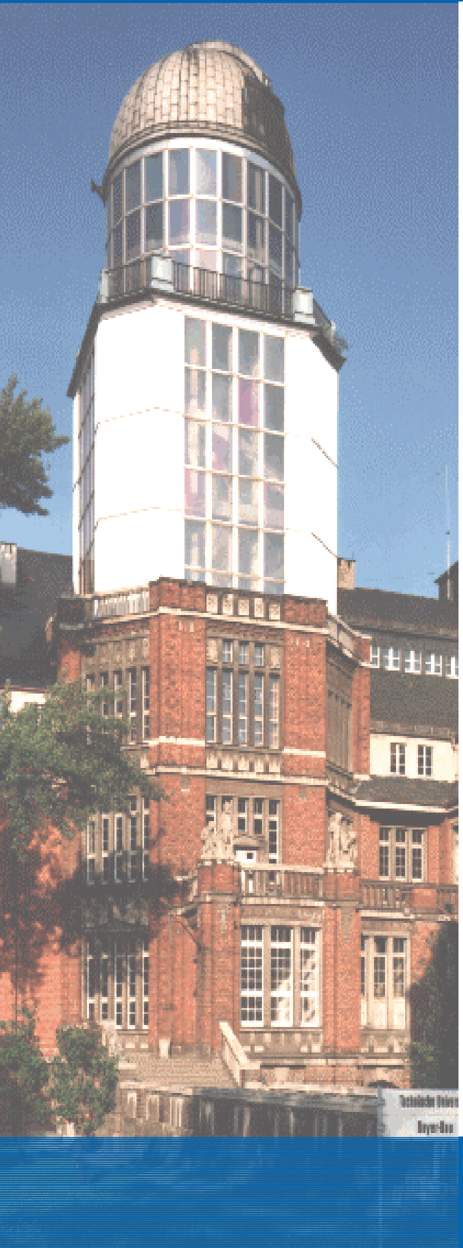
TECHNISCHE
UNIVERSITÄT
DRESDEN



Formal Specification of Non-functional Properties of Component-Based Software

Steffen Zschaler
Dresden University of Technology

Workshop on Models for Non-functional Aspects of Component-Based Systems (NfC'04)
October 11, 2004, Lisbon, Portugal

Motivation

Two trends underlying this work:

- CBSE
  - Current software systems have a high complexity
  - Modularity and component-based techniques can reduce complexity
- Non-functional properties
  - have been studied in the small, (Performance Engineering, …)
  - How to scale up?

➔ **Component-based technologies can be a key factor for scaling up non-functional specifications.**

- **Component-based systems open new ways to achieve non-functional properties**
  - Component-level scheduling
  - Buffers, Migration, Replication…

Outline

- General Principles
- The Specification
- Outlook/Conclusions

An Example: Timely Response From a System

- Depends on:
  - – The way the code is written (algorithmic issues)

