SMFF: System Models for Free

Moritz Neukirchner, Steffen Stein, Rolf Ernst
05.07.2011

How do we evaluate our algorithms?

How do we evaluate our algorithms?

Specific example:

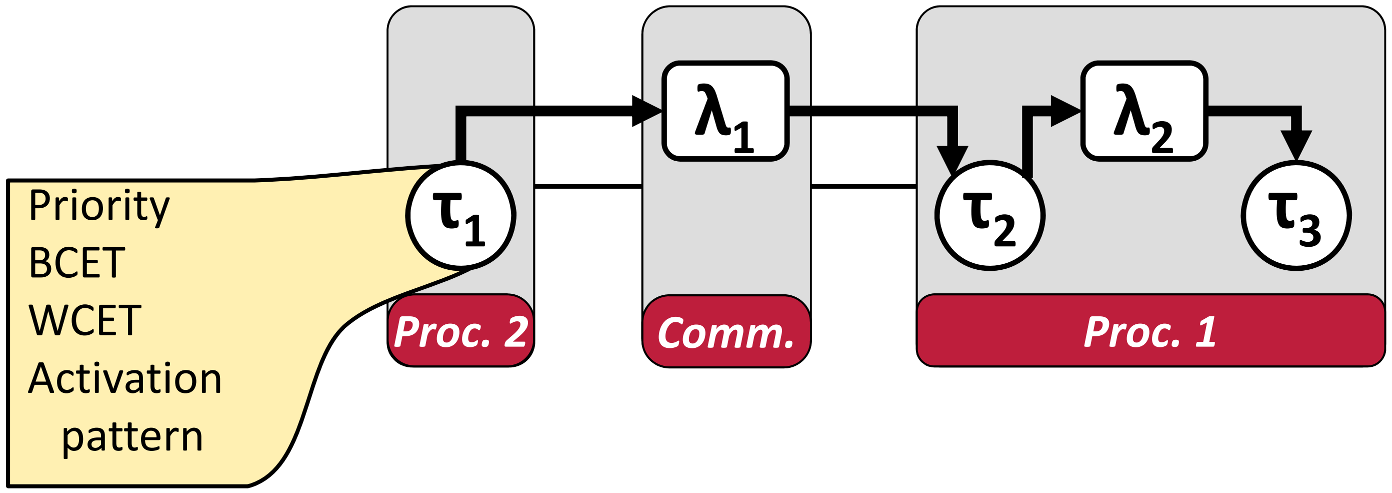
Distributed heuristic algorithm to find a priority assignment in a distributed system with end-to-end path latency constraints [1]

Questions:

- **Does the algorithm always find a solution if a solution exists?**
- **How does the runtime compare to other algorithms?**
- **Does the algorithm scale with the problem size?**

[1] M. Neukirchner, S. Stein, R. Ernst, „A Lazy Algorithm for Distributed Priority Assignment in Real-Time Systems,“ in Proc. of 2nd IEEE Workshop on Self-Organizing Real-Time Systems (SORT), 2011

System Model



Key problems:

- **Multitude of parameters**
  (Topology of application and platform, timing parameters, mapping)
- **How do we generate „typical“ system models?**

How do we evaluate our algorithms?

Evaluation methods:

• **Formal proofs** of correctness/performance	– not always possible
• Formally derived **performance bounds**	– possibly not very tight
• **Benchmark** Suites	– not always representative limited availability
• Industrial **Use-Cases**	– tested on only one/few systems
• **Handcrafted examples**	– tested on only one/few systems may not be representative
• Automatically **generated testcases**	– availability of tools? representative?

Automatic Testcase Generation

Common approaches:

- **Manual selection** of **platform** and **application** model
- **Automatic** assignment of **timing properties** (e.g. UUniFast)

- **Manual selection** of **platform** model
- **Automatic** generation of **application** model (e.g. TGFF) and of **timing properties**

Issues:

- No evaluation of **influence** of **different platforms**
- Might **not cover** common **corner-cases**
- **Limited reproducability**

No tool that allows to pseudo-randomly generate complete system models

Our Contribution

We present a **tool**

- **integrates** all steps of **testcase generation**
- allows to **customize algorithms** for testcase generation
- allows to **extend** the **system model**
- fully **seedable** for reproducability
- **no executable models**

