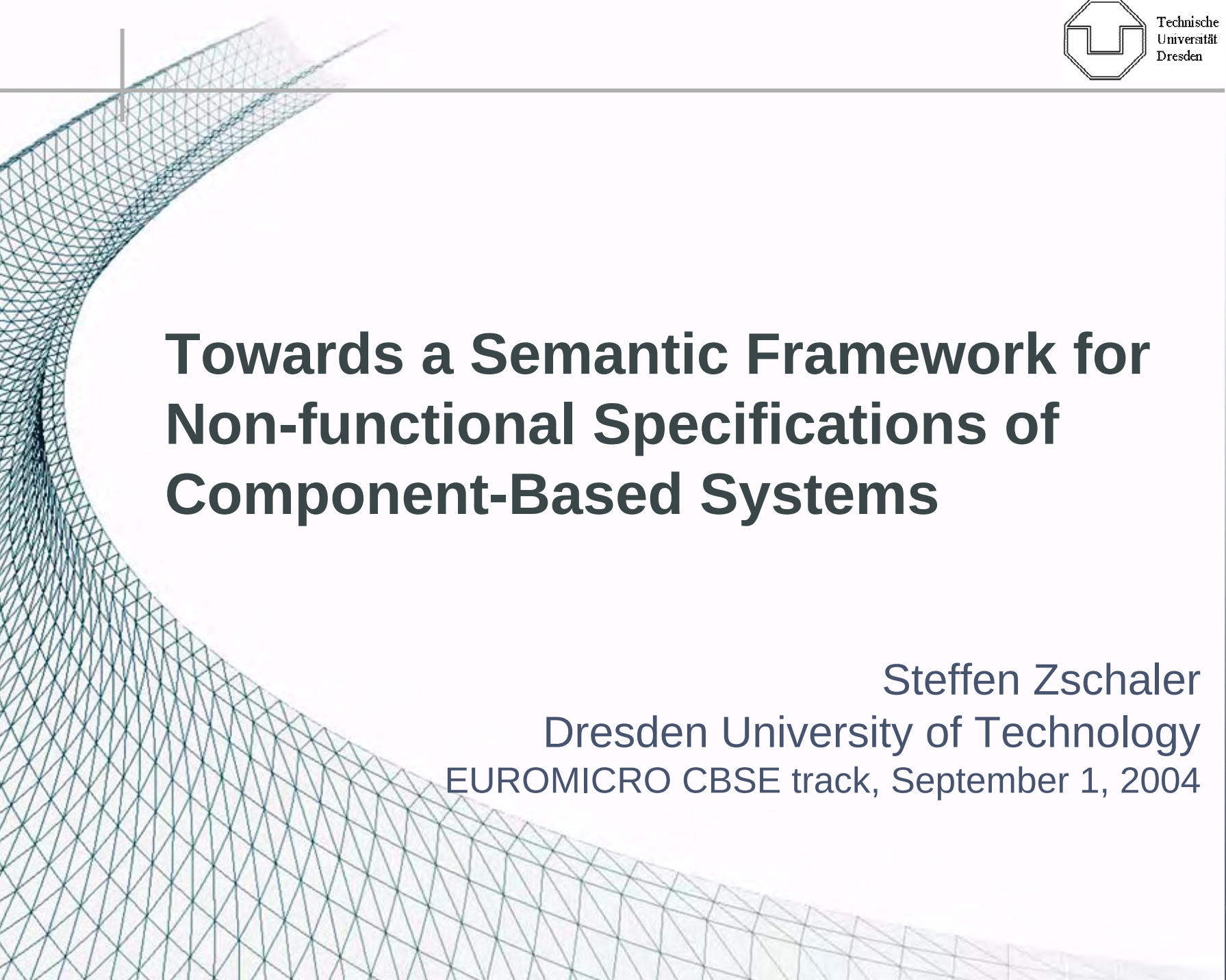


Towards a Semantic Framework for Non-functional Specifications of Component-Based Systems

Steffen Zschaler
Dresden University of Technology
EUROMICRO CBSE track, September 1, 2004

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
- An Example 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

