A Design Method for Modular Energy-Aware Software

OUrsi @ OU.NL

March 31, 2015

Christoph Bockisch
(christoph.bockisch@ou.nl)

Research Overview

Software Engineering Method for Energy-Aware Systems

Tool support

Conclusion

Career

TECHNISCHE UNIVERSITÄT DARMSTADT

2003 – 2008 PhD studies

Dissertation: An Efficient and Flexible Implementation of Aspect-Oriented Languages

UNIVERSITEIT TWENTE.

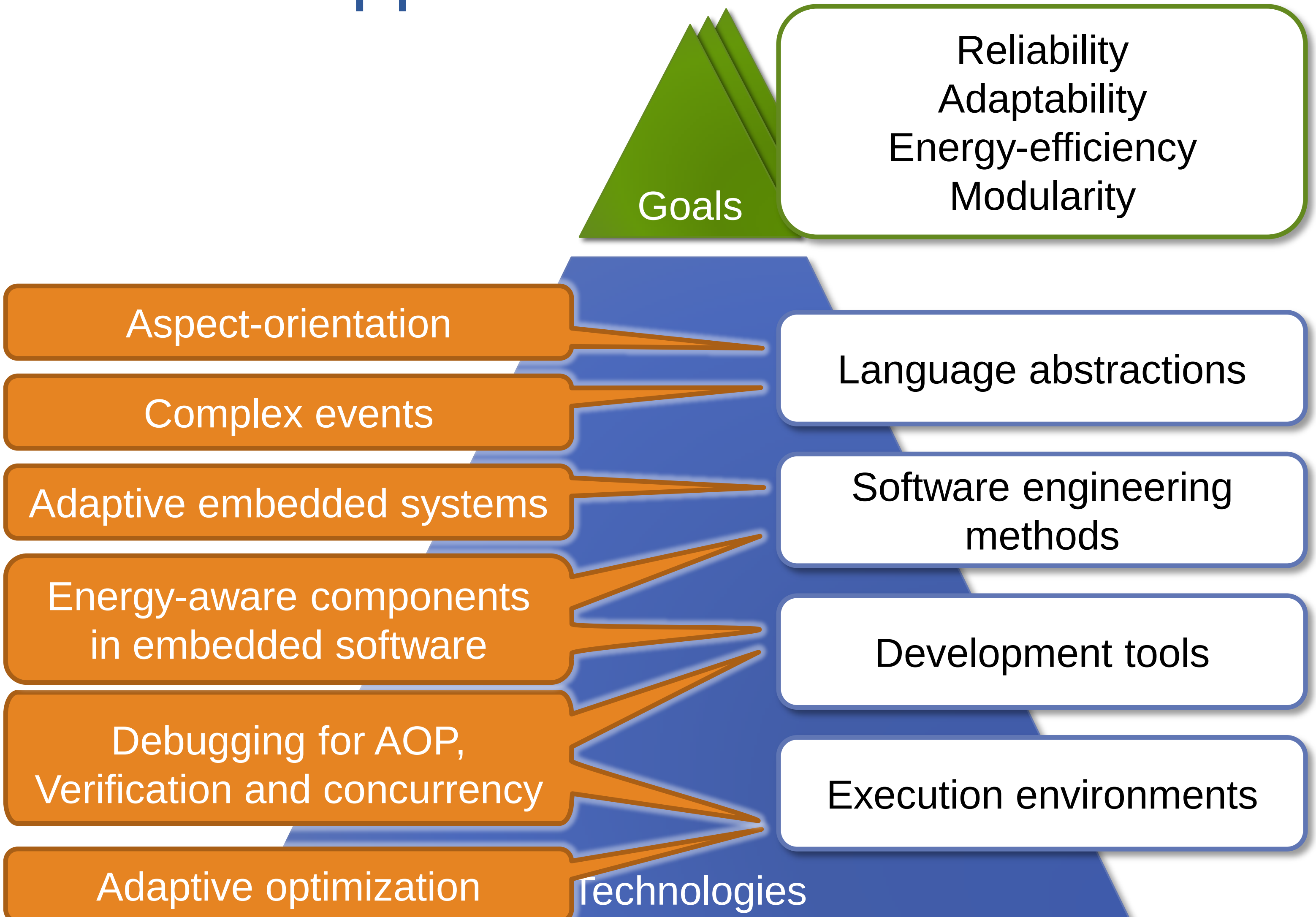
2009 – 2014 Assistant Professor for Software Composition

- Software architectures for reliability & adaptivity
- Energy-optimization for embedded systems
- Language technology for aspect-oriented programming

since 2014 Assistant Professor in Software Engineering

- Data analytics in education
- Energy-optimization in software
- Verification in concurrent systems

Research Approach



Engineering Energy-Aware Embedded Software

- Common goal in software engineering: **modularity**
- Energy issues do **not respect module boundaries**
  - They are a cross-cutting concern
  - Conventional approaches cannot separate energy-related code

Approach: method for **systematic design of energy-aware embedded software**

- Make **resources** explicit at **component interface** (energy is one possible resource)
- Facilitate implementing energy-optimization in separate components
- **Adapt & adopt tools** to support design process

joint work with

océ

Printing for Professionals

Overview

Method

Tooling

Conclusion

Project Scope

Our focus

- Software controlling energy-consuming devices/resources (Printer parts, mobile device components/activities, etc.)
- Modular implementation of energy-related code

Not our focus

- Reducing energy consumption of program execution itself
- Inventing new optimization algorithms

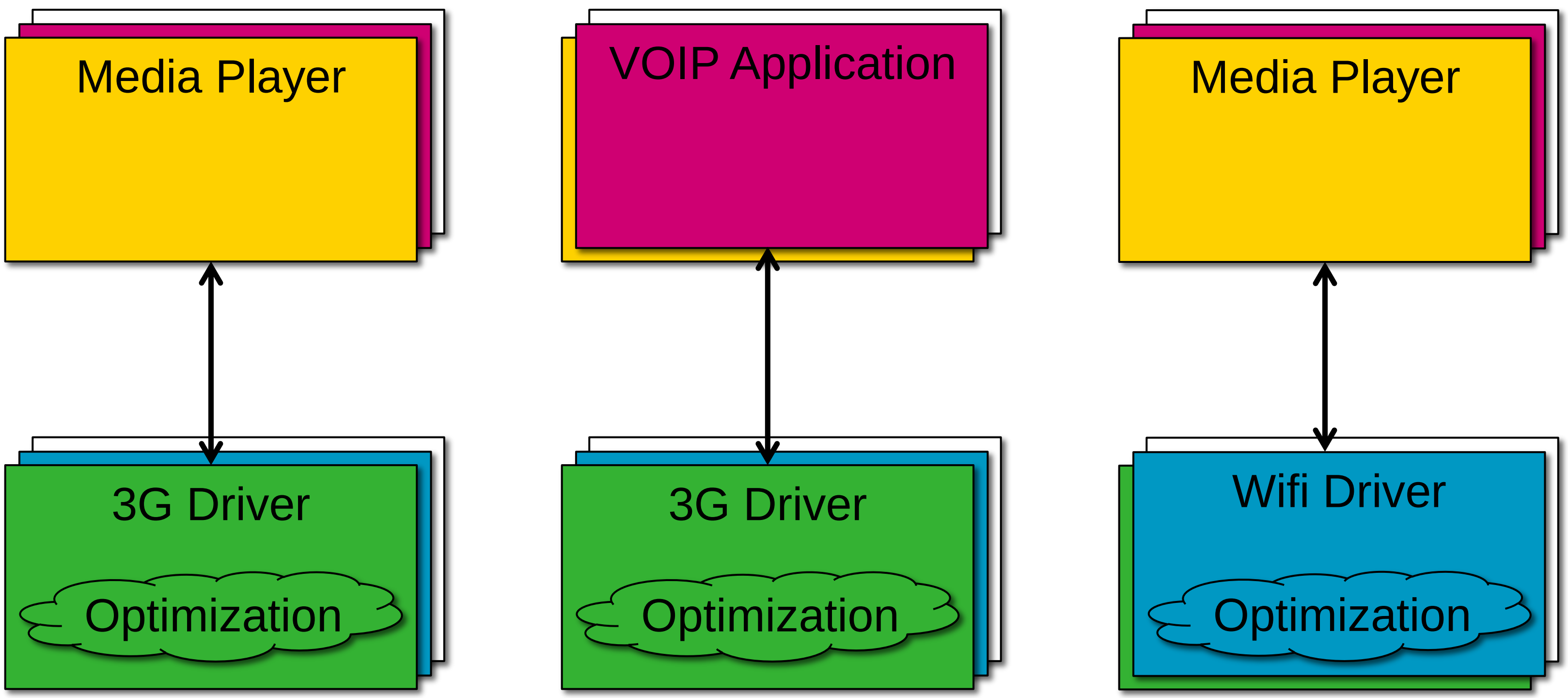
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network