Outline

- System Model
- Testcase Generation Steps
- Customization
- Evaluation Process

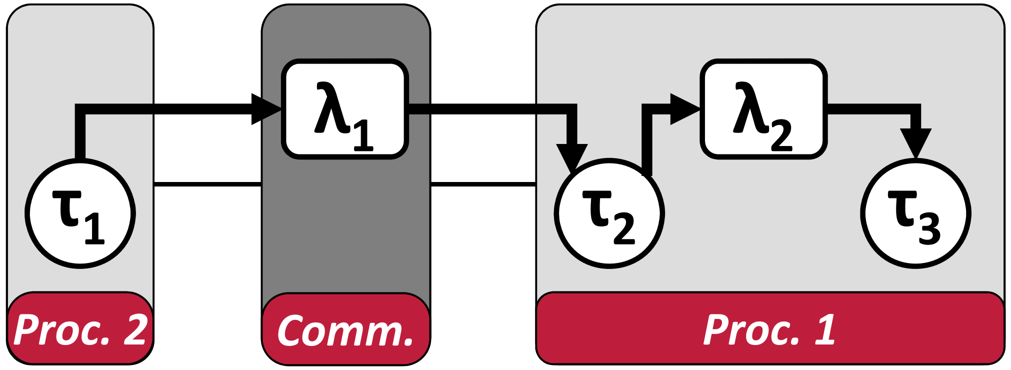
Outline

- System Model
- Testcase Generation Steps
- Customization
- Evaluation Process

System Model

Model Elements

- Platform Graph
  (bipartite graph of processors and busses)
- Application Graphs
  (bipartite graph of tasks and task links)
- Mapping of Tasks to Resources
  - Tasks to processors
  - Task links to processors or busses



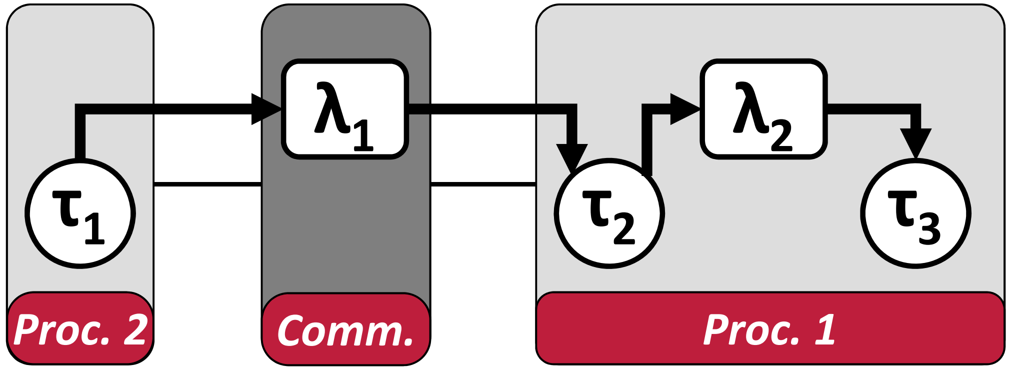
System Model

Timing Properties

- Best-case & worst-case execution time
- Activation pattern (e.g. period and jitter)
- Constraints on end-to-end latencies, worst-case response time, jitter

Scheduling Parameters

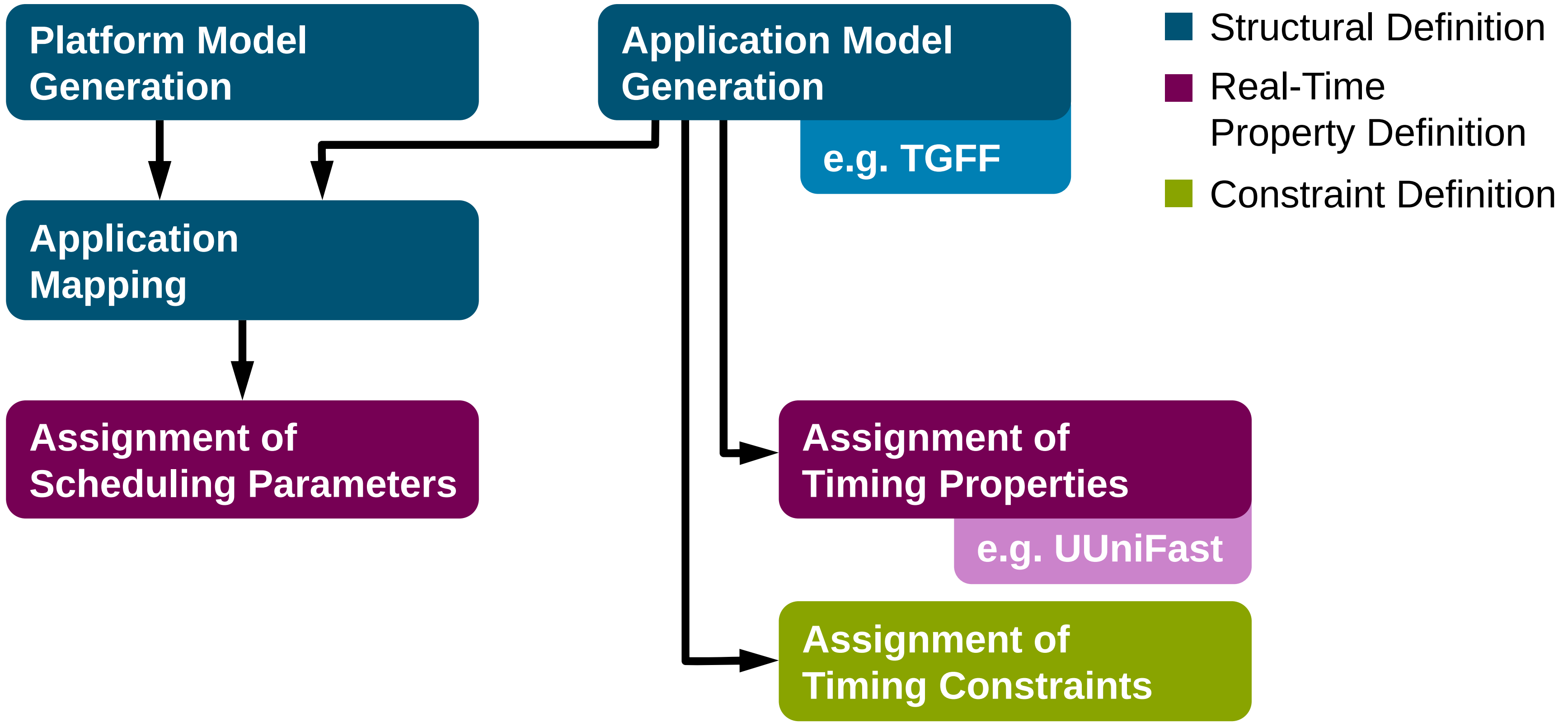
- Scheduler for each resource (e.g. static priority preemptive)
- Scheduling parameters for tasks and task links (e.g. priorities)



