Levels of Specialization in Real-Time Operating Systems

Björn Fiedler, Gerion Entrup, Christian Dietrich, Daniel Lohmann

Leibniz Universität Hannover

July 3, 2018

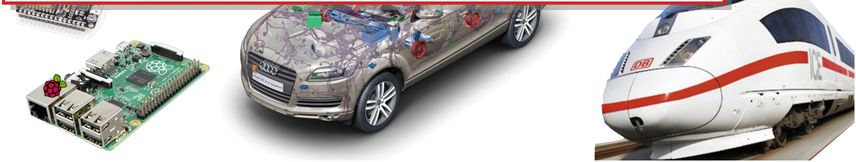
supported by DFG

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**

functional and nonfunctional **properties**

"Between a Rock and a Hard Place"

functional and nonfunctional **requirements**

Knowledge

functional and nonfunctional **properties**

"Between a Rock and a Hard Place"

functional and nonfunctional **requirements**

Knowledge

functional and nonfunctional **properties**

Method: Static/Dynamic Analysis

"Between a Rock and a Hard Place"

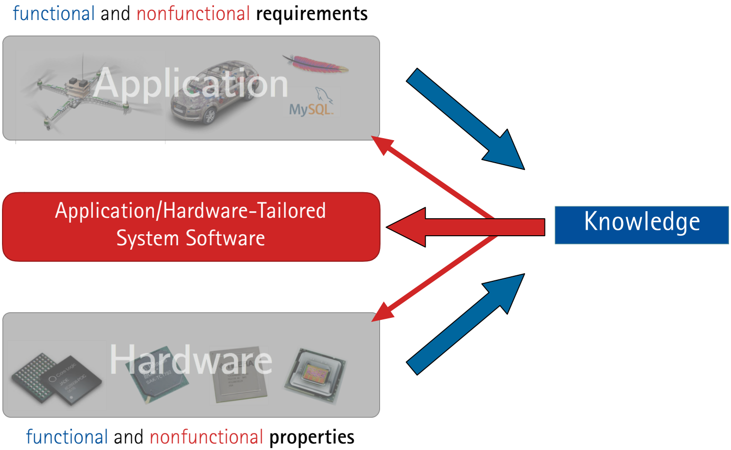
functional and nonfunctional **requirements**

Application/Hardware-Tailored System Software ← Knowledge

functional and nonfunctional **properties**

Technique: Compilation and Generation

"Between a Rock and a Hard Place"



Technique: Compilation and Generation

"Between a Rock and a Hard Place"

functional and nonfunctional **requirements**

Improved nonfunctional properties?

Application/Hardware-Tailored System Software

Knowledge

functional and nonfunctional **properties**

Technique: Compilation and Generation

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 $\vec{O}$

$$RTS_{\substack{APP\\RTOS\\HW}}(\vec{I}) = \vec{O}$$

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

What is a (correct) Specialization

$$RTS(\vec{I}) = \vec{O} = RTS_{\substack{\text{APP}\\ \text{RTOS}\\ \text{HW}}}(\vec{I})$$

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

$$RTS(\vec{I}) = \vec{O} \overset{RTS}{=\!=} RTS^{\substack{APP \\ RTOS \\ HW}}(\vec{I})$$

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

Specialization: modify implementation remove flexibility

$$RTS(\vec{I}) = \vec{O} \overset{RTS}{=} RTS^{\text{APP}}_{\text{RTOS, HW}}(\vec{I})$$

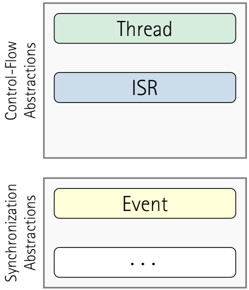
RT-system **specification** defines system input $\vec{I}$, defines correct output $\vec{O}$

