Levels of Specialization in Real-Time Operating Systems

Björn Fiedler, Gerion Entrup, Christian Dietrich, Daniel Lohmann

Leibniz Universität Hannover

July 3, 2018

supported by DFG
Embedded Systems are Special-Purpose Systems
Embedded Systems are Special-Purpose Systems

Applications are special-purpose systems.

RTOS can be a special-purpose system, too!
Agenda

- Motivation
- Specialization
  - Levels of System-Software Specialization
  - Case Study: GPSLogger
  - Benefits and Challenges
- Summary
“Between a Rock and a Hard Place”

**functional and nonfunctional requirements**

**Application**

**Hardware**

**functional and nonfunctional properties**
“Between a Rock and a Hard Place”

Functional and nonfunctional requirements

Application

Knowledge

Functional and nonfunctional properties

Hardware
“Between a Rock and a Hard Place”

**Method:** Static/Dynamic Analysis

**Knowledge**

functional and nonfunctional requirements

Application

hardware

functional and nonfunctional properties

**Method:** Static/Dynamic Analysis
“Between a Rock and a Hard Place”

**Technique:** Compilation and Generation

**Knowledge**

**Application/Hardware-Tailored System Software**

- **Functional and non-functional requirements**
- **Application** (e.g., MySQL)

**Hardware**

- **Functional and non-functional properties**

**LUH Levels of Specialization in Real-Time Operating Systems – Motivation**
functional and nonfunctional requirements

Application

Application/ Hardware-Tailored System Software

Hardware

functional and nonfunctional properties

Knowledge

Technique: Compilation and Generation
“Between a Rock and a Hard Place”

Improved nonfunctional properties?

functional and nonfunctional requirements

Application

Application/Hardware-Tailored System Software

Hardware

functional and nonfunctional properties

Knowledge

Technique: Compilation and Generation

LUH Levels of Specialization in Real-Time Operating Systems – Motivation
System-Software Specialization

- **Specialization:** Subsetting to “what is actually needed” by exploiting a-priori knowledge.

  ➔ There is always more knowledge than you think!
System-Software Specialization

- **Specialization:** Subsetting to “what is actually needed” by exploiting a-priori knowledge.

  🔄 There is always more knowledge than you think!

- **Levels:** Taxonomy of three subsequent levels of specialization.

  1. **Abstraction**  
     *What* functionality is used.

  2. **Instance**  
     *Which* entities use that functionality.

  3. **Interaction**  
     *How* do they use that functionality.
System-Software Specialization

- **Specialization:** Subsetting to “what is actually needed” by exploiting a-priori knowledge.

  → There is always more knowledge than you think!

- **Levels:** Taxonomy of three subsequent levels of specialization.

  1. **Abstraction**
     - *What* functionality is used.
  2. **Instance**
     - *Which* entities use that functionality.
  3. **Interaction**
     - *How* do they use that functionality.

- **Knowledge:** The more we know, the deeper we can specialize.
What is a (correct) Specialization

\[ RTS(\vec{I}) = \vec{O} \]

RT-system specification
defines system input \(\vec{I}\),
defines correct output \(\vec{O}\)
What is a (correct) Specialization

\[ RTS(\vec{I}) = \vec{O} \]

RT-system **specification** defines system input \( \vec{I} \), defines correct output \( \vec{O} \)

\[ RTS_{\text{RTOS HW}}(\vec{I}) = \vec{O} \]

Concrete **implementation** consumes input \( \vec{I} \), produces output \( \vec{O} \)
What is a (correct) Specialization

\[ RTS(\vec{I}) = \vec{O} = RTS^{\text{APP}}_{\text{RTOS}}(\vec{I}) \]

RT-system specification defines system input \( \vec{I} \), defines correct output \( \vec{O} \)

Concrete implementation consumes input \( \vec{I} \), produces output \( \vec{O} \)
What is a (correct) Specialization

**Specification** determines correctness

\[
RTS(\vec{I}) = \vec{O} \quad \overset{RTS}{=} \quad RTS_{\text{RTOS}}^{\text{APP}}(\vec{I})
\]

RT-system **specification**
defines system input \( \vec{I} \),
defines correct output \( \vec{O} \)

Concrete **implementation**
consumes input \( \vec{I} \),
produces output \( \vec{O} \)
What is a (correct) Specialization

$$RTS(\vec{I}) = \vec{O}$$

**Specification** determines correctness

RT-system **specification** defines system input $$\vec{I}$$, defines correct output $$\vec{O}$$