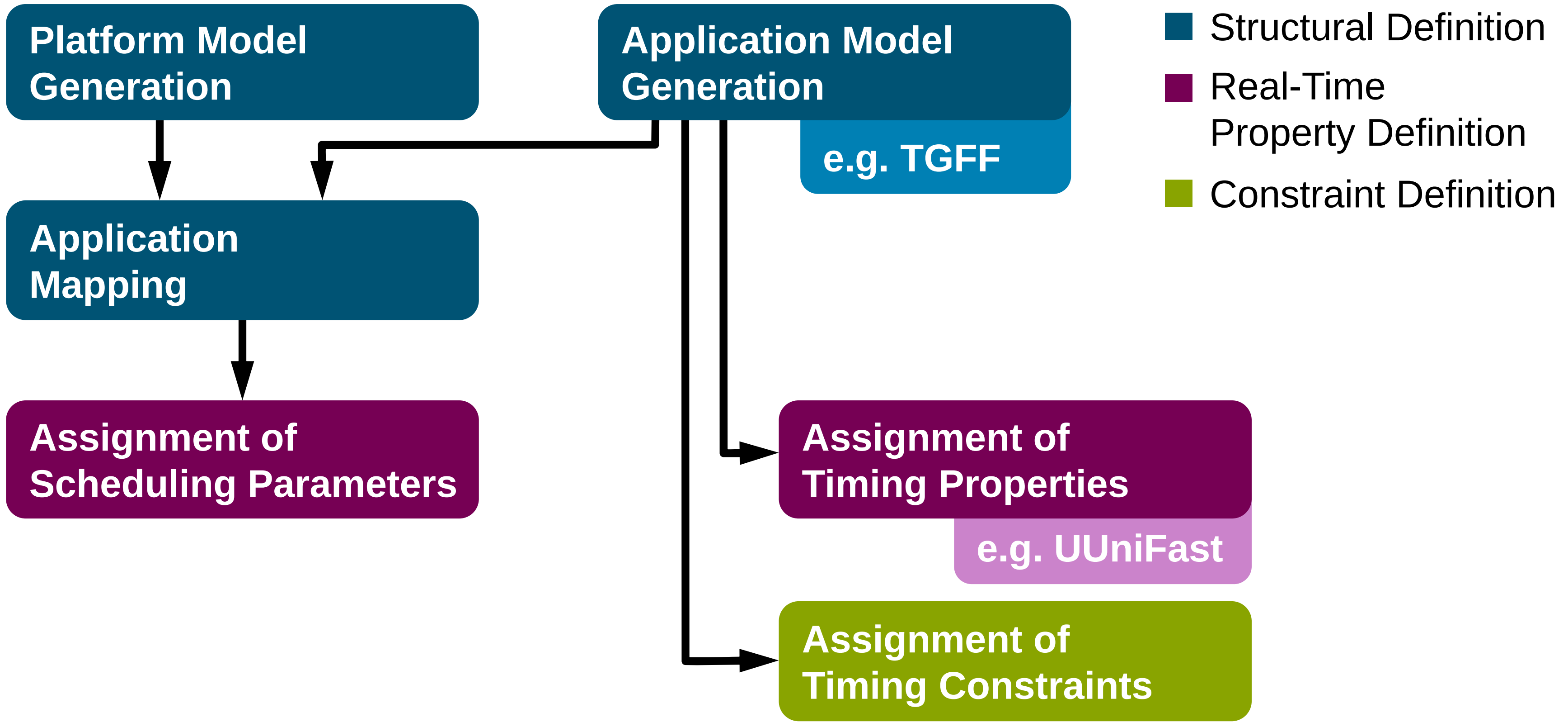
Outline

- System Model
- Testcase Generation Steps
- Customization
- Evaluation Process

Testcase Generation Steps


Platform Model Generation
Application Model Generation
e.g. TGFF
Structural Definition
Real-Time Property Definition
Constraint Definition
Application Mapping
Assignment of Scheduling Parameters
Assignment of Timing Properties
e.g. UUniFast
Assignment of Timing Constraints

Testcase Generation Steps


Platform Model Generation
Application Model Generation
e.g. TGFF
Application Mapping
Assignment of Scheduling Parameters
Assignment of Timing Properties
e.g. UUniFast
Assignment of Timing Constraints
Structural Definition
Real-Time Property Definition
Constraint Definition

Testcase Generation Steps

Platform Model Generation

Application Model Generation

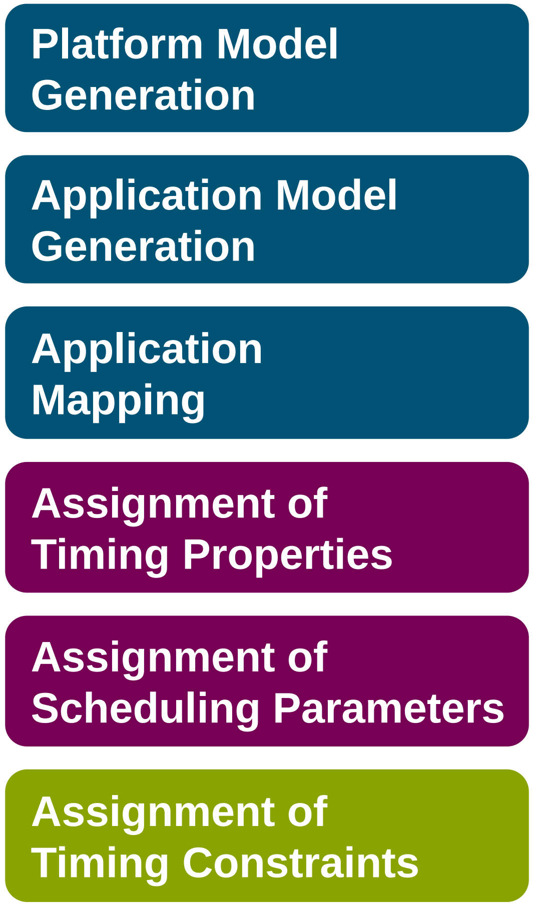
Application Mapping

Assignment of Scheduling Parameters

Assignment of Timing Properties

Assignment of Timing Constraints

Testcase Generation Steps