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

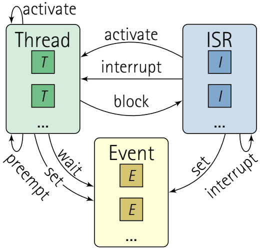
Agenda

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

OS Specialization: **0 – Standard/API**

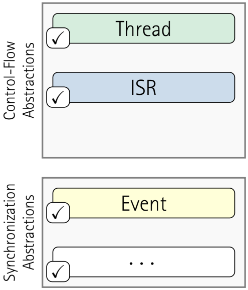


not specialized



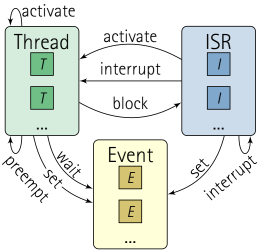
supports any application

OS Specialization: **1 – Abstraction-Level Specialization**

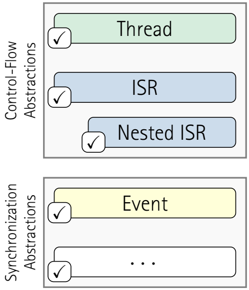


Abstractions

subsetting of available abstractions/features

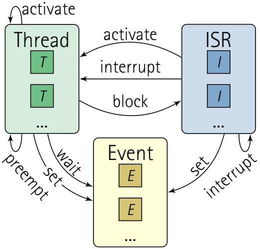


OS Specialization: **1 – Abstraction-Level Specialization**

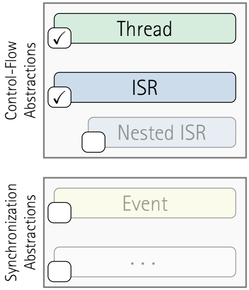


Abstractions

subsetting of available abstractions/features

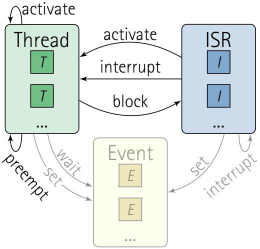


OS Specialization: **1 – Abstraction-Level Specialization**



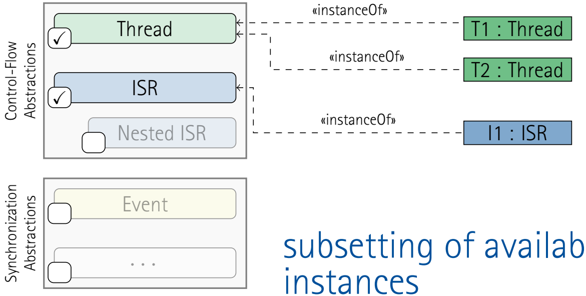
Abstractions

subsetting of available abstractions/features



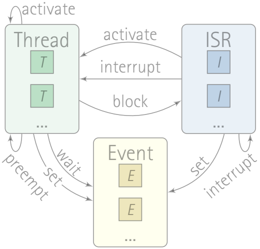
class of applications

OS Specialization: **2 – Instance-Level Specialization**

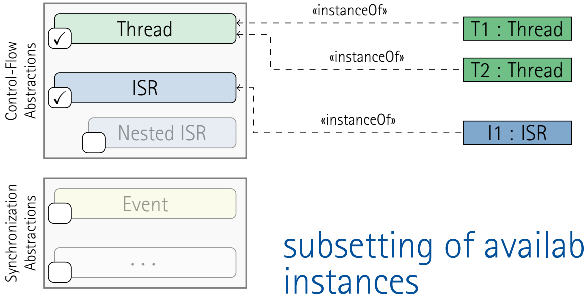


subsetting of available instances

Abstractions Instances



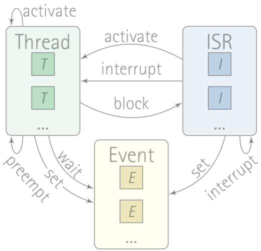
OS Specialization: **2 – Instance-Level Specialization**