S. Malakuti, S. te Brinke, L. Bergmans, and C. Bockisch. **Towards Modular Resource-Aware Applications**. In: *VariComp* 2012

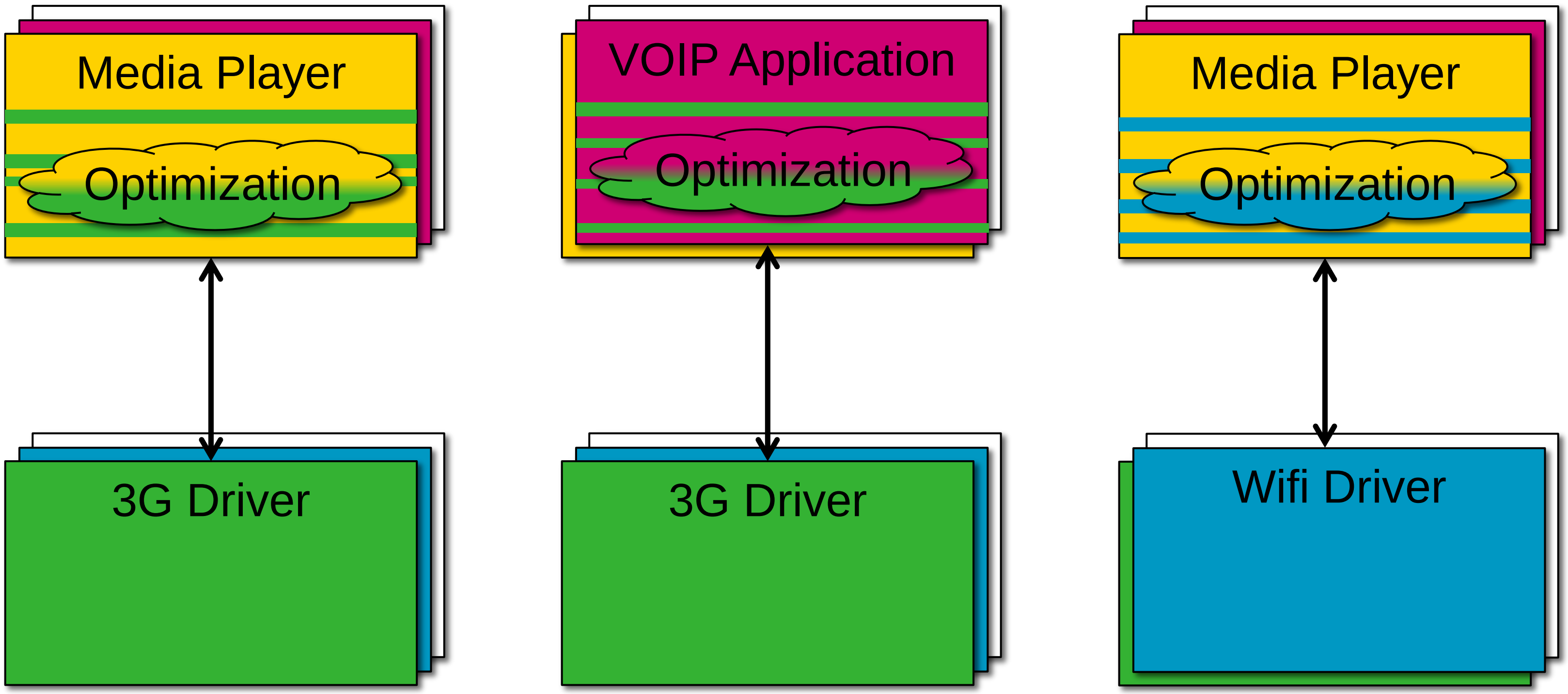
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network



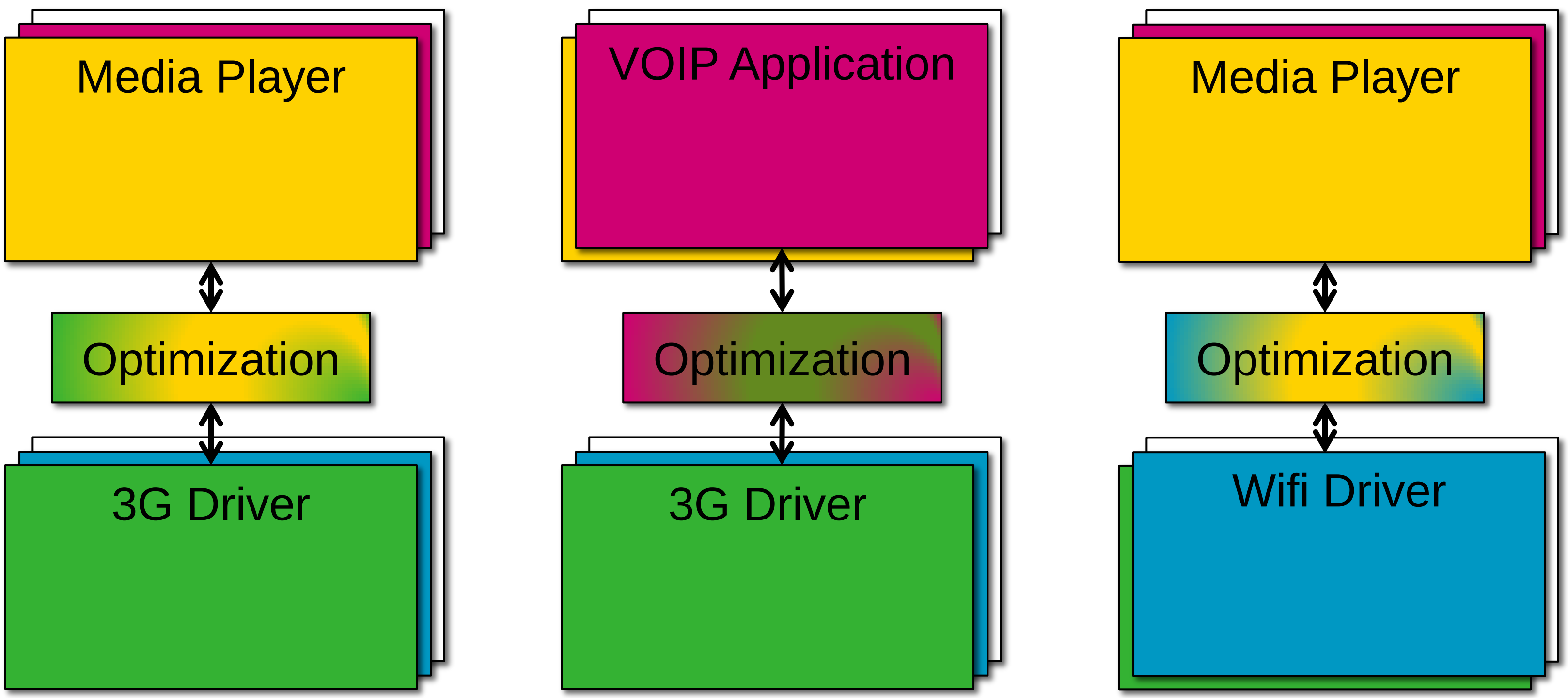
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network