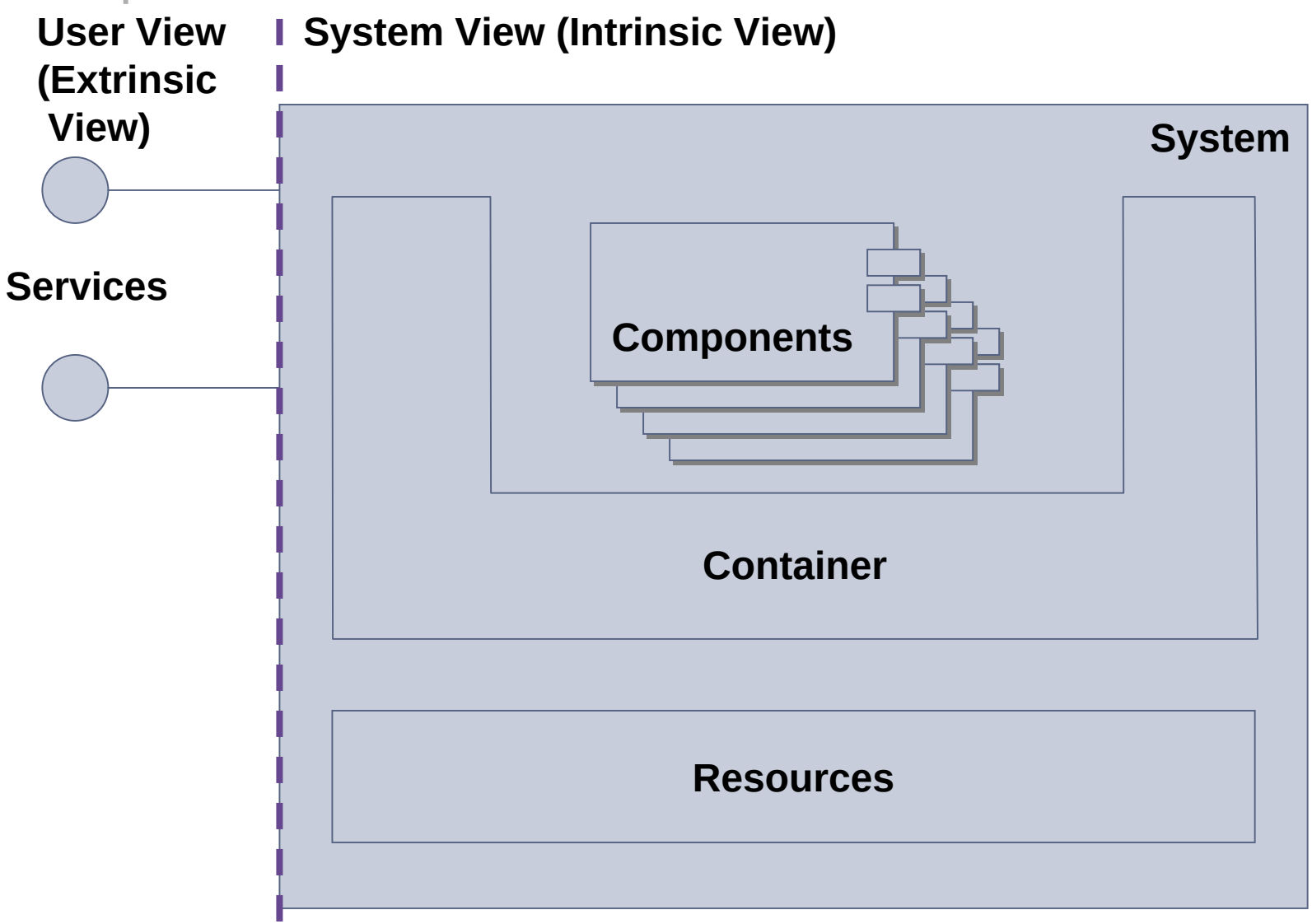
  - The time used other components take to do their part of the work
  - Buffering of requests between components
  - CPU scheduling and timely availability of other resources

