React in Time
Event-based Design of Time-triggered Distributed Real-time Systems

Florian Franzmann\textsuperscript{1}  Tobias Klaus\textsuperscript{1}  Fabian Scheler\textsuperscript{2}  
Peter Ulbrich\textsuperscript{1}  Wolfgang Schröder-Preikschat\textsuperscript{1}

\textsuperscript{1}Lehrstuhl für Verteilte Systeme und Betriebssysteme  
Friedrich-Alexander-Universität Erlangen-Nürnberg  
\{franzmann|klaus|ulbrich|wosch\}@cs.fau.de

\textsuperscript{2}Process Industries and Drives, Siemens AG  
fabian.scheler@siemens.com

13\textsuperscript{th} November 2015
Introduction

- Cars are becoming more autonomous $\leadsto$ more software
  - Consolidation $\leadsto$ mixed-criticality software
  $\leadsto$ Increasing safety demands
- Current trend: Event-triggered real-time systems
  - Easy to build
  - Hard to verify
Introduction

Cars are becoming more autonomous → more software
  - Consolidation → mixed-criticality software
  → Increasing safety demands

Current trend: Event-triggered real-time systems
  - Easy to build
  - Hard to verify

One solution: Time-triggered real-time systems
  - More difficult design process
  - Verified by construction
Introduction

- Cars are becoming more autonomous $\rightsquigarrow$ more software
  - Consolidation $\rightsquigarrow$ mixed-criticality software
  - Increasing safety demands
- Current trend: **Event-triggered** real-time systems
  - Easy to build
  - Hard to verify
- One solution: **Time-triggered** real-time systems
  - More difficult design process
  - Verified by construction
Problem: Important design decisions made early
- Target hardware ➞ what can be bought right now
- Target paradigm ➞ safety demands

Change after the fact expensive
~> sometimes complete redesign

Idea: postpone the decision
~> tool-based transformation

Our approach: Compiler-based
- Handles legacy software, model-based software
- Extraction of real-time properties
- Late decision w. r. t. target platform, paradigm
**Introduction**

- **Problem:** Important design decisions made early
  - Target hardware ⇔ what can be bought right now
  - Target paradigm ⇔ safety demands
- Change after the fact expensive
  ⇔ sometimes complete redesign
- **Idea:** postpone the decision
  ⇔ tool-based transformation

**Our approach:** Compiler-based

- Handles legacy software, model-based software
- Extraction of real-time properties
- Late decision w. r. t. target platform, paradigm
Abstract Representation of Real-Time Systems
  The RTSC
  Atomic Basic Blocks

Multicore
  State of the Art
  Challenges

Distributed Systems
  Target-System Model
  Partitioning Applications
The RTSC’s Pipeline

Front-End
- Simulink
- OSEK
- ...

ABB graphs

Middle-End
- Analysis
- Multi-Core
- Distributed
- Event-Triggered

ASR

Back-End
- Single-Node
- Multi-Node

Executables

System Description

App. Source Code

Target OS Descriptions

Hardware Descriptions
Abstract Representation of RTSes

**TASK (task1)**

```plaintext
...  
GetResource (SPI);  
ReceiveSPIData (&Press);  
ReleaseResource (SPI);  
...  
ActivateTask (task2);  
...  
}
```

**TASK (task2)**

```plaintext
...  
GetResource (SPI);  
SendSPIData (&conf);  
ReceiveSPIData (&data);  
ReleaseResource (SPI);  
...  
SendMessage (Message,  
&data);  
}
```

**TASK (task3)**

```plaintext
...  
ReceiveMessage (Message,  
&mydata);  
...  
}
```
Abstract Representation of RTSes

TASK (task1) {
  ...
  GetResource (SPI);
  ReceiveSPIData (&Press);
  ReleaseResource (SPI);
  ...
  ActivateTask (task2);
  ...
}

TASK (task2) {
  ...
  GetResource (SPI);
  SendSPIData (&conf);
  ReceiveSPIData (&data);
  ReleaseResource (SPI);
  ...
  SendMessage (Message, &data);
}

TASK (task3) {
  ...
  ReceiveMessage (Message, &mydata);
  ...
}
Abstract Representation of RTSes

TASK (task1){
  ...
  GetResource (SPI);
  ReceiveSPIData (&Press);
  ReleaseResource (SPI);
  ...
  ActivateTask (task2);
  ...
}