Platform Model Generation
Application Model Generation
Application Mapping
Assignment of Timing Properties
Assignment of Scheduling Parameters
Assignment of Timing Constraints

Testcase generation sequence **different from** typical **design sequence**

1. Timing constraints from specification
2. Timing properties immediate result of implementation and mapping
3. Scheduling parameters last

Senseful sequence of generation steps:

- Mapping before timing properties, to avoid overload situations
- Timing constraints as last steps to generate feasible systems/to define laxity of constraints

Testcase Generation Steps

Platform Model Generation

Application Model Generation

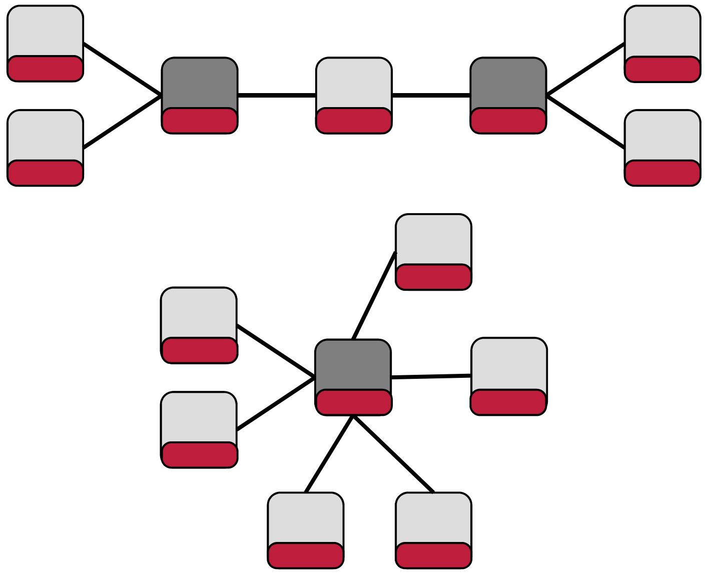
Application Mapping

Assignment of Timing Properties

Assignment of Scheduling Parameters

Assignment of Timing Constraints

- Number and type of processors?
- Communication topology and types of communication media?
- Scheduling mechanisms on platform components?