external properties depending on component usage

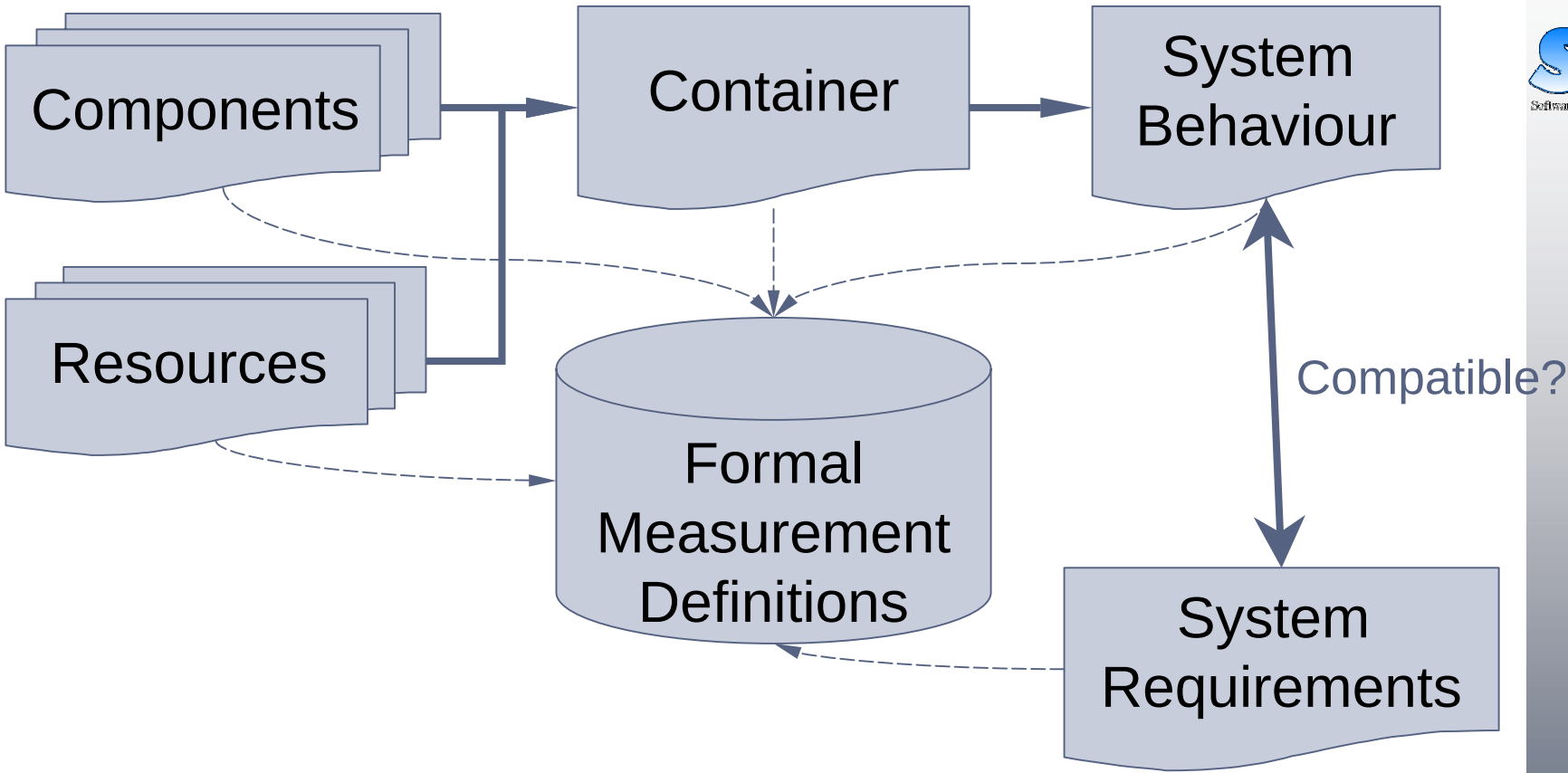
➔ **We can talk about:**

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

Global System Model



The 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*

$$InitEnv \triangleq unhandledRequest = \text{FALSE}$$

$$RequestArrival \triangleq unhandledRequest' = \text{TRUE} \wedge \text{UNCHANGED } inState$$

$$NextEnv \triangleq RequestArrival$$

$$EnvSpec \triangleq InitEnv \wedge \Box[NextEnv]_{unhandledRequest}$$

$$InitServ \triangleq inState = Idle$$

$$StartRequest \triangleq inState = Idle \wedge unhandledRequest = \text{TRUE} \wedge inState' = HandlingRequest \wedge unhandledRequest' = \text{FALSE}$$

$$FinishRequest \triangleq inState = HandlingRequest \wedge inState' = Idle \wedge \text{UNCHANGED } unhandledRequest$$