TASK (task2){
  ...
  GetResource (SPI);
  SendSPIData (&conf);
  ReceiveSPIData (&data);
  ReleaseResource (SPI);
  ...
  SendMessage (Message, &data);
  ...
}

TASK (task3){
  ...
  ReceiveMessage (Message, &mydata);
  ...
}

Global Control Flow
- Predecessor/Successor
- Directed Dependency
- Timing Information (Delay)
Abstract Representation of RTSes

**TASK (task1){**

...  
GetResource (SPI);
ReceiveSPIData (&Press);
ReleaseResource (SPI);
...  
ActivateTask (task2);
...  
}

**TASK (task2){**

...
GetResource (SPI);
SendSPIData (&conf);
ReceiveSPIData (&data);
ReleaseResource (SPI);
...
SendMessage (Message,
    &data);
}

**TASK (task3 ){**

...
ReceiveMessage (Message,
    &mydata);
...
}

---

Global Data Flow
- Predecessor/Successor
- Directed Dependency
Abstract Representation of RTSe

TASK (task1){
  ...
  GetResource (SPI);
  ReceiveSPIData (&Press);
  ReleaseResource (SPI);
  ...
  ActivateTask (task2);
  ...
}

TASK (task2){
  ...
  GetResource (SPI);
  SendSPIData (&conf);
  ReceiveSPIData (&data);
  ReleaseResource (SPI);
  ...
  SendMessage (Message, &data);
}

TASK (task3){
  ...
  ReceiveMessage (Message, &mydata);
  ...
}
Abstract Representation of RTSes

**TASK (task1)**{

... 
GetResource (SPI);

ReceiveSPIData (&Press);  

ReleaseResource (SPI);

...  
ActivateTask (task2);

...  
}

**TASK (task2)**{

...  
GetResource (SPI);

SendSPIData (&conf);  

ReceiveSPIData (&data);

ReleaseResource (SPI);

...  
SendMessage (Message, &data);

}

**TASK (task3)**{

...  
ReceiveMessage (Message, &mydata);

...  
}
Abstract Representation of RTSes

TASK (task1){
... 
  GetResource (SPI);
  ReceiveSPIData (&Press);
  SendSPIData (&conf);
  ReceiveSPIData (&data);
  ReleaseResource (SPI);
  ...
  ActivateTask (task2);
  ...
}

TASK (task2){
... 
  GetResource (SPI);
  SendSPIData (&conf);
  ReceiveSPIData (&data);
  ReleaseResource (SPI);
  ...
  SendMessage (Message, &data);
}

TASK (task3){
... 
  ReceiveMessage (Message, &mydata);
  ...
}

ff, tk, fs, pu, wosch
React in Time (13.11.2015)
Abstract Representation of RTSes

Global ABB-Graph
- Includes all dependencies
- No OS-Mechanisms
Multicore systems are up to standard today
- E.g., Automotive industry leverages multicores for consolidation
- Deploy multiple applications on same ECU
  ⇒ Consequence: mixed-criticality systems

Widely applicable algorithms for time-triggered design:
- Optimal, branch and bound, feature complete
  ⇒ Assignment: Peng et al., 1997¹ Scheduling: Abdelzaher et al., 1999²

But: Nobody uses these ⇒ Why?

Multicore

- **Multicore systems** are up to standard today
  - E.g., Automotive industry leverages multicores for consolidation
  - Deploy multiple applications on same ECU

  \[\Rightarrow\] Consequence: **mixed-criticality systems**

- Widely applicable algorithms for **time-triggered design**:
  - Optimal, branch and bound, **feature complete**

  \[\Rightarrow\] **Assignment**: Peng et al., 1997\(^1\)  **Scheduling**: Abdelzaher et al., 1999\(^2\)

- **But**: Nobody uses these \[\Rightarrow\] Why?

**Experience with RTSC:**

- Numerous adaptations for multicore necessary

---


Multicore

- Multicore systems are up to standard today
  - E.g., Automotive industry leverages multicores for consolidation
  - Deploy multiple applications on same ECU
  - Consequence: mixed-criticality systems

- Widely applicable algorithms for time-triggered design:
  - Optimal, branch and bound, feature complete
  - Assignment: Peng et al., 1997\(^1\) Scheduling: Abdelzaher et al., 1999\(^2\)

- But: Nobody uses these \⇝ Why?

