React in Time

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

Florian Franzmann[1] Tobias Klaus[1] Fabian Scheler[2]
Peter Ulbrich[1] Wolfgang Schröder-Preikschat[1]

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

[2]Process Industries and Drives, Siemens AG
`fabian.scheler@siemens.com`

13th November 2015

Introduction

Steve Jurvetson (cc-by-2.0)

- 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

Introduction

Steve Jurvetson (cc-by-2.0)

- Cars are becoming more autonomous $\rightsquigarrow$ more software
  - Consolidation $\rightsquigarrow$ mixed-criticality software
  - $\rightsquigarrow$ 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

Steve Jurvetson (cc-by-2.0)

- Cars are becoming more autonomous $\rightsquigarrow$ more software
  - Consolidation $\rightsquigarrow$ mixed-criticality software
- $\rightsquigarrow$ 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

- Problem: Important design decisions made early
  - Target hardware $\rightsquigarrow$ what can be bought right now
  - Target paradigm $\rightsquigarrow$ safety demands
- Change after the fact expensive
  $\rightsquigarrow$ sometimes complete redesign
- Idea: postpone the decision
  $\rightsquigarrow$ 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

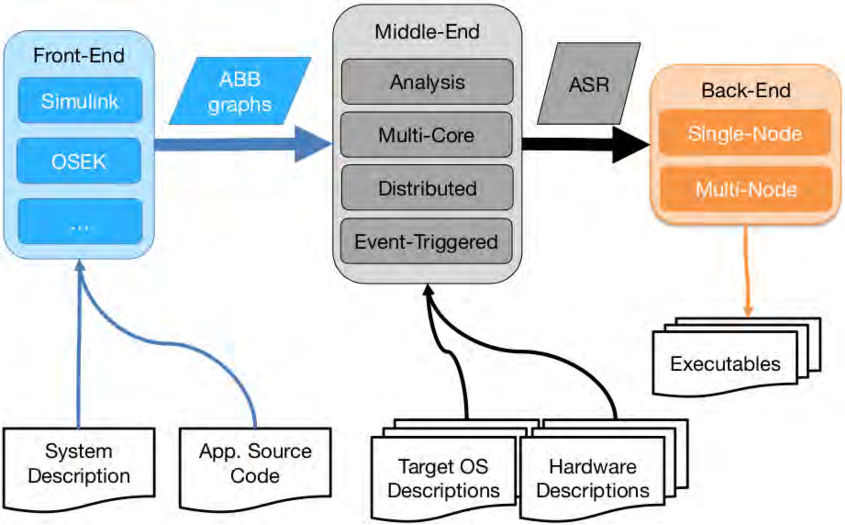
- Problem: Important design decisions made early
  - Target hardware $\rightsquigarrow$ what can be bought right now
  - Target paradigm $\rightsquigarrow$ safety demands
- Change after the fact expensive
  $\rightsquigarrow$ sometimes complete redesign
- Idea: postpone the decision
  $\rightsquigarrow$ 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

Table of contents

The RTSC's Pipeline



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);
  ...
}
```

Mutual Exclusion

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 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);
  ...
}
```