Testcase Generation Steps

- **Platform Model Generation**
- **Application Model Generation**
- **Application Mapping**
- **Assignment of Timing Properties**
- **Assignment of Scheduling Parameters**
- **Assignment of Timing Constraints**

Provided algorithm:

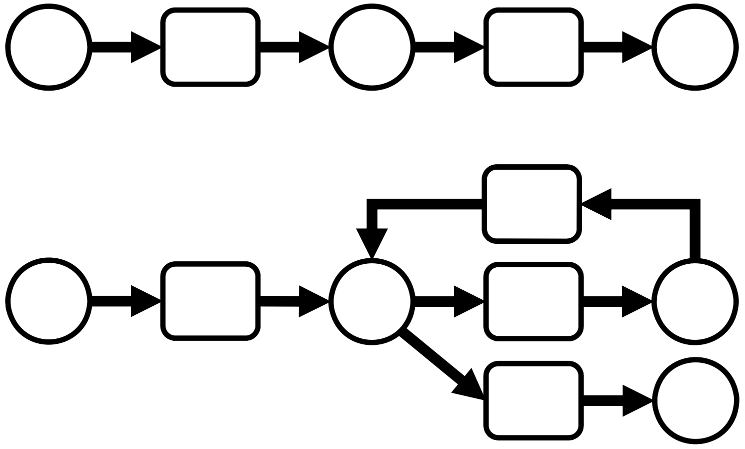
- Specification of number of processors
- Specification of mean number of communication resources (as percentage of number of processors)

⇒ Allows to influence the „connectivity" of the platform
(bus-like vs. networked structures)

Testcase Generation Steps

- Platform Model Generation
- **Application Model Generation**
- Application Mapping
- Assignment of Timing Properties
- Assignment of Scheduling Parameters
- Assignment of Timing Constraints

- Communication among tasks?
- Functional cycles?
- Forks and joins?



Testcase Generation Steps

- Platform Model Generation
- **Application Model Generation**
- Application Mapping
- Assignment of Timing Properties
- Assignment of Scheduling Parameters
- Assignment of Timing Constraints

Two algorithms provided:

1. Based on task graphs for free (TGFF) allows parametric task-graph generation
2. Generating task chains of defined length

Technische Universität Braunschweig

Testcase Generation Steps

Platform Model Generation

Application Model Generation

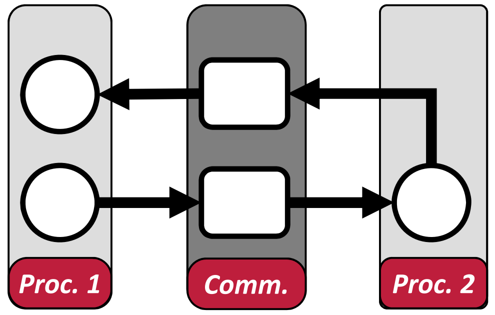
Application Mapping

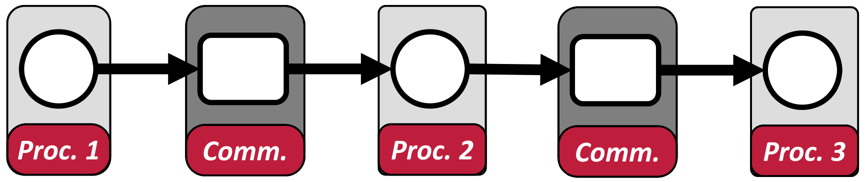
Assignment of Timing Properties

Assignment of Scheduling Parameters

Assignment of Timing Constraints

- Tasks clustered or distributed?
- Request-Response topology?
- Sensor-Actuator topology?





Testcase Generation Steps

- Platform Model Generation
- Application Model Generation
- **Application Mapping**
- Assignment of Timing Properties
- Assignment of Scheduling Parameters
- Assignment of Timing Constraints

Provided algorithm:

- Maps task chains only
- Spreads tasks across several resources
- Tasks are only on the same resource if they are predecessor or successor of each other

⇒ Sensor-Actuator mapping

Testcase Generation Steps

Platform Model Generation

Application Model Generation

Application Mapping

Assignment of Timing Properties

Assignment of Scheduling Parameters

Assignment of Timing Constraints

- BCET, WCET?
- Activation patterns?