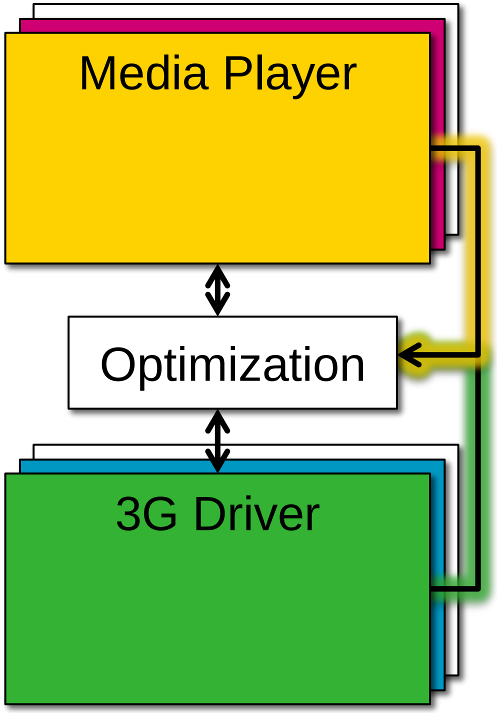
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network



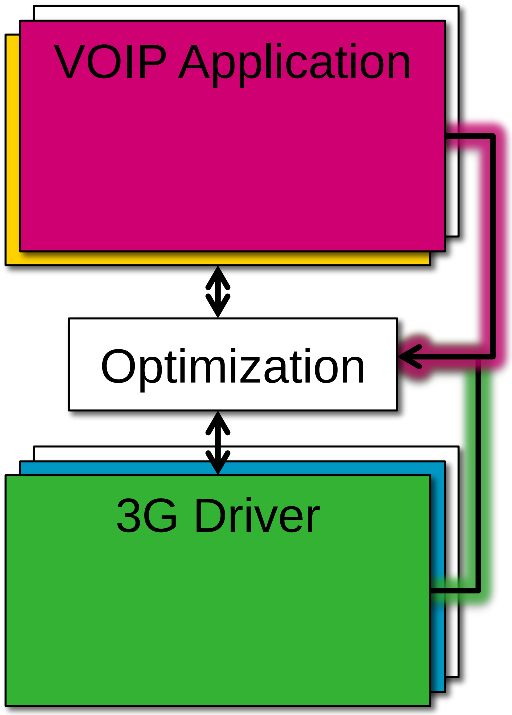
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network



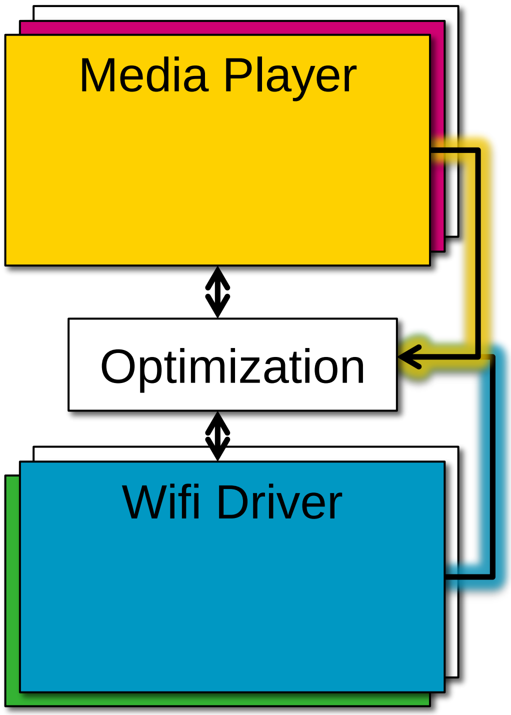
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network



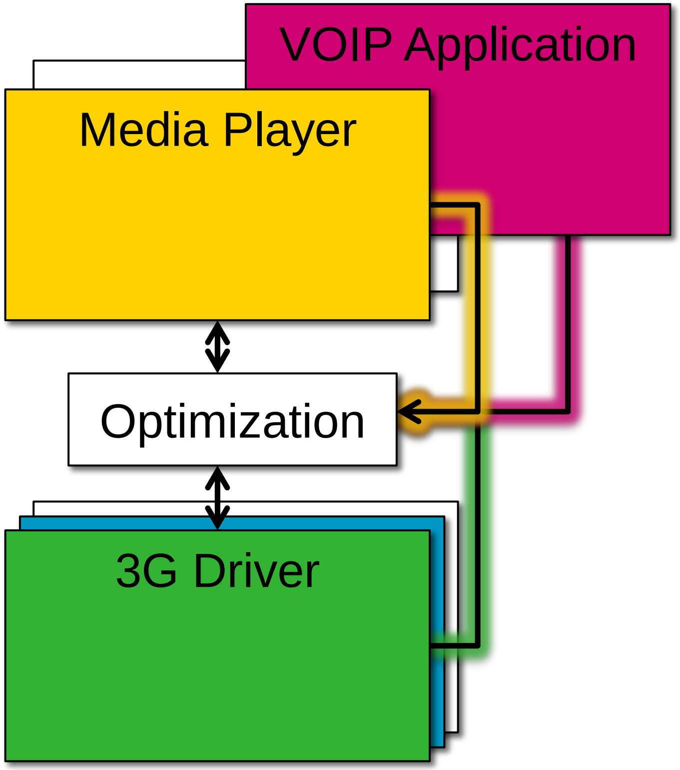
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network



Case Study: Smart Phone

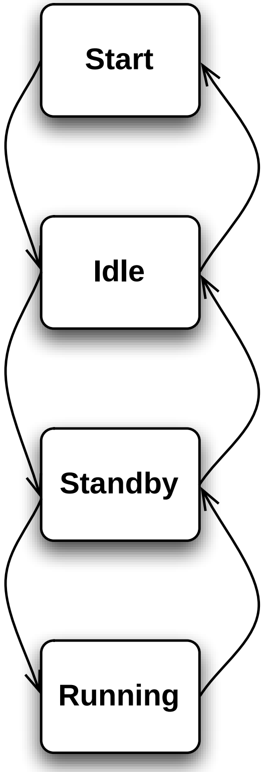
Media player on a mobile phone, streaming music over the network



Case Study: Professional Printers

- Industrial case: Océ

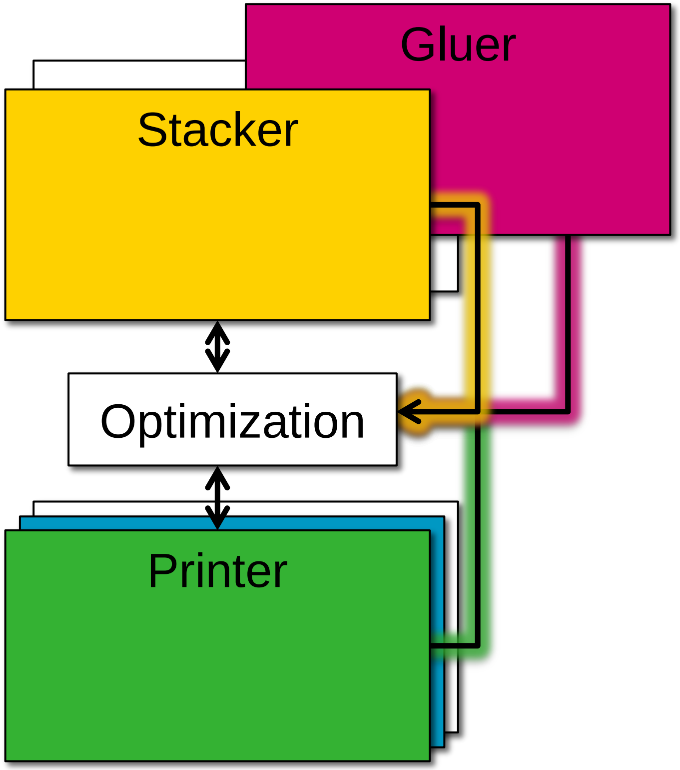
- Printer has few main states (start, idle, standby, running)
- All finishers have similar states
- All finishers must be in the same state
- Otherwise, system complexity unmanageable