internal properties of the component

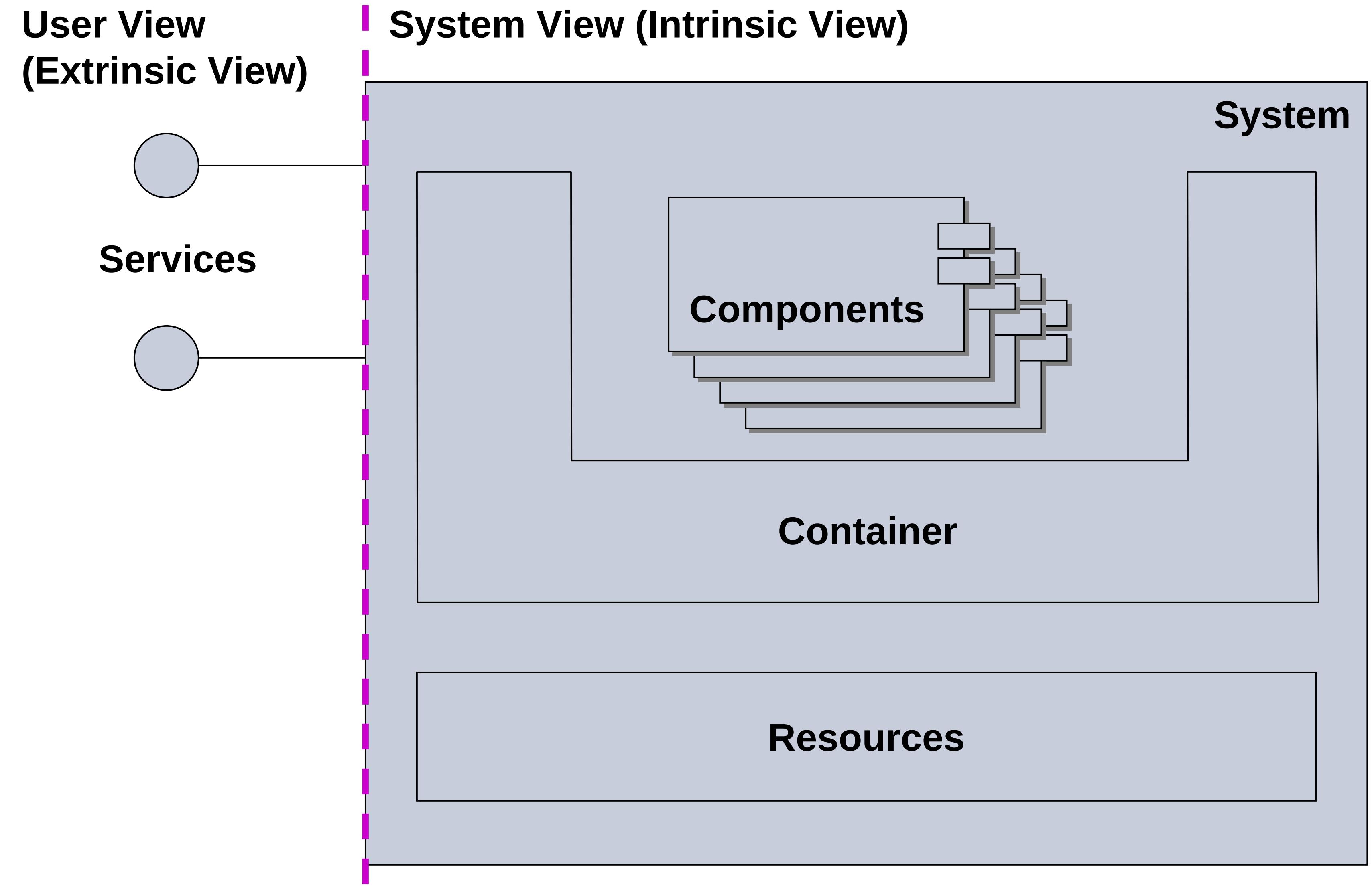
  - – The time used other components take to do their part of the work
  - – Buffering of requests between components
  - – CPU scheduling and timely availability of other resources