Technische Universität Braunschweig

Testcase Generation Steps

- Platform Model Generation
- Application Model Generation
- Application Mapping
- **Assignment of Timing Properties**
- Assignment of Scheduling Parameters
- Assignment of Timing Constraints

Provided algorithm:

- based on UUniFast [2]
- assigns activation periods in specified window
- assigns WCETs such that resource utilization is at specified value

[2] E. Bini, G. Buttazzo, „Measuring the Performance of Schedulability Tests," in Real-Time Syst., 2005

Testcase Generation Steps

Platform Model Generation

Application Model Generation

Application Mapping

Assignment of Timing Properties

Assignment of Scheduling Parameters

Assignment of Timing Constraints

- Priorities?
- Time-Slots?
- Budgets?

Technische Universität Braunschweig

Testcase Generation Steps

Platform Model Generation

Application Model Generation

Application Mapping

Assignment of Timing Properties

Assignment of Scheduling Parameters

Assignment of Timing Constraints

Provided Algorithms:

1. Random assignment of priorities
2. Priority assignment such that tasks in a chain receive priorities in decreasing order (to reduce functional cycles)

Testcase Generation Steps

Platform Model Generation

Application Model Generation

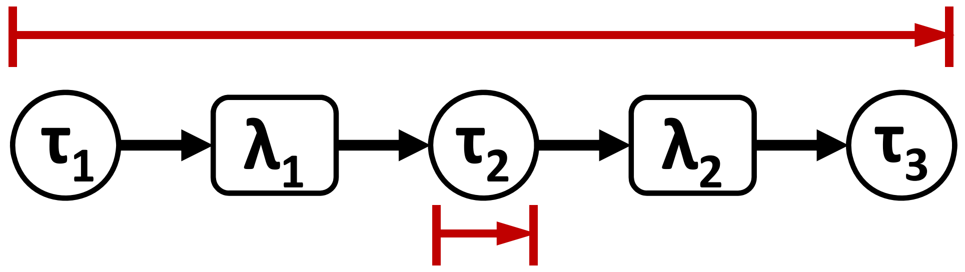
Application Mapping

Assignment of Timing Properties

Assignment of Scheduling Parameters

Assignment of Timing Constraints

- Worst-case response time?
- Jitter?
- End-to-end latency?



Testcase Generation Steps

- Platform Model Generation
- Application Model Generation
- Application Mapping
- Assignment of Timing Properties
- Assignment of Scheduling Parameters
- **Assignment of Timing Constraints**

Provided algorithm:

- Generates path latency constraint
- Constraint as multiple of sum of WCETs along the path

Isn‘t this very limited?

YES!

Outline

- System Model
- Testcase Generation Steps
- Customization
- Evaluation Process

Model Extensions

- Tool programmed in Java
- Every **model element** can be **extended by own class**
- Allows to **extensions** in **data** and **function**

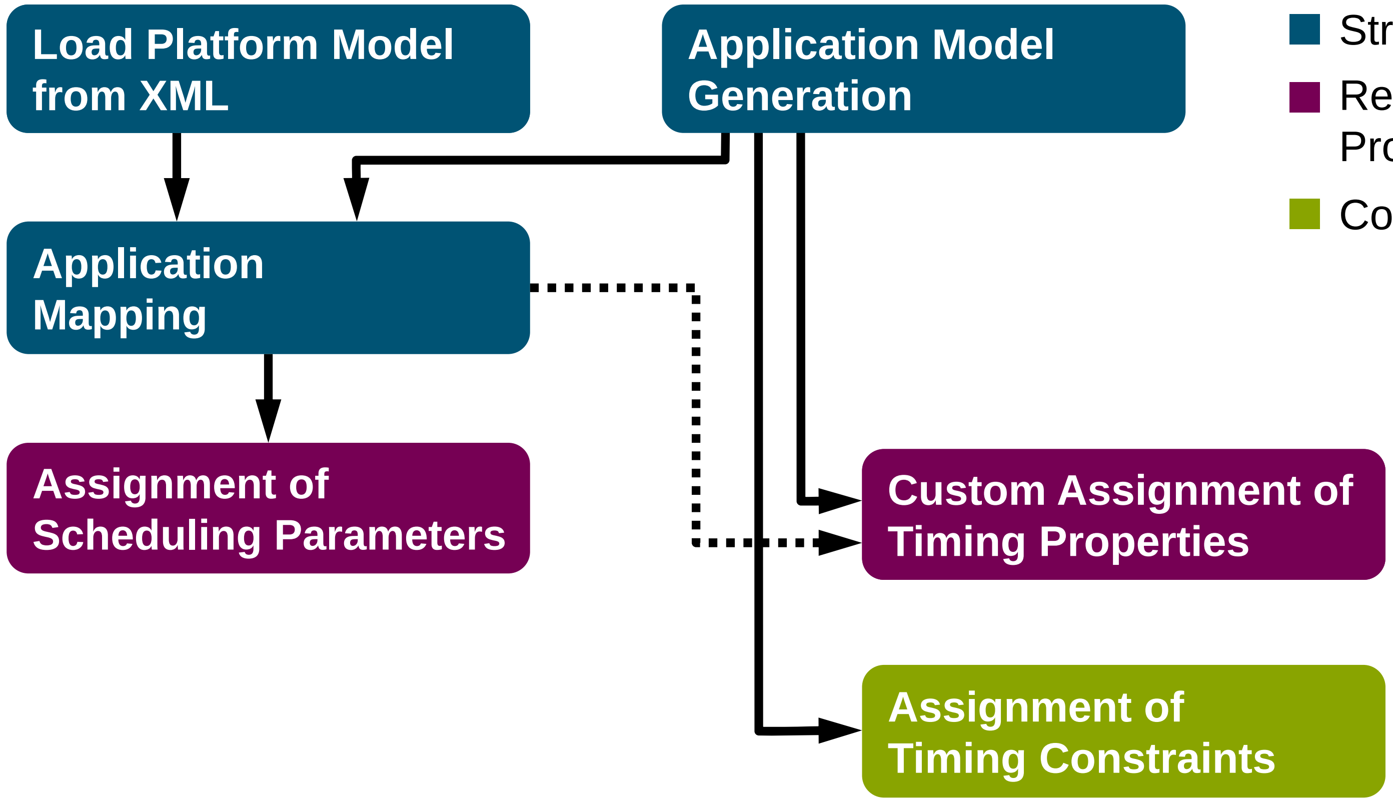
- Declaration of extension by inheritance of abstract extension class

```
public class WCRTConstraint extends AbstractDataExtension {…}
```