- Problem statement
  - Gluer can have hot or cold glue
  - Leads to two separate running states
  - Increases number of states of ***all*** finishers
  - Increases complexity

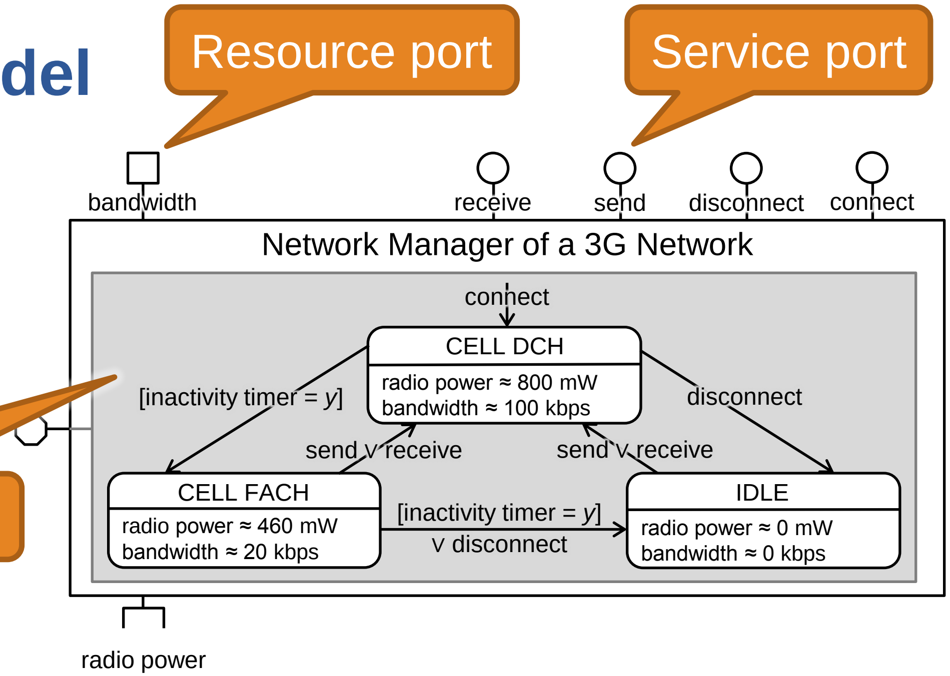
Case Study: Professional Printers

- Printer is connected to many finishers
- Finisher can be connected to various printers



Resource-Aware Component Interface

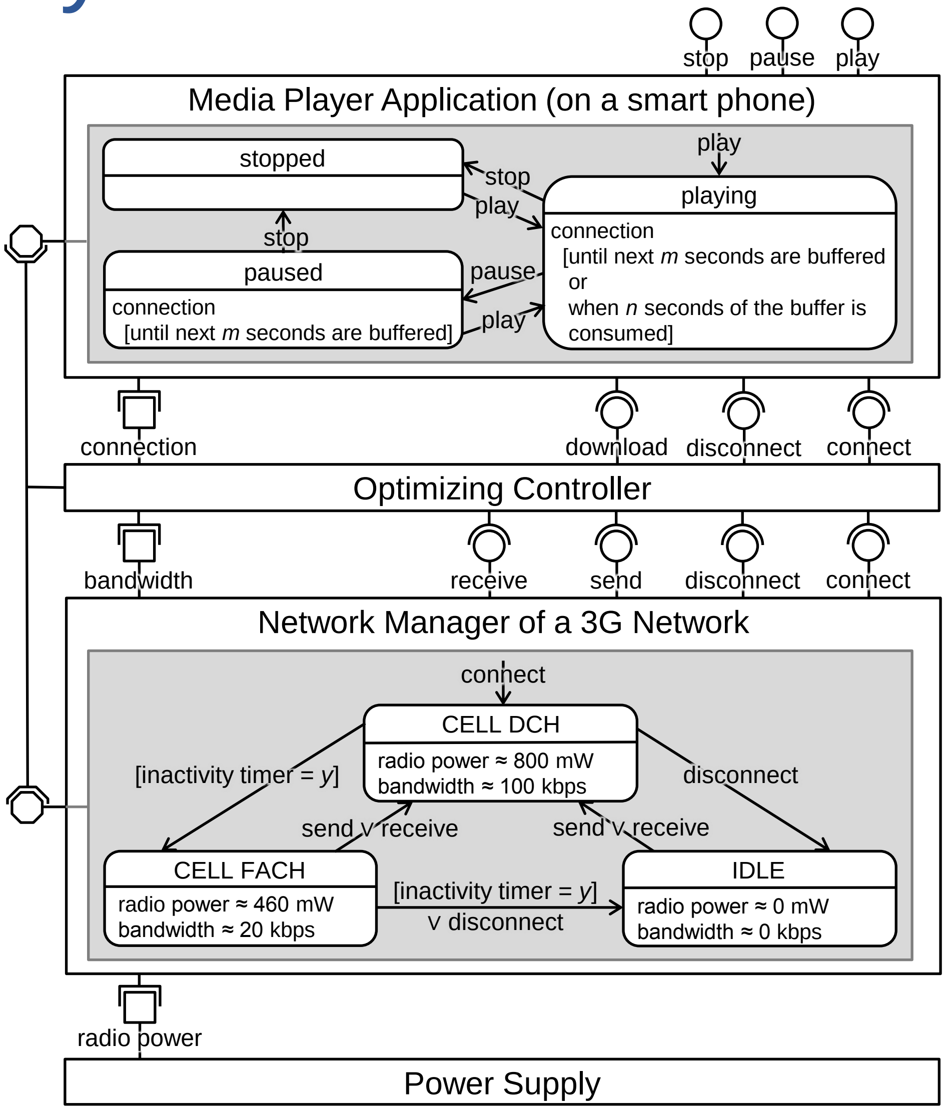
- Dedicated **component model**
  - Resource ports
    Service ports
  - Resource Utilization Model (RUM)



- RUM defined as **state chart**
  - States model stable resource usage
  - Services or internal events trigger transitions