$$NextServ \triangleq StartRequest \vee FinishRequest$$

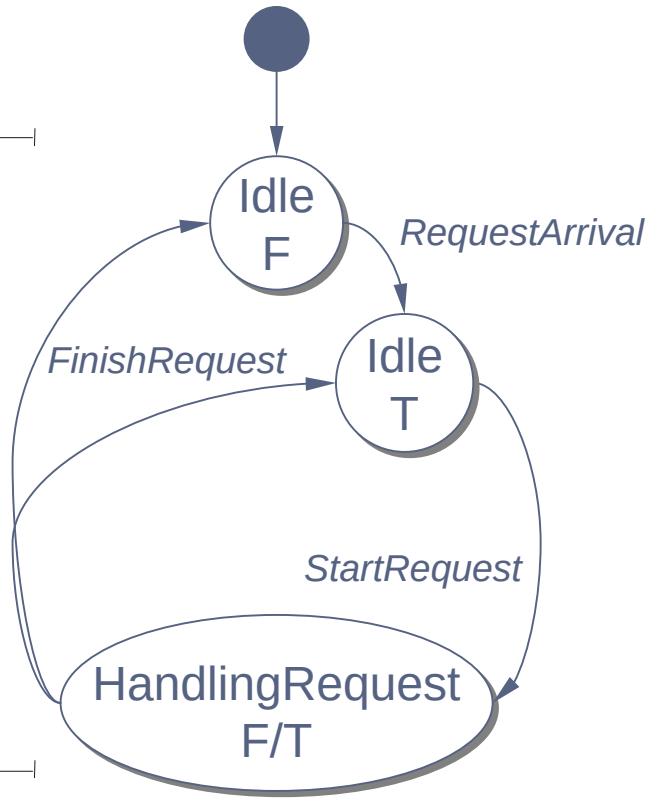
$$vars \triangleq \langle inState, unhandledRequest \rangle$$

$$ServiceSpec \triangleq InitServ \wedge \Box[NextServ]_{vars}$$

$$Service \triangleq EnvSpec \stackrel{+}{\Rightarrow} ServiceSpec$$

- **Currently very simple model:**

Service = Single Operation



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*

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

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

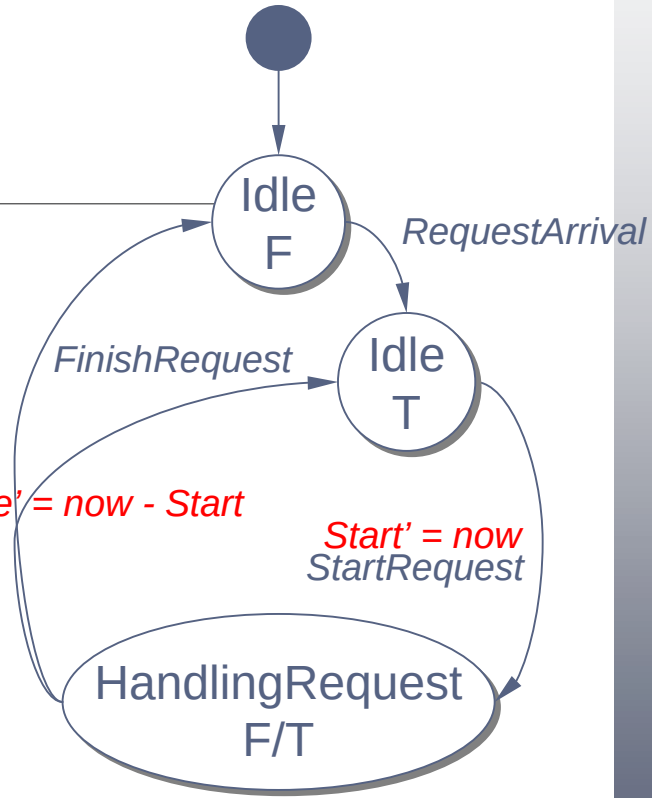
$RespNext \triangleq Serv!FinishRequest \Rightarrow ResponseTime' = now - Start$