external properties depending on component usage

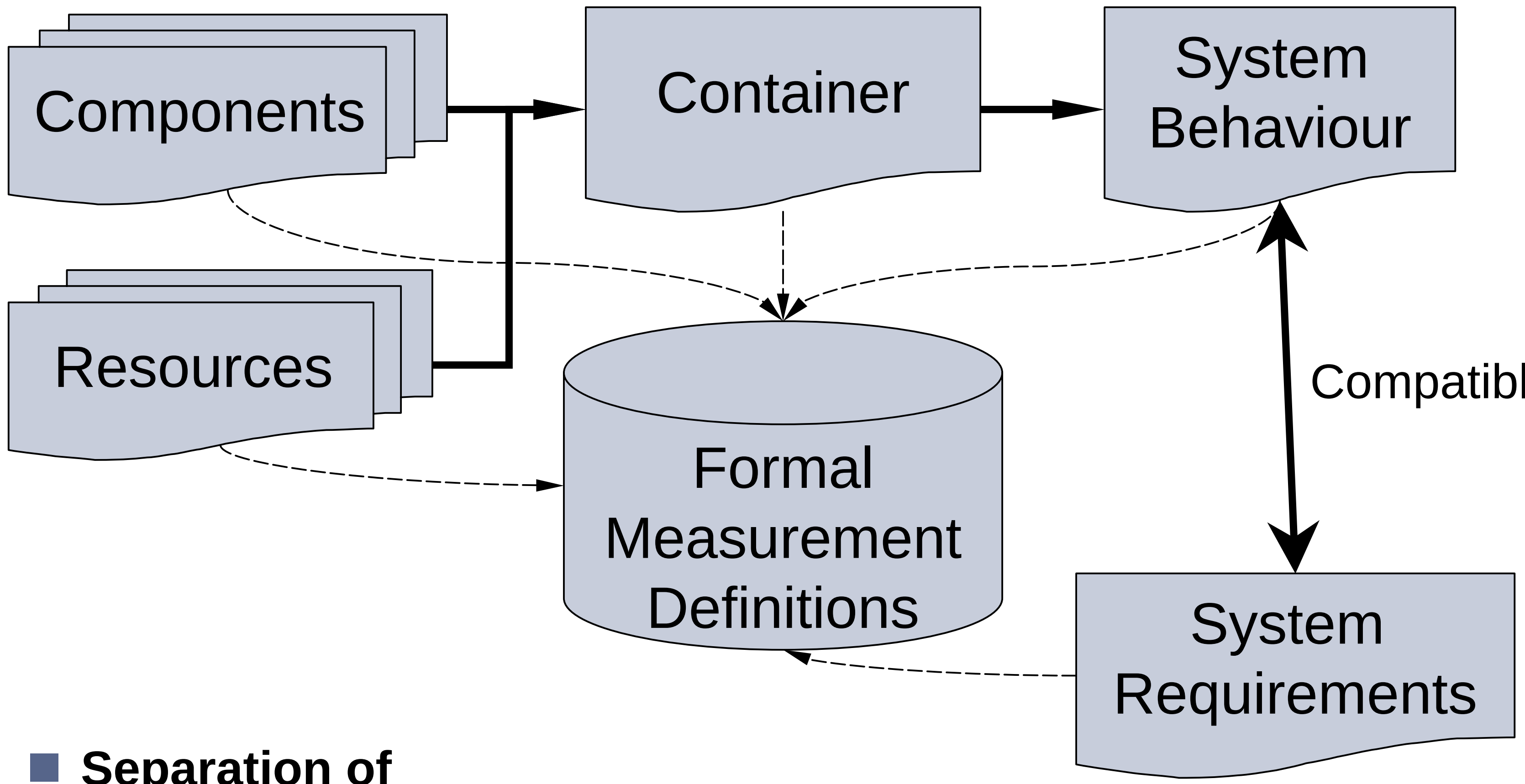
➔ **We can talk about:**

- – Execution time = an *intrinsic property* of a *component*
- – Response time = an *extrinsic property* of a *system*