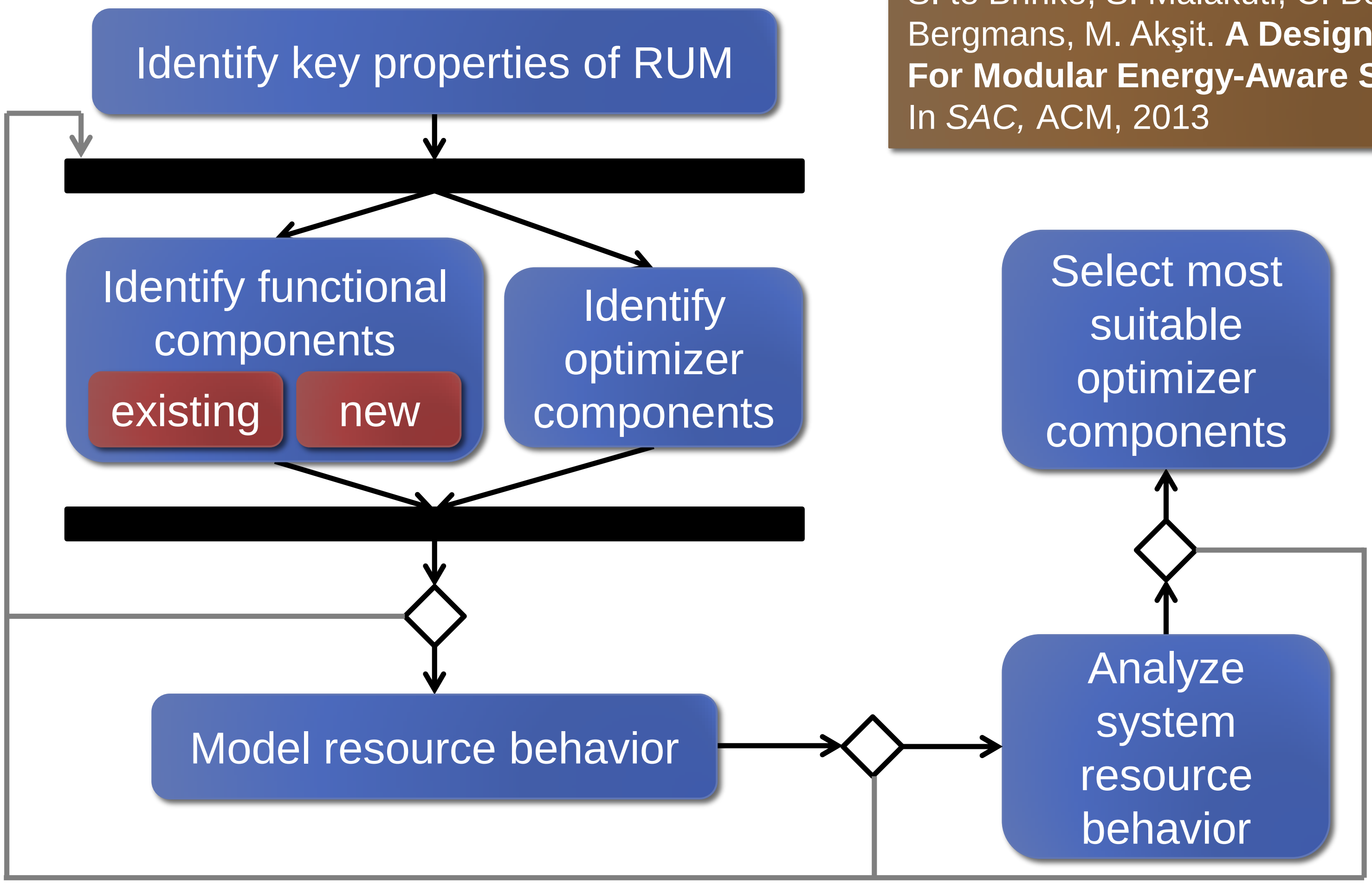
Overview
Method
Tooling
Conclusion

Case Study: Smart Phone



Design Method for Energy-Aware Embedded Software

S. te Brinke, S. Malakuti, C. Bockisch, L. Bergmans, M. Akşit. **A Design Method For Modular Energy-Aware Software**. In *SAC*, ACM, 2013