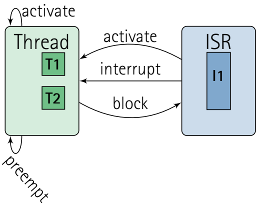


subsetting of available instances

Abstractions

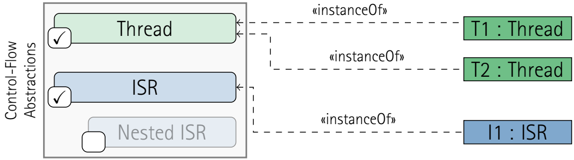
Instances

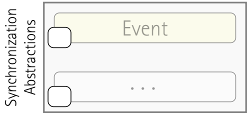


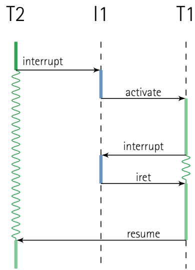


specific application

OS Specialization: **3 – Interaction-Level Specialization**

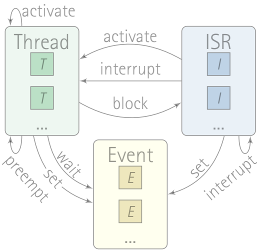


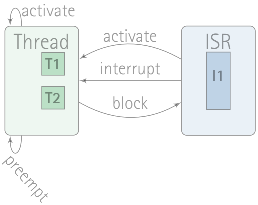


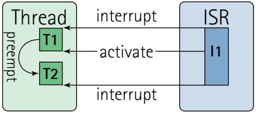


subsetting of possible instance interactions

Abstractions | Instances | Interactions



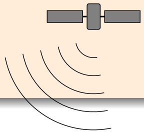


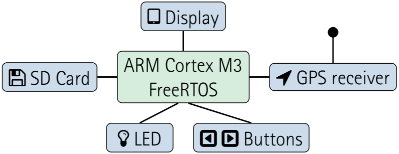


specific application implementation

Agenda

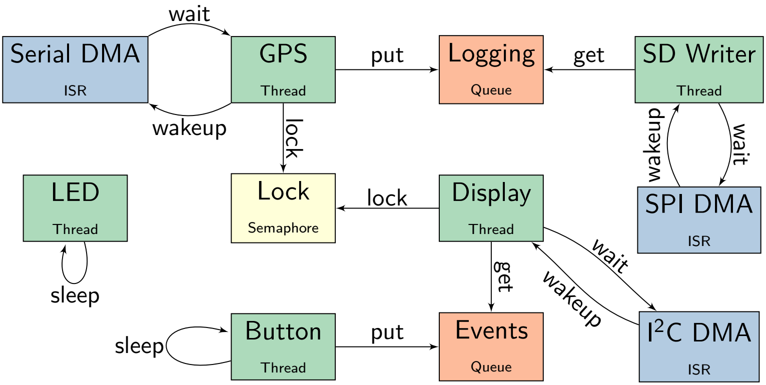
- Motivation
- Specialization
- Levels of System-Software Specialization
- Case Study: GPSLogger
- Benefits and Challenges
- Summary





https://github.com/grafalex82/GPSLogger

Case Study: GPSLogger



- Abstractions: Already done by CPP macros

	Abstractions
Text segment (bytes)	91,084
Data & BSS segment (bytes)	18,328
Startup time (cycles)	60,426

Case Study: GPSLogger

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

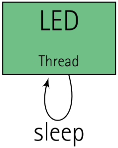
	Abstractions
Text segment (bytes)	91,084
Data & BSS segment (bytes)	18,328
Startup time (cycles)	60,426

Case Study: GPSLogger

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

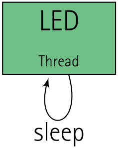
	Abstractions	Instances
Text segment (bytes)	91,084	91,196
Data & BSS segment (bytes)	18,328	17,472
Startup time (cycles)	60,426	53,827

Case Study: GPSLogger



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

	Abstractions	Instances
Text segment (bytes)	91,084	91,196
Data & BSS segment (bytes)	18,328	17,472
Startup time (cycles)	60,426	53,827



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

	Abstractions	Instances	Interactions
Text segment (bytes)	91,084	91,196	90,572
Data & BSS segment (bytes)	18,328	17,472	16,712
Startup time (cycles)	60,426	53,827	51,029

Agenda

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

Benefits

- Memory footprint reduction
  - CiAO [ISORC '14]

CiAO

CiAO is Aspect-Oriented

90% less code
50% less RAM usage

ISE SRA

Benefits

Leibniz Universität Hannover

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

CiAO

CiAO is Aspect-Oriented

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

Benefits

- Memory footprint reduction
  - CiAO [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
  - CiAO [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
  - CiAO [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