**Specialization**: modify implementation, remove flexibility

$$RTS = RTS_{\text{RTOS}}(\vec{I})$$

Concrete **implementation** consumes input $$\vec{I}$$, produces output $$\vec{O}$$
Motivation

Specialization

Levels of System-Software Specialization

Case Study: GPSLogger

Benefits and Challenges

Summary
OS Specialization: 0 – Standard/API

not specialized

supports any application
OS Specialization: 1 – Abstraction-Level Specialization

Abstractions

Control-Row Abstractions
- ✓ Thread
- ✓ ISR

Synchronization Abstractions
- ✓ Event
- · · ·

Subsetting of available abstractions/features

activate
interrupt
resume
iret
interrupt
activate
interrupt
preempt
activate
interrupt
interrupt
preempt
### OS Specialization: 1 – Abstraction-Level Specialization

<table>
<thead>
<tr>
<th>Control-Flow Abstractions</th>
<th>Synchronization Abstractions</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Thread</td>
<td>✓ Event</td>
</tr>
<tr>
<td>✓ ISR</td>
<td>✓ Nested ISR</td>
</tr>
<tr>
<td>✓ Nested ISR</td>
<td>...</td>
</tr>
</tbody>
</table>

**Abstractions**

- **Thread**
- **ISR**
- **Nested ISR**
- **Event**

Subsetting of available abstractions/features:

- **Thread**: `T`
- **ISR**: `I`
- **Event**: `E`

**Actions**

- **activate**
- **interrupt**
- **block**
- **set**
- **wait**
- **preempt**
- **activate**
- **interrupt**
- **set**
- **interrupt**
- **preempt**

**Example Interaction**

- Thread: `T`
  - `activate`
  - `interrupt`
  - `block`
- ISR: `I`
  - `activate`
  - `interrupt`
- Event: `E`
  - `set`
  - `interrupt`
OS Specialization: 1 – Abstraction-Level Specialization

Abstractions

Control-Flow Abstractions
- Thread ✓
- ISR ✓
- Nested ISR

Synchronization Abstractions
- Event
- ...

Subsetting of available abstractions/features

Abstractions

Thread
- T
- T...

ISR
- I
- I...

Event
- E
- E...

activate
interrupt
block
preempt
wait
set
interrupt
set
activate

Class of applications
OS Specialization: 2 – Instance-Level Specialization

Control-Row Abstractions

- Thread
- ISR
- Nested ISR

Synchronization Abstractions

- Event
- ...

Abstractions

Instances

subsetting of available instances

activate
interrupt
block
preempt
set
activate
interrupt
set
interrupt
OS Specialization: 2 – Instance-Level Specialization

Abstractions

Instances

subsetting of available instances

specific application
OS Specialization: 3 – Interaction-Level Specialization

Control-Flow Abstractions
- Thread
- ISR
- Nested ISR

Synchronization Abstractions
- Event
- ...

Abstractions
- instances

Instances
- T1: Thread
- T2: Thread
- I1: ISR

Interactions
- Subsetting of possible instance interactions

Specific application implementation

Diagram showing
- Activate
- Interrupt
- Ret
- Resume
- Preempt
- Set
- Wait
Agenda

- Motivation
- Specialization
- Levels of System-Software Specialization
- Case Study: GPSLogger
- Benefits and Challenges
- Summary

https://github.com/grafalex82/GPSLogger
Case Study: GPSLogger
Case Study: GPSLogger

- **Abstractions**: Already done by CPP macros

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstractions</td>
<td></td>
</tr>
<tr>
<td>Text segment (bytes)</td>
<td>91,084</td>
</tr>
<tr>
<td>Data &amp; BSS segment (bytes)</td>
<td>18,328</td>
</tr>
<tr>
<td>Startup time (cycles)</td>
<td>60,426</td>
</tr>
</tbody>
</table>
Case Study: GPSLogger

- **Abstractions:** Already done by CPP macros
- **Instances:** Threads and queues statically instantiated

### Abstractions

<table>
<thead>
<tr>
<th>Segment</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>91,084</td>
</tr>
<tr>
<td>Data &amp; BSS</td>
<td>18,328</td>
</tr>
<tr>
<td>Startup time</td>
<td>60,426</td>
</tr>
</tbody>
</table>
Case Study: GPSLogger

- **Abstractions:** Already done by CPP macros
- **Instances:** Threads and queues statically instantiated

<table>
<thead>
<tr>
<th></th>
<th>Abstractions</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text segment (bytes)</td>
<td>91,084</td>
<td>91,196</td>
</tr>
<tr>
<td>Data &amp; BSS segment (bytes)</td>
<td>18,328</td>
<td>17,472</td>
</tr>
<tr>
<td>Startup time (cycles)</td>
<td>60,426</td>
<td>53,827</td>
</tr>
</tbody>
</table>
Case Study: GPSLogger