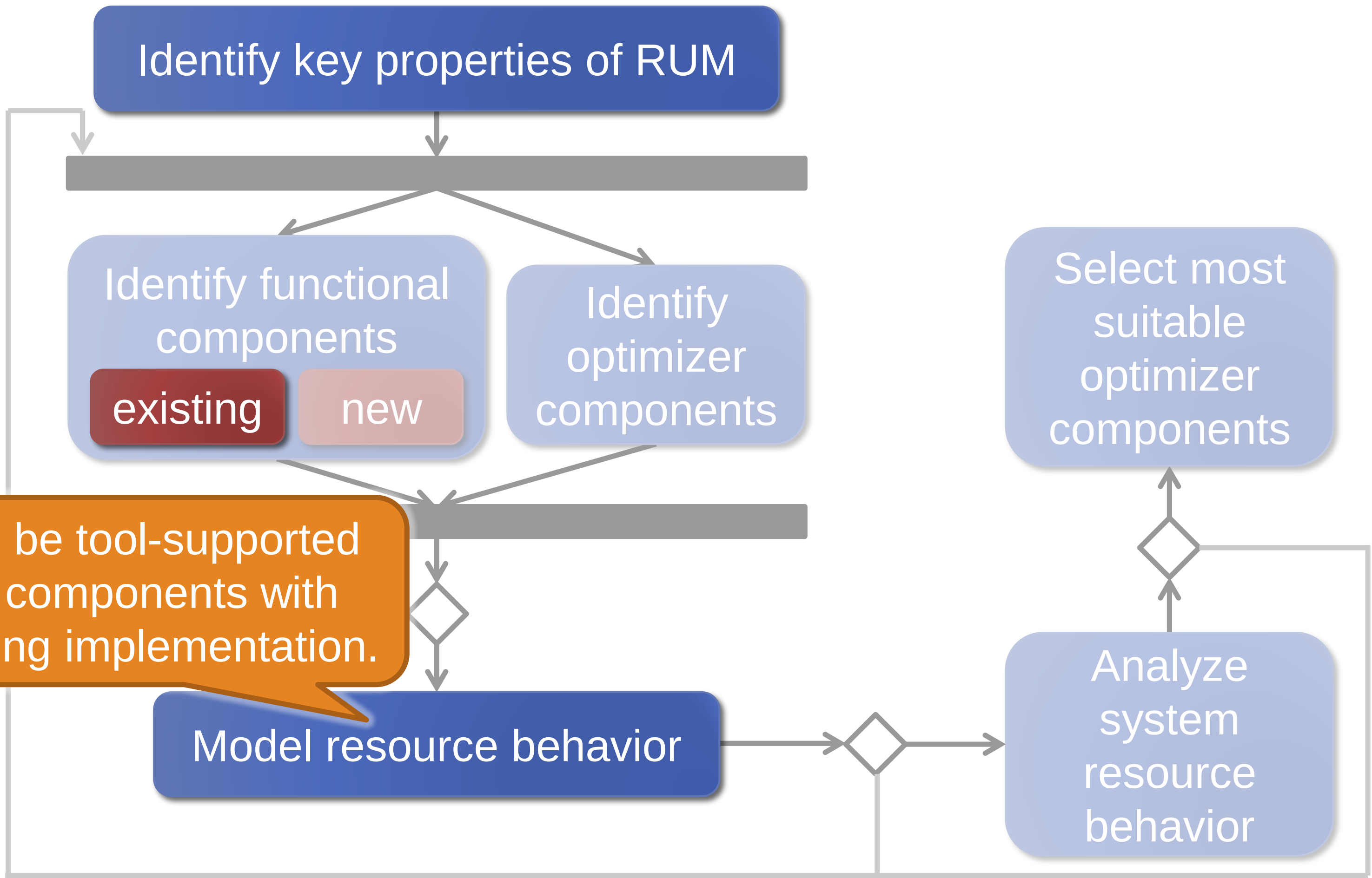
Overview

Method

Tooling

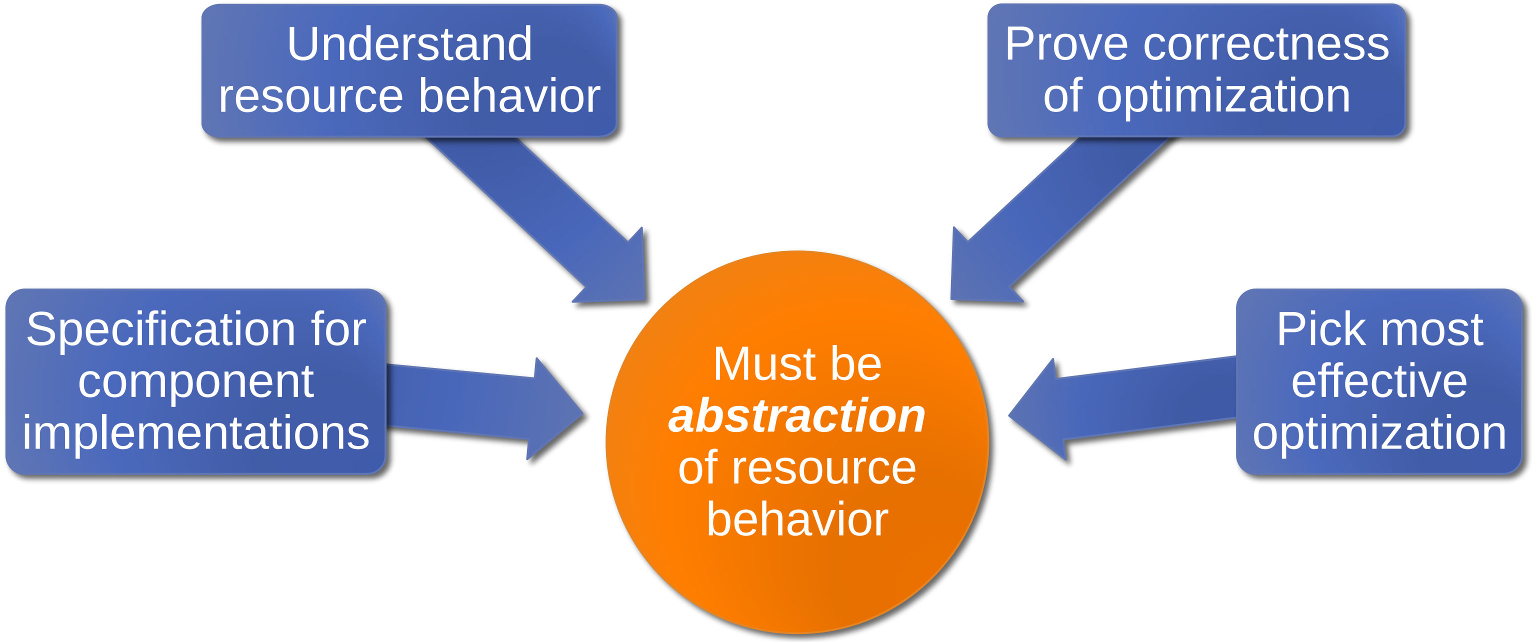
Conclusion

Design Method for Energy-Aware Embedded Software



Can be tool-supported for components with existing implementation.

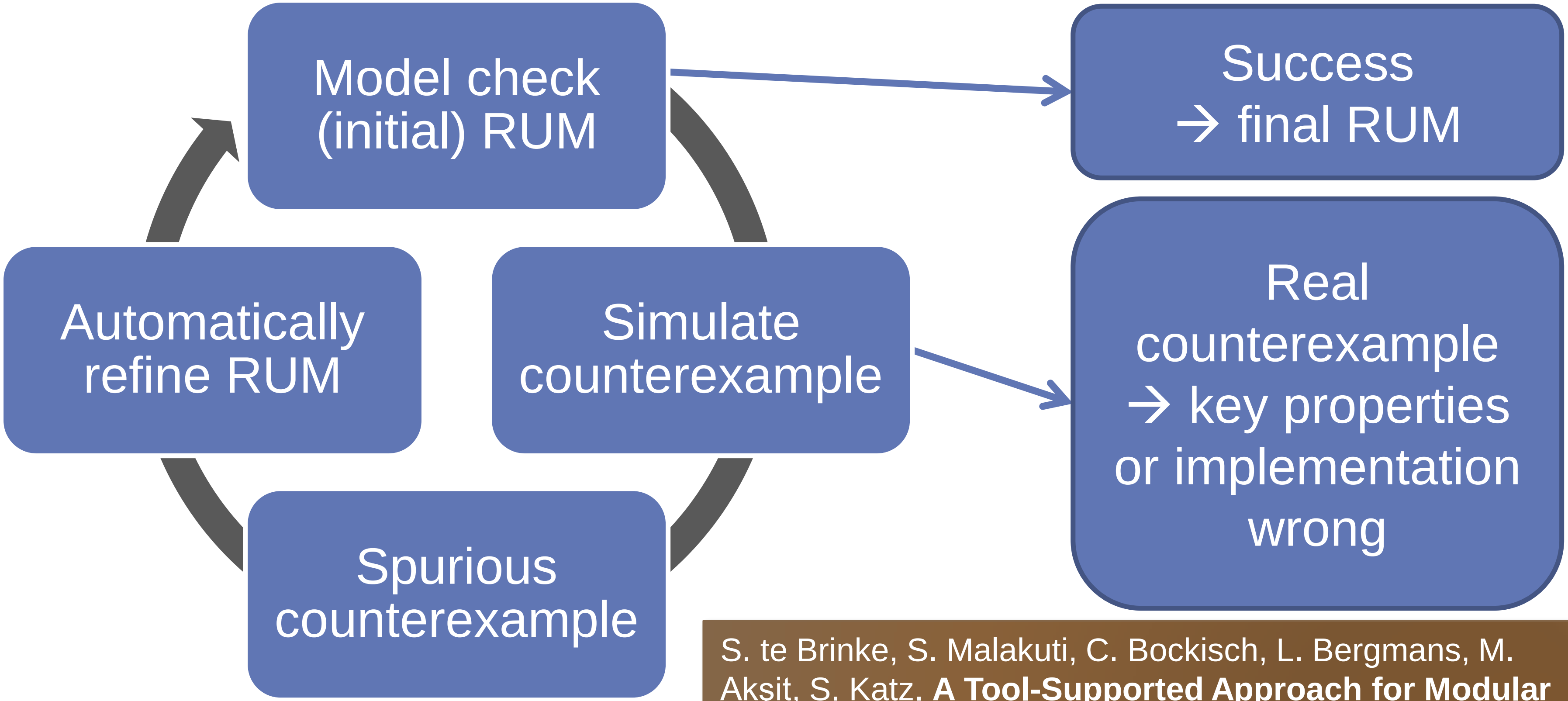
Purpose of RUM at design-time



- **Guarantee** liveness and safety properties for **all concretizations**
  → Over-approximation
- **Human-readable**
  → Abstraction must be minimal

A Formal Method for Extracting RUMs

- Counterexample-Guided Abstraction Refinement (CEGAR) [16]
- Can be applied to create RUMs for **existing components**



S. te Brinke, S. Malakuti, C. Bockisch, L. Bergmans, M. Akşit, S. Katz. **A Tool-Supported Approach for Modular Design of Energy-Aware Software.** In *SAC*, ACM, 2014