Experience with RTSC:

Numerous adaptations for multicore necessary

---


**Model \(\mapsto\) Model**

ABBs derived from tasks \(\leadsto\) ABB deadline, release time too

- Deadline/release time part of cost function

\(\leadsto\) Solutions for assignment indistinguishable for assignment algo

\(\leadsto\) Cost function non-monotonous

**Solution:**

Shift deadlines/release times to honour WCETs
ABBs derived from tasks → ABB deadline, release time too
- Deadline/release time part of cost function
→ Solutions for assignment indistinguishable for assignment algo
→ Cost function non-monotonous

Solution:
Shift deadlines/release times to honour WCETs
Original scheduling algorithm shifts deadlines to enforce dependencies

Similar to our approach in assignment algorithm

Unnecessary expensive context switches

Solution:

Use explicit ready queue to enforce dependencies
Execution Environment

Original scheduling algorithm shifts deadlines to enforce dependencies

- Similar to our approach in assignment algorithm
  \( \Rightarrow \) Unnecessary expensive context switches

Solution:

Use explicit ready queue to enforce dependencies
Optimization

Original assignment algo chooses arbitrary solution if cost the same

Exploration of large parts of search space

Optimization: prefer solutions closer to algorithmic termination

Effect: Run-time reduced by almost 50 %
Optimization

Original assignment algo chooses arbitrary solution if cost the same

Exploration of large parts of search space
Optimization: prefer solutions closer to algorithmic termination

Effect: Run-time reduced by almost 50%
Multicore system only *small part* of real-world systems
Automotive system contains *multiple busses* and *many ECUs*
Interaction of *multiple* communication systems

**Idea:**
Extend the RTSC to generate *distributed time-triggered systems*
Target-System Model

- Real-time-capable multicore special case of distributed system
  - Processing nodes
  - Scratchpad memory
  - On-chip communication network

⇝ Problem solved?

Problems:

- Distributed systems often heterogeneous
- Latency load-dependent
Target-System Model

- Real-time-capable **multicore special case of distributed system**
  - Processing nodes
  - Scratchpad memory
  - On-chip communication network

\[\Rightarrow\] Problem solved?

Problems:

- Distributed systems often heterogeneous
- Latency load-dependent

\[\Rightarrow\] Detailed system model necessary

- Complete communication stack
- Subsystems
**Target-System Model**

- Real-time-capable **multicore special case of distributed system**
  - Processing nodes
  - Scratchpad memory
  - On-chip communication network

⇝ Problem solved?

**Problems:**

- Distributed systems often heterogeneous
- Latency load-dependent

⇝ **Detailed system model necessary**

- Complete communication stack
- Subsystems
Partitioning Applications

Application fragments must be assigned to nodes

Problem: Access to shared memory

- Clumsy partitioning $\leadsto$ lots of message passing
  - Competition for global communication media
  - Negative impact on other nodes

Approach: Compiler knows data flow

- Analyse to find points of minimal local state
- Then decide where to cut optimally for minimal message exchange
Partitioning Applications

Application fragments must be assigned to nodes

**Problem:** Access to shared memory

- Clumsy partitioning $\leadsto$ lots of message passing
- $\leadsto$ Competition for **global** communication media
- $\leadsto$ Negative impact on other nodes

**Approach:** Compiler knows data flow

- Analyse to find points of **minimal local state**
- Then decide where to cut optimally for **minimal message exchange**
Aim: Consolidation of legacy software

Formerly independent software on same node

New shared resources: memory, sensors, . . .

Current state:
RTSC extracts dependencies within an application

Now we need:
Inter-application dependency extraction
So far: Distributed system **within** one car

Future work: **Platooning** of multiple cars

Research focus: Codesign of **communication, control, real-time**

- Skillful partitioning of real-time applications
- Extending Matlab models for codesign
- Latency and loss model
- Generation of verifiable components
Conclusion

Real-Time Systems Compiler

- Tool for automated transformations on real-time systems
- Generates time-triggered systems \( \rightarrow \) easier verification
- Currently handles singlecore, multicore
- Numerous adaptations to existing algorithms necessary

Current work on distributed systems within one car

- Better system model necessary
- Optimal cutting of applications necessary
- Colocation of applications introduces new shared resources

Future work: Platooning of multiple cars