 alarms, 1 ISR
- 53 syscall invocations

Results (compared to ERIKA enterprise)

- SDC reduction by \textbf{5 orders of magnitude}: \( 10^9 \rightarrow 10^4 \) SDCs
- Code size increases by \textbf{factor 25}: \( 8 \rightarrow 200 \) KiB
- Syscall latency increases by \textbf{factor 4}: \( 100 \rightarrow 400 \) cycles
dOSEK: Dependability-Oriented Static Embedded Kernel

An extremely fault-tolerant OSEK implementation

- Dependability by constructive measures
  - Employ standard hardware memory protection
  - Agressive avoidance of indirections \( \leadsto \) lots of inlining
  - Arithmetic encoding of the kernel path (scheduler)

- Scenario: quadrotor flight-control application
  - 11 tasks, 3 alarms, 1 ISR
  - 53 syscall invocations

- Results (compared to ERIKA enterprise)
  - SDC reduction by **5 orders of magnitude**: \( 10^9 \rightarrow 10^4 \) SDCs
  - Code size increases by **factor 25**: \( 8 \rightarrow 200 \) KiB
  - Syscall latency increases by **factor 4**: \( 100 \rightarrow 400 \) cycles

*Culprit:* arithmetically encoded scheduler \( \leadsto \) avoid scheduling!
Ein vereinfachtes OSEK System

**Task 1; Priority 4**

```c
TASK(Task1) {
    int data = read_data();
    if (data == '\0') {
        ActivateTask(Task3);
    } else {
        bb_put(data);
    }
    ChainTask(Task2);
}
```

**Task 2; Priority 5**

```c
TASK(Task2) {
    setup_of_device();
    TerminateTask();
}
```

**Task 3; Priority 3**

```c
TASK(Task3) {
    parse_message();
    bb_clear_buffer();
    TerminateTask();
}
```
An OSEK System: Control-Flow Graphs

Task 1; Priority 4

```plaintext
data = read_data();
if (data == '\0')
bb_put(data);
ActivateTask(Task3);
ChainTask(Task2);
```

Task 2; Priority 5

```plaintext
setup_of_device()
TerminateTask()
```

Task 3; Priority 3

```plaintext
parse_message()
bb_clear_buffer();
TerminateTask();
```
An OSEK System: Control-Flow Graphs

Task 1; Priority 4

- Data = read_data();
- if (data == \0)
  - bb_put(data);
  - ActivateTask(Task3);
  - ChainTask(Task2);

Task 2; Priority 5

- TerminateTask();

Task 3; Priority 3

- TerminateTask();
Spezialisierung von Systemaufrufen

Task 1; Priority 4

- ActivateTask(Task3);
- ChainTask(Task2);

Task 2; Priority 5

- TerminateTask();

Task 3; Priority 3

- TerminateTask();
(Partial) Specialization of System Calls

**Task 1; Priority 4**

- SetReady(Task3);
- ChainTask(Task2);

**Task 2; Priority 5**

- TerminateTask();

**Task 3; Priority 3**

- TerminateTask();

Data = read_data();
if (data == '\0')
bb_put(data);
SetReady(Task3);
ChainTask(Task2);

TerminateTask();
Idle
(Partial) Specialization of System Calls

Task 1; Priority 4

SetSuspended(Task1)
SetReady(Task3);

Task 2; Priority 5

TerminateTask();

Idle

Task 3; Priority 3

Idle

TerminateTask();
(Partial) Specialization of System Calls

Task 1; Priority 4

- SetReady(Task3);
- SetSuspended(Task1)
- SetReady(Task2)
- DispatchTo(Task2);

Task 2; Priority 5

- SetSuspended(Task2)
- Reschedule()

Task 3; Priority 3

- Idle
- TerminateTask();

Idle
(Partial) Specialization of System Calls

Task 1; Priority 4

SetReady(Task3);
SetSuspended(Task1)
SetReady(Task2)
DispatchTo(Task2);

Task 2; Priority 5

SetSuspended(Task2)
Reschedule()

Task 3; Priority 3

SetSuspended(Task3);
GotoIdle();
dOSEK: Dependability-Oriented Static Embedded Kernel

An extremely fault-tolerant OSEK implementation

- Dependability by constructive measures
  - Employ standard hardware memory protection
  - Aggressive avoidance of indirections $\leadsto$ lots of inlining
  - Arithmetic encoding of the kernel path (scheduler)

- Scenario: quadrotor flight-control application
  - 11 tasks, 3 alarms, 1 ISR
  - 53 syscall invocations

- Results (compared to ERIKA enterprise)
  - SDC reduction by **5 orders of magnitude**: $10^9 \rightarrow 10^4$ SDCs
  - Code size increases by **factor 25**: $8 \rightarrow 200$ KiB
  - Syscall latency increases by **factor 4**: $100 \rightarrow 400$ cycles
dOSEK: Dependability-Oriented Static Embedded Kernel

An extremely fault-tolerant OSEK implementation

- Dependability by constructive measures
  - Employ standard hardware memory protection
  - Aggressive avoidance of indirections \(\leadsto\) lots of inlining
  - Arithmetic encoding of the kernel path (scheduler)

Scenario: quadrotor flight-control application
- 11 tasks, 3 alarms, 1 ISR
- 53 syscall invocations
  \[\{\text{243 GCFG edges}\}\]

Results with **call-site specialization** LCTES ’15 [1]
- SDC reduction by **5 orders of magnitude:** \(10^9 \rightarrow 10^4\) SDCs
- Code size increases by **factor 10.5:** \(8 \rightarrow 85\) KiB
- Syscall latency increases by **factor 1.5:** \(100 \rightarrow 150\) cycles

\(\leadsto\) **Further application-specific tailoring pays off!**
Instance-Based Tailoring
(e.g., based on OIL file)

Kernel (tailored to configuration)

- Kernel constrained to specified features and system objects.
Interaction-Based Tailoring (e.g., based on GCFG analysis)

Kernel (tailored to actual usage)

- Kernel constrained to specified features and system objects.
- Further constrained to actually possible app → kernel interactions.
Agenda

7.1 Summary
7.2 From Instance- to Interaction Tailoring
7.3 Evaluation und Diskussion
7.4 References
Evaluation

...
Diskussion

Am coolsten finde / fand ich...

Ich habe vermisst...

Bei einer Erweiterung auf 5 ECTS...
Wie gehts weiter?

(Bachelor/Master)

Systemprogrammierung
10 ECTS

SST
7,5

Systemsoftwaretechnik

EZS
5 – 7,5

BS
5 – 7,5

MW
5 – 7,5

EZS2

P: 10

PASST

P: 10

BST
5

SST
7,5

KSS
2,5

VS
5 – 7,5

Examensarbeit / Projektarbeit

BA / SA, MA / DA, PA

Seminar

MA

Summary and Discussion | 7.3 Evaluation und Diskussion

7–28