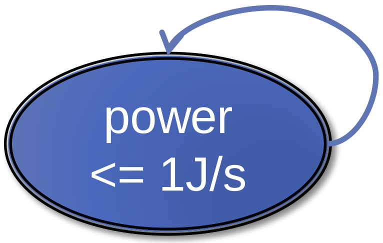
Overview

Method

Tooling

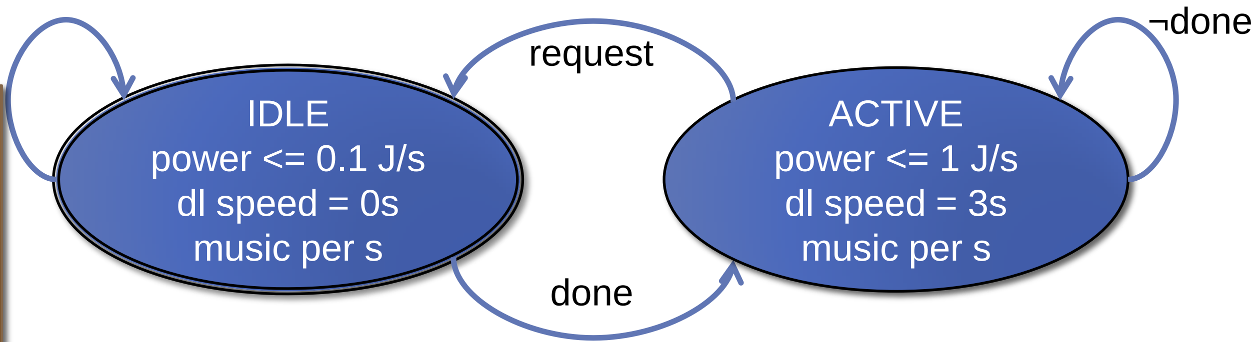
Conclusion

Extract RUM using CEGAR

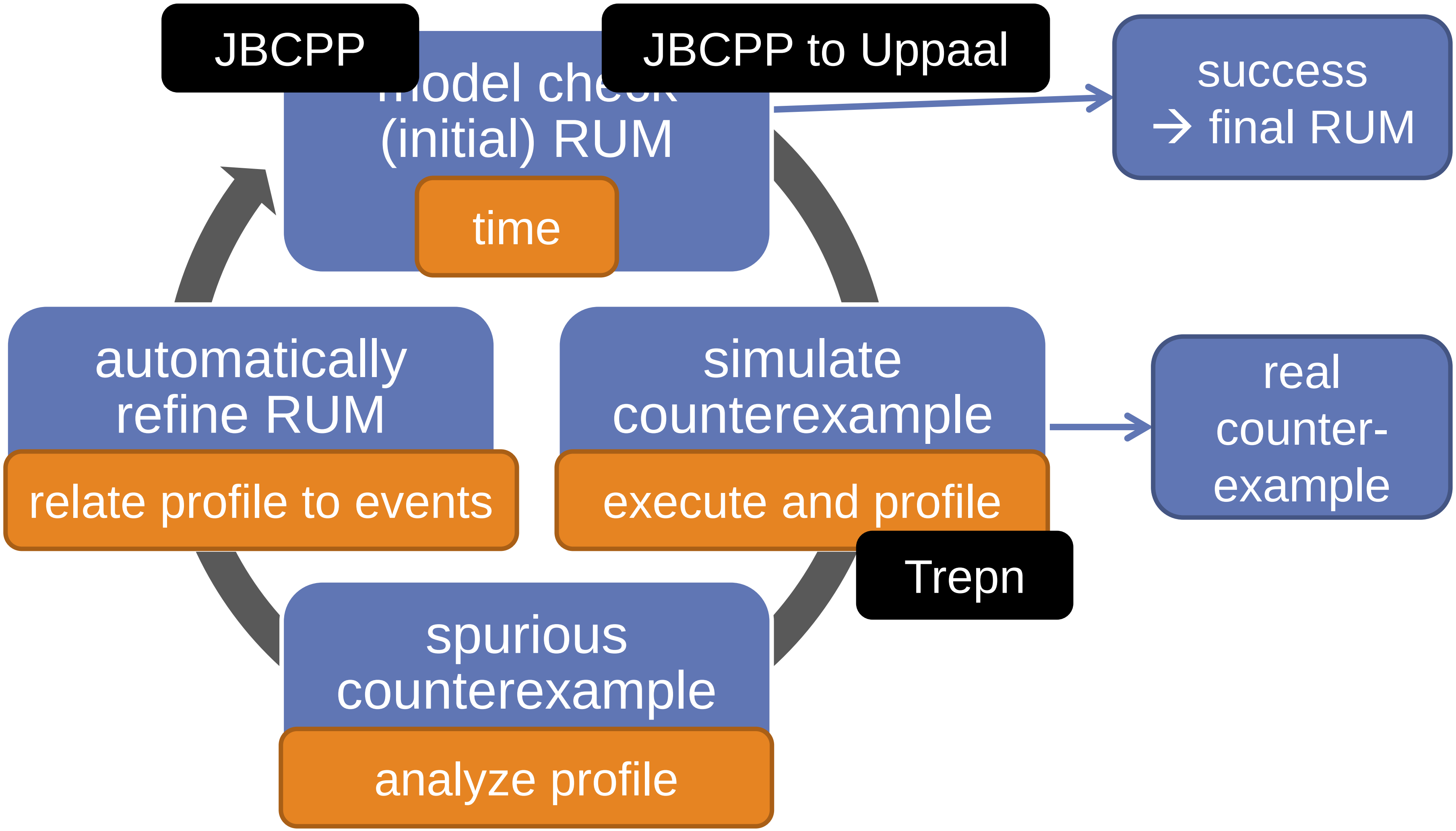


- Initial abstraction
  - Identify maximum power consumption
  - Specify one re-entrant state
  - With power consumption <= maximum
- Example key property: *in all execution sequences, the media player consumes less than 10 J for playing 20 s of music*
  - Counter example exists in abstract model
  - This counterexample does not exist in concrete model because of time-out and IDLE state

S., te Brinke, C. Bockisch, L., Bergmans, S. Malakuti, M. Aksit, S. Katz. **Deriving Minimal Models for Resource Utilization.** In: *GIBSE*, ACM, 2013