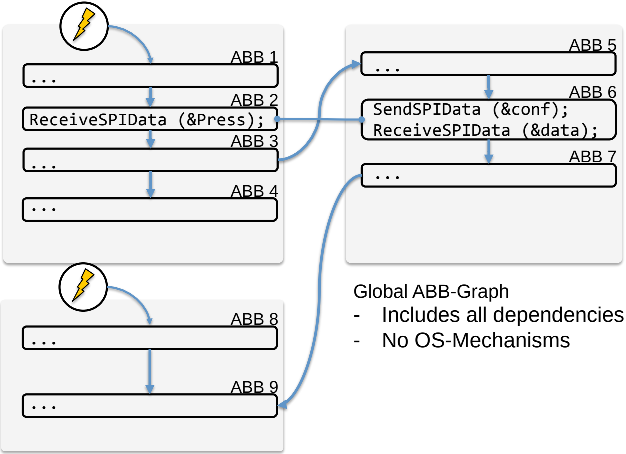
Abstract Representation of RTSes

```
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



Global ABB-Graph
- Includes all dependencies
- No OS-Mechanisms

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?

[1] Peng, Shin, Abdelzaher: *Assignment and scheduling communicating periodic tasks in distributed real-time systems*. IEEE Transactions on Software Engineering, vol. 23, no. 12, pp. 745–758, Dec 1997

[2] Abdelzaher, Shin: *Combined task and message scheduling in distributed real-time systems*. IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 11, pp. 1179–1191, Nov 1999

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

[1] Peng, Shin, Abdelzaher: *Assignment and scheduling communicating periodic tasks in distributed real-time systems*. IEEE Transactions on Software Engineering, vol. 23, no. 12, pp. 745–758, Dec 1997

[2] Abdelzaher, Shin: *Combined task and message scheduling in distributed real-time systems*. IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 11, pp. 1179–1191, Nov 1999

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?

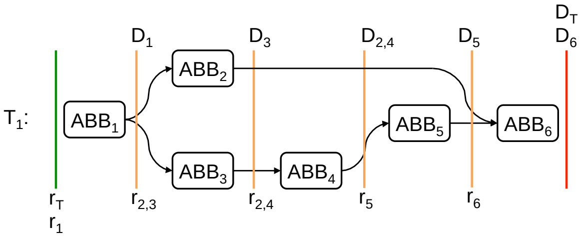
Experience with RTSC:

Numerous adaptations for multicore necessary

[1] Peng, Shin, Abdelzaher: *Assignment and scheduling communicating periodic tasks in distributed real-time systems*. IEEE Transactions on Software Engineering, vol. 23, no. 12, pp. 745–758, Dec 1997

[2] Abdelzaher, Shin: *Combined task and message scheduling in distributed real-time systems*. IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 11, pp. 1179–1191, Nov 1999

Model $\mapsto$ Model



- ABBs derived from tasks $\leadsto$ ABB deadline, release time too
  - Deadline/release time part of cost function

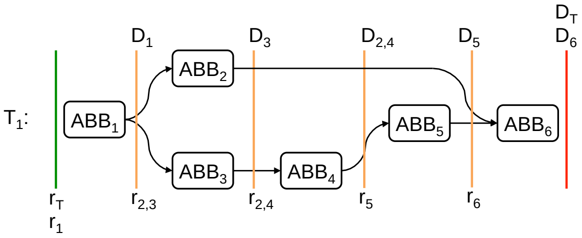
$\leadsto$ Solutions for assignment indistinguishable for assignment algo

$\leadsto$ Cost function non-monotonous

Solution:

Shift deadlines/release times to honour WCETs

Model $\mapsto$ Model



- ABBs derived from tasks $\rightsquigarrow$ ABB deadline, release time too
  - Deadline/release time part of cost function

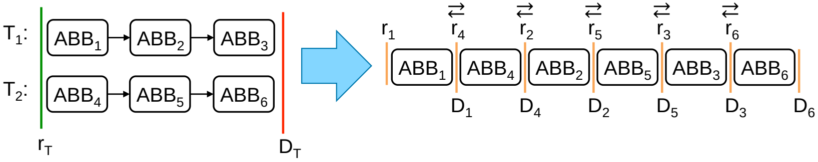