- **Abstractions:** Already done by CPP macros
- **Instances:** Threads and queues statically instantiated
- **Interactions:** LED task inlined in ISR

<table>
<thead>
<tr>
<th></th>
<th>Abstractions</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text segment (bytes)</td>
<td>91,084</td>
<td>91,196</td>
</tr>
<tr>
<td>Data &amp; BSS segment (bytes)</td>
<td>18,328</td>
<td>17,472</td>
</tr>
<tr>
<td>Startup time (cycles)</td>
<td>60,426</td>
<td>53,827</td>
</tr>
</tbody>
</table>
Case Study: GPSLogger

- **Abstractions:** Already done by CPP macros
- **Instances:** Threads and queues statically instantiated
- **Interactions:** LED task inlined in ISR

<table>
<thead>
<tr>
<th></th>
<th>Abstractions</th>
<th>Instances</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text segment (bytes)</td>
<td>91,084</td>
<td>91,196</td>
<td>90,572</td>
</tr>
<tr>
<td>Data &amp; BSS segment (bytes)</td>
<td>18,328</td>
<td>17,472</td>
<td>16,712</td>
</tr>
<tr>
<td>Startup time (cycles)</td>
<td>60,426</td>
<td>53,827</td>
<td>51,029</td>
</tr>
</tbody>
</table>
Agenda

- Motivation
- Specialization
- Levels of System-Software Specialization
- Case Study: GPSLogger
- Benefits and Challenges
- Summary
Benefits

- Memory footprint reduction
  - CiAO [ISORC ’14]

90% less code
50% less RAM usage
Benefits

- Memory footprint reduction
  - CiA0 [ISORC '14]
- Security and safety improvements
  - CADOS [HotDep '12]

10% less code with CVE entries
5× more robust against bitflips
Benefits

- Memory footprint reduction
  - CiA0 [ISORC '14]
- Security and safety improvements
  - CADOS [HotDep '12]
- Better exploitation of hardware
  - SLOTH [RTSS '09; RTSS '12]

171× less dispatch latency
Benefits

- Memory footprint reduction
  - CiA0 [ISORC '14]
- Security and safety improvements
  - CADOS [HotDep '12]
- Better exploitation of hardware
  - SLOTH [RTSS '09; RTSS '12]
- Reduction of jitter and kernel latency
  - OSEK-V [LCTES '17]

OSEK-V

81% less cycles for dispatch
Benefits

- Memory footprint reduction
  - CiA0 [ISORC '14]
- Security and safety improvements
  - CADOS [HotDep '12]
- Better exploitation of hardware
  - SLOTH [RTSS '09; RTSS '12]
- Reduction of jitter and kernel latency
  - OSEK-V [LCTES '17]
- Analyzability and testability
  - SysWCET [RTAS '17]
  - SysWCEC [ECRTS '18]
  - 15% better WCRT bounds
Challenges

- You have to know what you need

17000 features in Linux
Challenges

- You have to know what you need
- You have to be able to express what you need

POSIX style
Challenges

- You have to know what you need
- You have to be able to express what you need
- Testability and certifiability

Costs
Summary

Levels of specialization

Abstractions
- Thread ✓
- ISR ✓
- Nested ISR
- Event
- ... 

Intstances
- T1 : Thread
- T2 : Thread
- I1 : ISR

Interactions
T2
activate
interrupt
iret
resume
I1
interrupt
T1
- Specialization significantly improves nonfunctional properties
- Manual specialization: too costly and time-consuming
Summary

Levels of specialization

- Abstractions
  - Thread
  - ISR
  - Nested ISR
- Intstances
  - T1 : Thread
  - T2 : Thread
  - I1 : ISR
- Interactions
  - T2
  - I1
  - T1

Specialization significantly improves nonfunctional properties

- Manual specialization: too costly and time-consuming
  - Automation
  - Integration in toolchain
Summary

Levels of specialization

<table>
<thead>
<tr>
<th>Abstractions</th>
<th>Instances</th>
<th>Interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>T1: Thread</td>
<td>T2: Thread</td>
</tr>
<tr>
<td>ISR</td>
<td>I1: ISR</td>
<td></td>
</tr>
<tr>
<td>Nested ISR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Event</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Specialization significantly improves nonfunctional properties

- Manual specialization: too costly and time-consuming

  → Automation

→ Integration in toolchain

There always is more to specialize than you’d expect