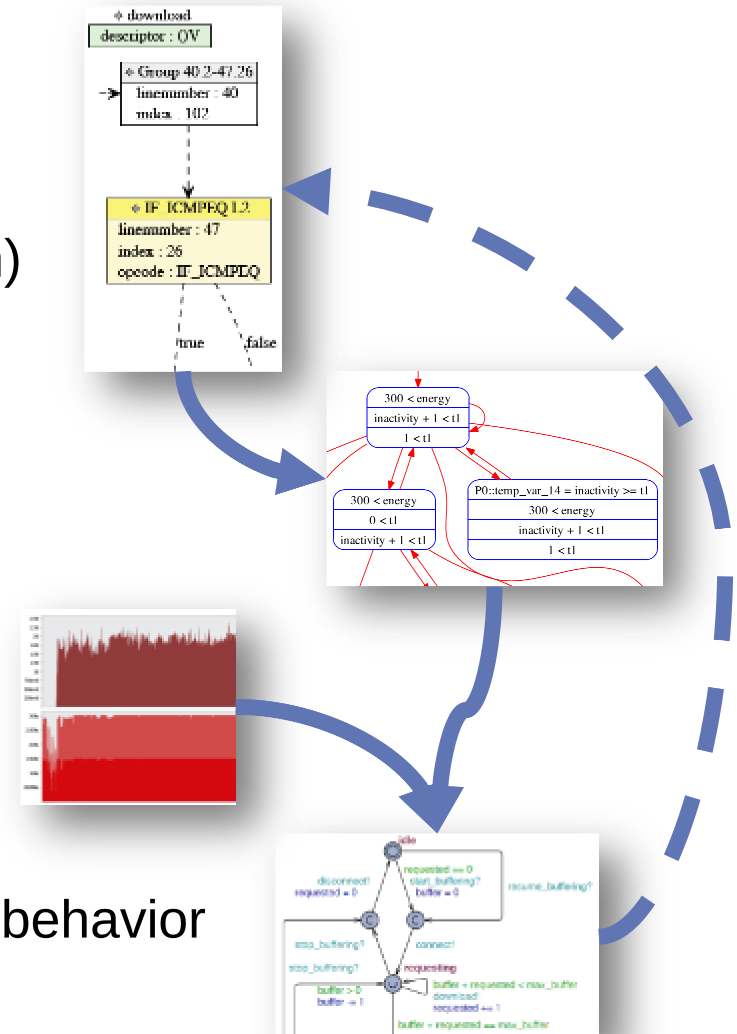


CEGAR for Extracting RUMs

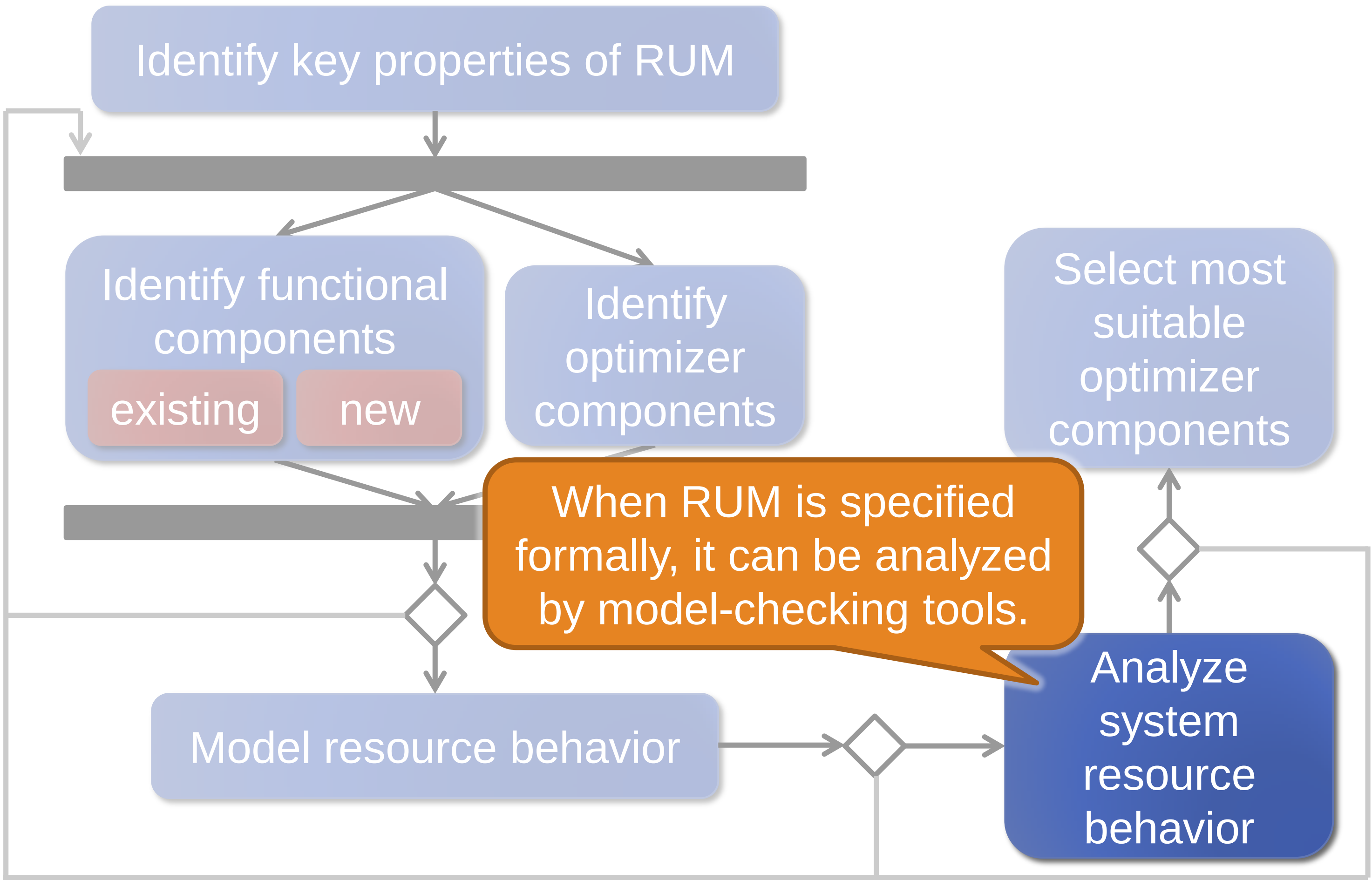


Tool support

- Developed JBCPP
  - Ecore-based model of Java bytecode
  - Extensible (e.g., energy/time information)
- Adapted MAGIC
  - CEGAR-implementation
  - Extract RUM from C source code
  - Optimize resulting RUM
- Adopted Trepn
  - Energy profiling Android applications
- Adopted UPPAAL
  - Compose and analyze system resource behavior
  - Simulate using model checking



Design Method for Energy-Aware Embedded Software

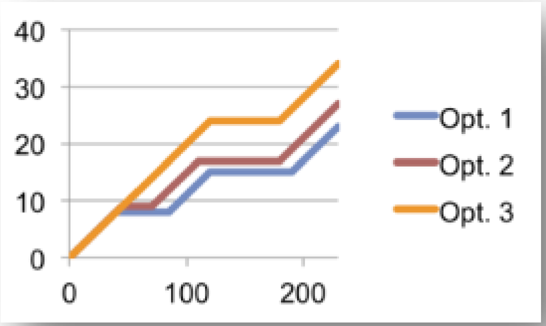


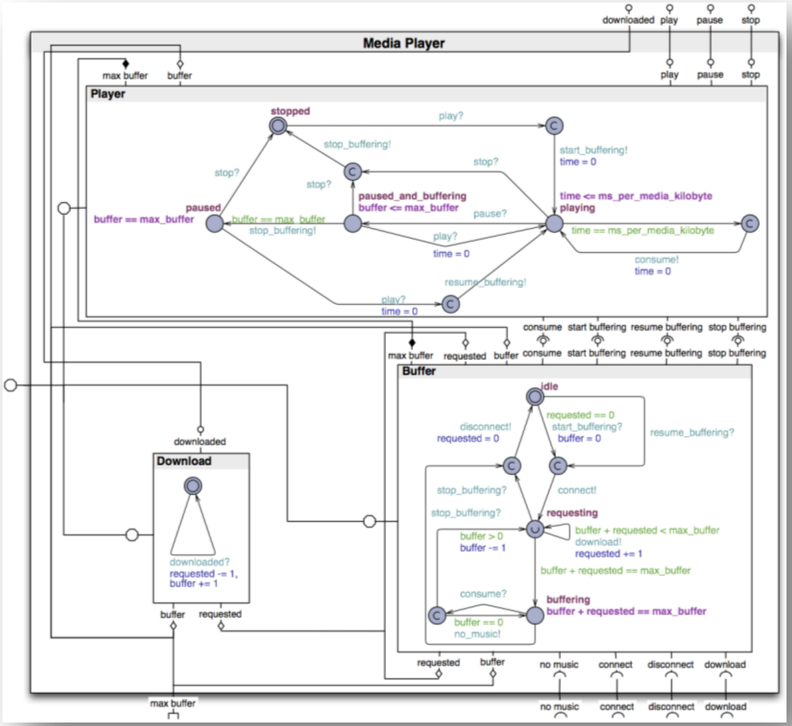
Analyzing System Resource Behavior with UPPAAL

- Commercially used model checker
- Model, verify, and validate **timed automata**
- Models are finite-state machines with numeric and clock variables (RUM)
  - **Transitions react to events** (invocation of provided service)
  - **Create events** (invoke required service)
  - **Variables** (can represent resource consumption)
- Key properties
  - Subset of **timed computation tree logic**

Analyzing System Resource Behavior with UPPAAL

- **Consistency checks**: only use specified services and resources
- **Liveness checks**:
- **Simulate** model to determine resource usage





- Cannot automatically choose the best composition

Overview

Method

Tooling

Conclusion

Summary

- **Iterative method** for developing energy-aware software
  - Software controlling energy-intensive hardware
  - Modular implementation of optimizations
  - Specify energy (resource) behavior at interface
- Tool for **extracting resource utilization model**
  - Based on formal method
  - Yields timed automaton
- **Analysis** of system's resource utilization
- Not shown here:
  Programming language support for **automatic, online tracking** of resource state

Future Work

- Improve energy profiling
  - Software Energy Footprint lab:
    Dedicated hardware measuring energy consumption
  - High accuracy

- Use analysis result to improve profiling automatically

- Time Performance Improvement with Parallel Processing Systems
  - Use model checker simulate system with soft real-time constraints
  - Identify bottlenecks and propose optimizations

Next Research Idea

Optimize energy consumption of execution itself

- Create extensive profile:
  - Energy consumption
  - Non-deterministic behavior, such as:
    thread-switching, optimization decisions, garbage collection
- Discover dependencies with data mining
- Derive heuristics for non-deterministic decisions
- Possibly develop online optimizations