Global System Model



A Specification Framework



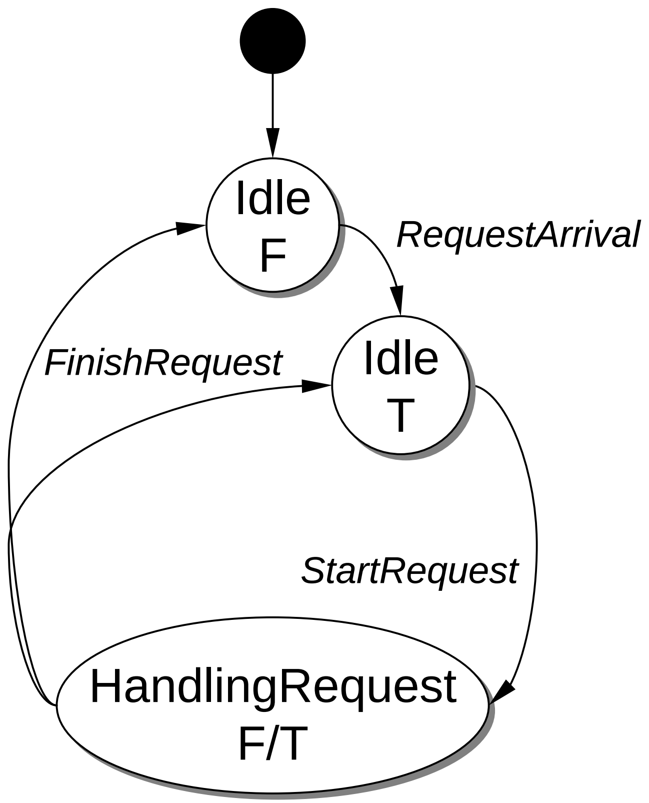
- **Separation of**
  - *Context models:* Models of the “system mechanics”
  - *Measurement specification:* Definition of actual non-functional aspects

Service – Context Model

```
─────────────────────── MODULE Service ───────────────────────
VARIABLE inState
VARIABLE unhandledRequest

InitEnv ≜ unhandledRequest = FALSE

RequestArrival ≜    unhandledRequest′ = TRUE
                 ∧  UNCHANGED inState

NextEnv ≜ RequestArrival

EnvSpec ≜    InitEnv
          ∧  □[NextEnv]_unhandledRequest
──────────────────────────────────────────────────────────────
InitServ ≜ inState = Idle

StartRequest ≜    inState = Idle
               ∧  unhandledRequest = TRUE
               ∧  inState′ = HandlingRequest
               ∧  unhandledRequest′ = FALSE

FinishRequest ≜    inState = HandlingRequest
                ∧  inState′ = Idle
                ∧  UNCHANGED unhandledRequest

NextServ ≜ StartRequest ∨ FinishRequest

vars ≜ ⟨inState, unhandledRequest⟩

ServiceSpec ≜    InitServ
              ∧  □[NextServ]_vars
──────────────────────────────────────────────────────────────
Service ≜ EnvSpec -+-▷ ServiceSpec
──────────────────────────────────────────────────────────────
```

- Currently very simple model:

Service = Single Oeration



Service – Measurement Specification

MODULE *ResponseTimeConstrainedService*

EXTENDS *RealTime*

CONSTANT *ResponseTimeBound*
ASSUME $(ResponseTimeBound \in Real) \land (ResponseTimeBound > 0)$

VARIABLES *ResponseTime*, *inState*, *unhandledRequest*, *Start*

$Serv \triangleq$ INSTANCE *Service*

$Init \triangleq Start = 0 \land ResponseTime = 0$

$StartNext \triangleq Serv!StartRequest \Rightarrow Start' = now$

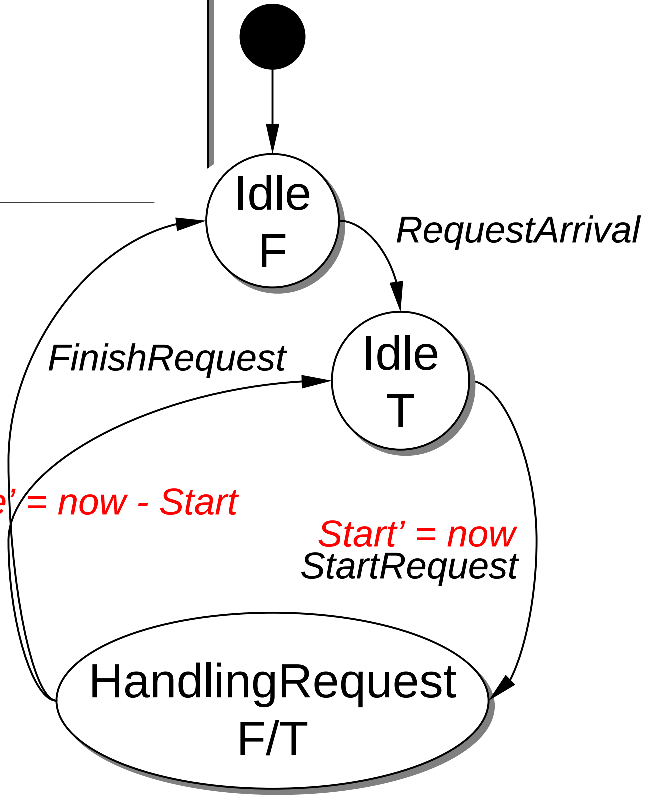
$RespNext \triangleq Serv!FinishRequest \Rightarrow ResponseTime' = now - Start$