- Adding, Deleting and Querying of extensions in a type-safe manner

```
task.getExtDataByClass(WCRTConstraint.class)
```

Algorithm Customization



Load Platform Model from XML
Application Model Generation
Application Mapping
Assignment of Scheduling Parameters
Custom Assignment of Timing Properties
Assignment of Timing Constraints

🟦 Structural Definition

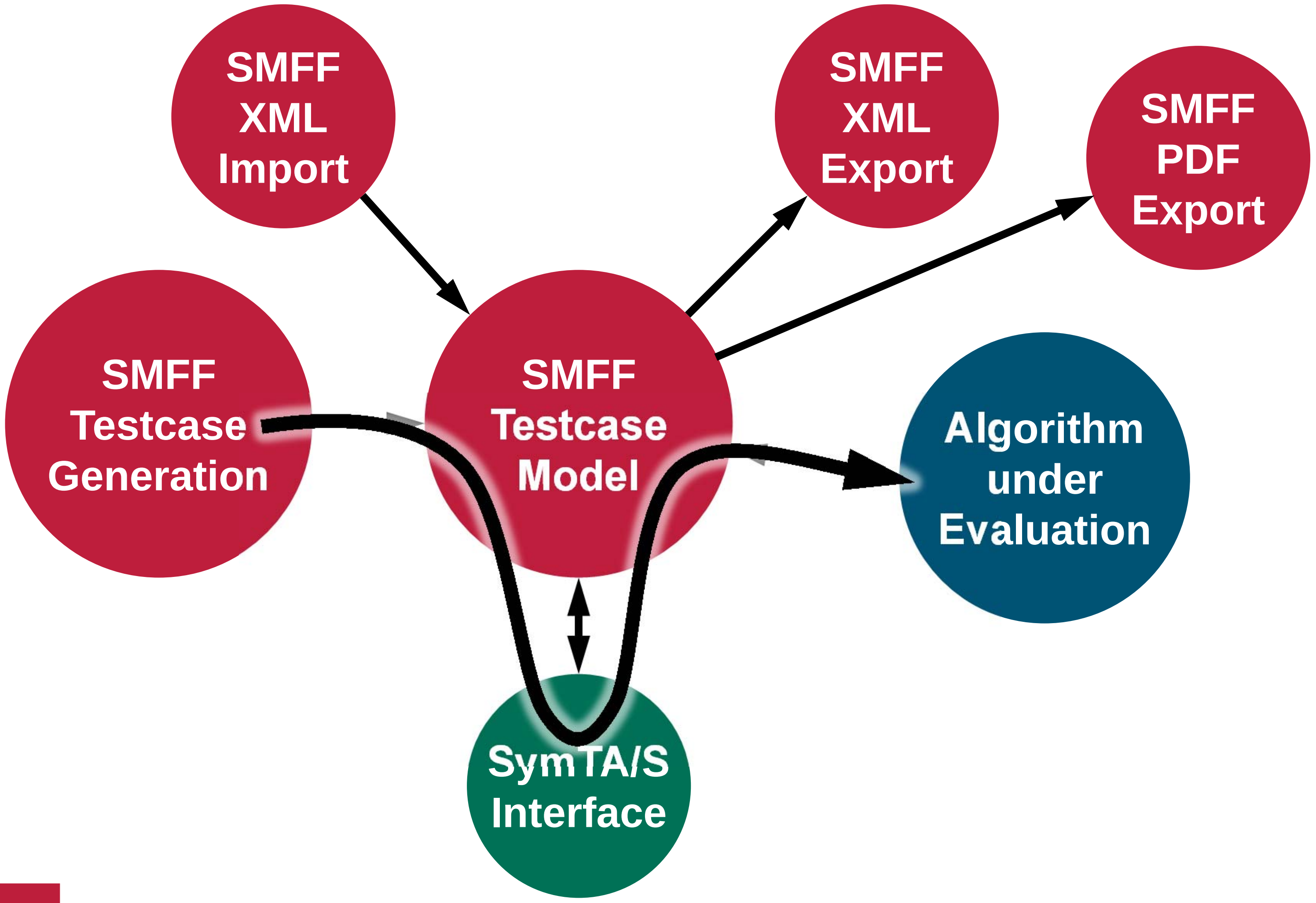
🟪 Real-Time Property Definition

🟩 Constraint Definition

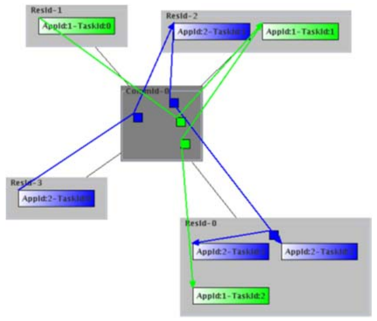
Outline

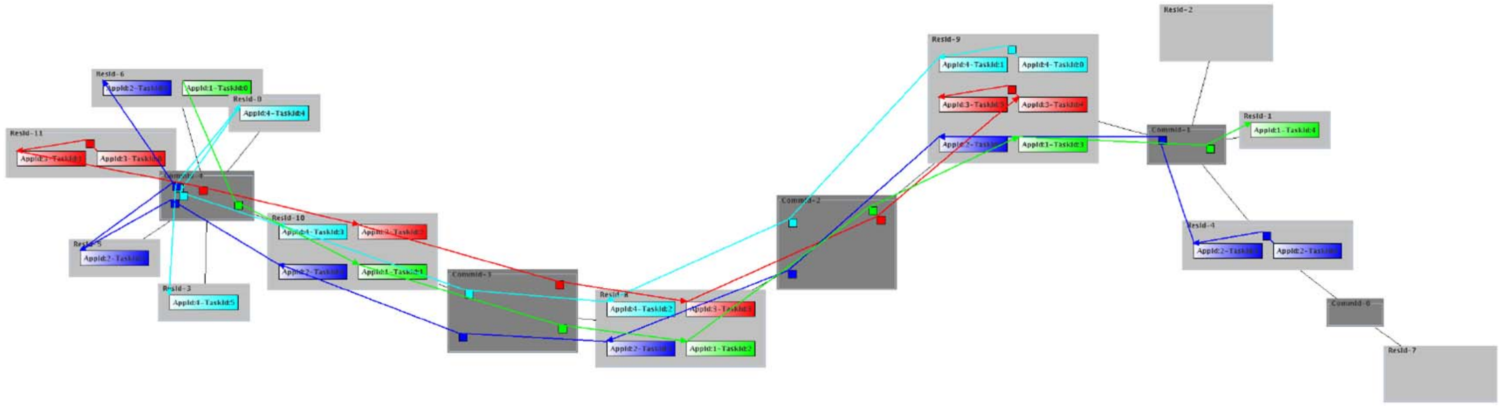
- System Model
- Testcase Generation Steps
- Customization
- Evaluation Process

Evaluation Process



Pdf Output


ResId-1
AppId:1-TaskId:0
ResId-2
AppId:1-TaskId:1
CommId-0
ResId-3
ResId-0
AppId:1-TaskId:2


ResId-2
ResId-9
ResId-6
ResId-11
ResId-5
ResId-3
ResId-10
ResId-8
ResId-1
ResId-4
ResId-7
CommId-2
CommId-3
CommId-0

Conclusion

SMFF allows to

- **generate** completely specified **testcase system models**
- **extend** the **system model** by implementing own extensions
- use **custom testcase generation** algorithms by inheritance from abstract factories
- reorder generation steps to introduce additional dependencies in testcase generation

Output as

- XML file (including custom model extensions)
- PDF file

Tool available at:

http://smff.sourceforge.net

Thank you for your attention.

Technische Universität Braunschweig