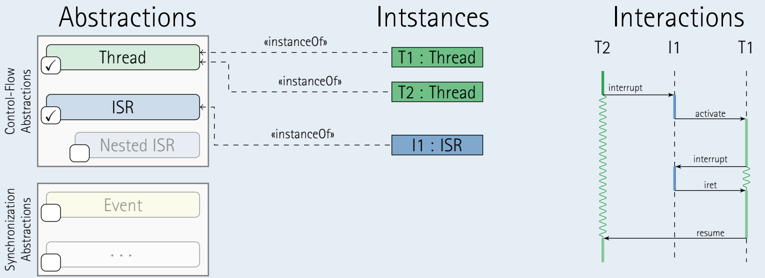
POSIX style

Challenges

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

Costs

Levels of specialization



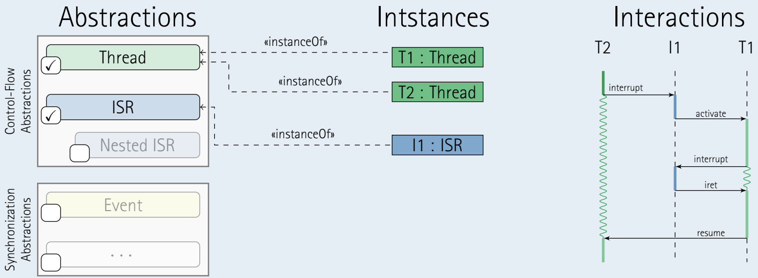
Summary

Levels of specialization

- Specialization significantly improves nonfunctional properties
- Manual specialization: too costly and time-consuming

Summary

Levels of specialization



- Specialization significantly improves nonfunctional properties
- Manual specialization: too costly and time-consuming

↪ Automation

↪ Integration in toolchain

Summary

Levels of specialization

- 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

References I

Christian Dietrich, Peter Wägemann, Peter Ulbrich, and Daniel Lohmann. "SysWCET: Whole-System Response-Time Analysis for Fixed-Priority Real-Time Systems". In: *Proceedings of the 23rd IEEE International Symposium on Real-Time and Embedded Technology and Applications (RTAS '17)*. Washington, DC, USA: IEEE Computer Society Press, 2017, pp. 37–48. ISBN: 978-1-5090-5269-1. DOI: `10.1109/RTAS.2017.37`.

Christian Dietrich and Daniel Lohmann. "OSEK-V: Application-Specific RTOS Instantiation in Hardware". In: *Proceedings of the 2017 ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES '17)* (Barcelona, Spain). New York, NY, USA: ACM Press, June 2017. DOI: `10.1145/3078633.3078637`.

Wanja Hofer, Daniel Lohmann, Fabian Scheler, and Wolfgang Schröder-Preikschat. "Sloth: Threads as Interrupts". In: *Proceedings of the 30th IEEE International Symposium on Real-Time Systems (RTSS '09)* (Washington, D.C., USA, Dec. 1–4, 2009). IEEE Computer Society Press, Dec. 2009, pp. 204–213. ISBN: 978-0-7695-3875-4. DOI: `10.1109/RTSS.2009.18`.

Wanja Hofer, Daniel Danner, Rainer Müller, Fabian Scheler, Wolfgang Schröder-Preikschat, and Daniel Lohmann. "Sloth on Time: Efficient Hardware-Based Scheduling for Time-Triggered RTOS". In: *Proceedings of the 33rd IEEE International Symposium on Real-Time Systems (RTSS '12)* (San Juan, Puerto Rico, Dec. 4–7, 2012). IEEE Computer Society Press, Dec. 2012, pp. 237–247. ISBN: 978-0-7695-4869-2. DOI: `10.1109/RTSS.2012.75`.

References II

Martin Hoffmann, Christoph Borchert, Christian Dietrich, Horst Schirmeier, Rüdiger Kapitza, Olaf Spinczyk, and Daniel Lohmann. "Effectiveness of Fault Detection Mechanisms in Static and Dynamic Operating System Designs". In: *Proceedings of the 17th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '14)* (Reno, Nevada, USA). IEEE Computer Society Press, 2014, pp. 230–237. DOI: `10.1109/ISORC.2014.26`.

Reinhard Tartler, Anil Kurmus, Bernard Heinloth, Valentin Rothberg, Andreas Ruprecht, Daniela Doreanu, Rüdiger Kapitza, Wolfgang Schröder-Preikschat, and Daniel Lohmann. "Automatic OS Kernel TCB Reduction by Leveraging Compile-Time Configurability". In: *Proceedings of the 8th International Workshop on Hot Topics in System Dependability (HotDep '12)* (Los Angeles, CA, USA). Berkeley, CA, USA: USENIX Association, 2012, pp. 1–6. URL: `https://www.usenix.org/system/files/conference/hotdep12/hotdep12-final11.pdf`.

Peter Wägemann, Christian Dietrich, Tobias Distler, Peter Ulbrich, and Wolfgang Schröder-Preikschat. "Whole-System Worst-Case Energy-Consumption Analysis for Energy-Constrained Real-Time Systems". In: *Proceedings of the 30th Euromicro Conference on Real-Time Systems 2018*. Ed. by Sebastian Altmeyer. to appear. Dagstuhl Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. DOI: `10.4230/LIPIcs.ECRTS.2018.YY`.