$Next \triangleq StartNext \land RespNext$

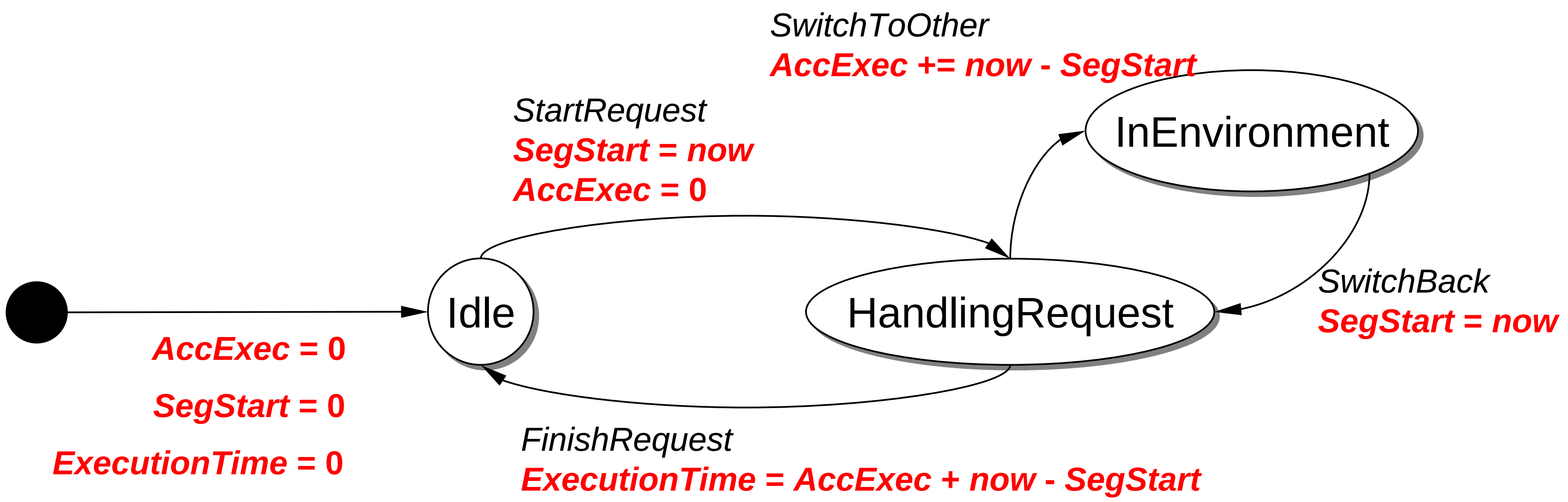
$vars \triangleq \langle inState, unhandledRequest, Start, ResponseTime \rangle$

$RespSpec \triangleq Init \land \Box[Next]_{vars}$

$$\begin{aligned} Service \triangleq\ & Serv!Service \\ \land\ & RTnow(vars) \\ \land\ & RespSpec \\ \land\ & \Box(ResponseTime \le ResponseTimeBound) \end{aligned}$$



Component


SwitchToOther
AccExec += now - SegStart
StartRequest
SegStart = now
AccExec = 0
InEnvironment
Idle
HandlingRequest
SwitchBack
SegStart = now
AccExec = 0
SegStart = 0
ExecutionTime = 0
FinishRequest
ExecutionTime = AccExec + now - SegStart