$Next \triangleq StartNext \land RespNext$

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

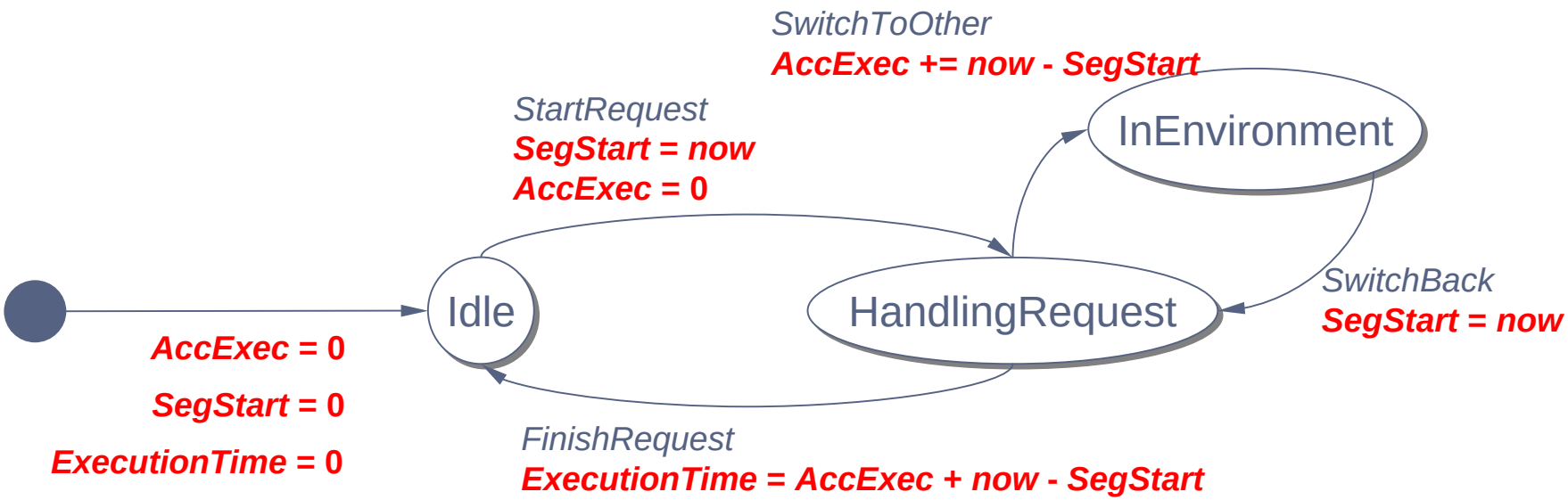
$RespSpec \triangleq Init \land \Box[Next]_{vars}$

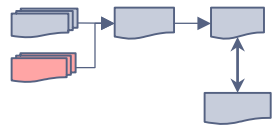
$$Service \triangleq \begin{array}{ll} & Serv!Service \\ \land & RTnow(vars) \\ \land & RespSpec \\ \land & \Box(ResponseTime \leq ResponseTimeBound) \end{array}$$



Definition of response time measurement

Component





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

MODULE *RMSScheduler*

EXTENDS *Reals*

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

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

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

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

$TimedCPUSched \triangleq$ INSTANCE *TimedCPUScheduler*

$Schedulable \triangleq$ LET $usage \triangleq [k \in \{1 \,..\, TaskCount\} \mapsto (Wcets[k]/Periods[k])]$
IN
$Sum(usage) \leq (TaskCount * (sqrt(TaskCount, 2) - 1))$

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

Container

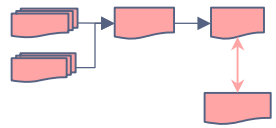
$$
\begin{aligned}
ContainerPreCond \triangleq\ & ExecutionTime \leq ResponseTime \\
\wedge\ & (TaskCount = 1 \wedge \\
& \ Periods = [1 \rightarrow ResponseTime] \wedge \\
& \ Wcets = [1 \rightarrow ExecutionTime] \wedge \\
& \ CPUCanSchedule(TaskCount,\ Periods,\ Wcets)) \\
\wedge\ & ComponentMaxExecTime(ExecutionTime)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

$$
ContainerPostCond \triangleq ServiceResponseTime(ResponseTime)
$$

$$
Container \triangleq ContainerPreCond \Rightarrow ContainerPostCond
$$

- **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



$$\textbf{VARIABLES}\ \mathit{TaskCount},\ \mathit{Periods},\ \mathit{Wcets}$$

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

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

$$\mathit{IsFeasible} \triangleq \mathit{System} \Rightarrow \mathit{ExternalService}$$

Outlook

- **Future work:**
  - Extend to services delivered by networks of components
  - Extend to multiple properties per specification
  - Mapping context model ↔ application model
  - 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*