$\rightsquigarrow$ Solutions for assignment indistinguishable for assignment algo

$\rightsquigarrow$ Cost function non-monotonous

Solution:

Shift deadlines/release times to honour WCETs

Execution Environment



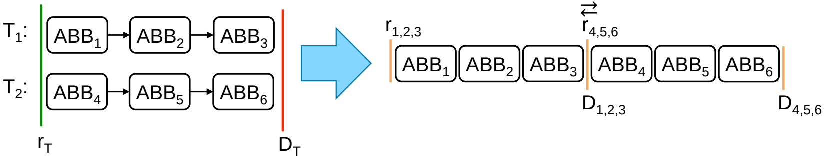
- Original scheduling algorithm shifts deadlines to enforce dependencies
  - Similar to our approach in assignment algorithm

⇝ Unnecessary **expensive** context switches

Solution:

Use explicit ready queue to enforce dependencies

Execution Environment



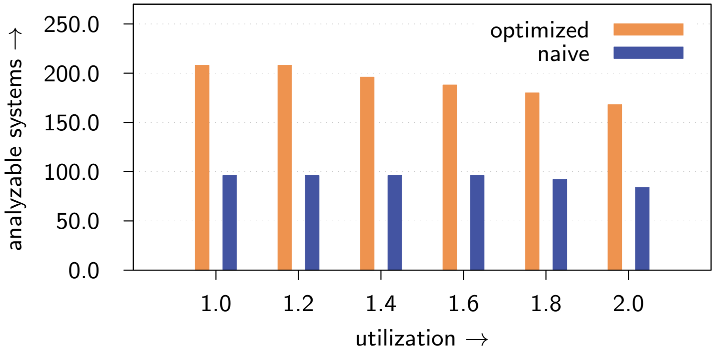
- Original scheduling algorithm shifts deadlines to enforce dependencies
  - Similar to our approach in assignment algorithm

$\rightsquigarrow$ Unnecessary **expensive** context switches

Solution:

Use explicit ready queue to enforce dependencies

Optimization



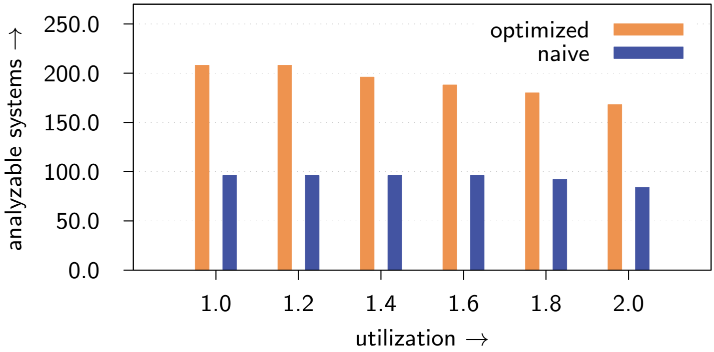
- Original assignment algo chooses arbitrary solution if cost the same

⇝ Exploration of large parts of search space

Optimization: prefer solutions closer to algorithmic termination

Effect: Run-time reduced by almost **50 %**

Optimization

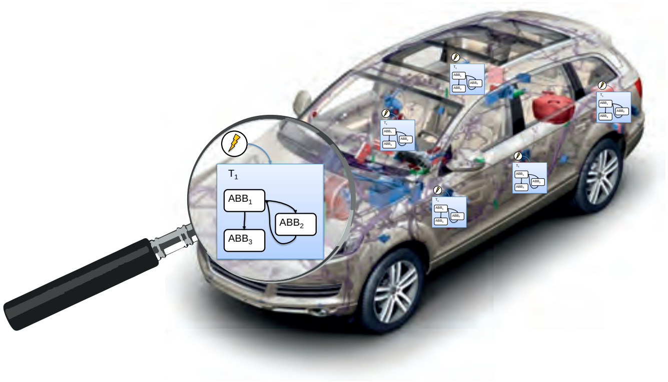


- Original assignment algo chooses arbitrary solution if cost the same

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

Effect: Run-time reduced by almost **50 %**

Distributed Systems



- 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
  - ⇝ Problem solved?

Problems:

- Distributed systems often heterogeneous
- Latency load-dependent

⇝ 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 $\rightsquigarrow$ lots of message passing

$\rightsquigarrow$ Competition for global communication media

$\rightsquigarrow$ 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 $\rightsquigarrow$ lots of message passing

$\rightsquigarrow$ Competition for global communication media

$\rightsquigarrow$ 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

Extracting Inter-Application Dependencies

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

Codesign

http://www.sartre-project.eu

- 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 $\rightsquigarrow$ 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