Resource

- Example: CPU scheduled by RMS (Rate-Monotonic Scheduling)

MODULE *RMSScheduler*

EXTENDS *Reals*

CONSTANT *TaskCount*
ASSUME $(TaskCount \in Nat) \land (TaskCount > 0)$

CONSTANT *Periods*
ASSUME $Periods \in [\{1 \,..\, TaskCount\} \rightarrow Real]$

CONSTANT *Wcets*
ASSUME $Wcets \in [\{1 \,..\, TaskCount\} \rightarrow Real]$

VARIABLES *MinExecTime*, *AssignedTo*, *now*

$TimedCPUSched \triangleq$ INSTANCE *TimedCPUScheduler*

$$Schedulable \triangleq \sum_{k=1}^{TaskCount} \frac{Wcets[k]}{Periods[k]} \leq TaskCount\left(\sqrt[TaskCount]{2} - 1\right)$$

$$RMSScheduler \triangleq \quad TimedCPUSched!TimedCPUScheduler$$
$$\land\ Schedulable \Rightarrow \Box\, TimedCPUSched!ExecutionTimesOk$$

Container

$$
\begin{aligned}
\mathit{ContainerPreCond} \triangleq\ & \mathit{ExecutionTime} \leq \mathit{ResponseTime} \\
\land\ & (\mathit{TaskCount} = 1 \land \\
& \mathit{Periods} = [\mathbf{1} \rightarrow \mathit{ResponseTime}] \land \\
& \mathit{Wcets} = [1 \rightarrow \mathit{ExecutionTime}] \land \\
& \mathit{CPUCanSchedule}(\mathit{TaskCount}, \mathit{Periods}, \mathit{Wcets})) \\
\land\ & \mathit{ComponentMaxExecTime}(\mathit{ExecutionTime}) \\
\land\ & \mathit{MinInterrequestTime}(\mathit{ResponseTime})
\end{aligned}
$$

$$\mathit{ContainerPostCond} \triangleq \mathit{ServiceResponseTime}(\mathit{ResponseTime})$$

$$\mathit{Container} \triangleq \mathit{ContainerPreCond} \Rightarrow \mathit{ContainerPostCond}$$

- So far no selection of
  - Concrete component(s)
  - Concrete resource realizations
    - We selected that we need CPU, but didn’t say anything about RMS

A Sample System Specification

VARIABLES $TaskCount, Periods, Wcets$

$$
\begin{aligned}
System \triangleq\ & MyComponent(20) \\
\land\ & MyCPU(TaskCount, Periods, Wcets) \\
\land\ & MyContainer(20, ResponseTime, TaskCount, Periods, Wcets)
\end{aligned}
$$

$$ExternalService \triangleq Environment(RequestPeriod) \overset{+}{\Rightarrow} Service(ResponseTime)$$

$$IsFeasible \triangleq System \Rightarrow ExternalService$$

Outlook

- Future work:
  - Mapping context model ↔ application model

  - Extend to services delivered by networks of components
  - Extend to multiple properties per specification

  - Apply to other examples
    - other service models (stream based services)
    - stochastic extrinsic properties

Conclusion

- Distinction of intrinsic/extrinsic properties of components/services

- System specification = Composition of component, service, resource and container specifications
  → Scalability of the specifications through clear modularization
  → Formal measurement definitions as interface between specs.

- Feasible System = available components and resources allow the container to provide the required non-functional properties

Related Work

- Non-functional specifications:
  - Specifically semantics:

    Staehli, R., Eliassen, F., Aagedal, J.Ø., Blair, G.: *Quality of service semantics for component-based systems*. In: Middleware 2003 Companion, 2nd Int'l Workshop on Reflective and Adaptive Middleware Systems.

  - Specification approaches:
    - Characteristic-specific

      … lots
    - Measurement-based

      Aagedal: *Quality of service support in development of distributed systems* → CQML

      Selic: *A generic framework for modelling resources with UML*

      Skene et al: *Precise Service-Level Agreements*