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)
  - 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

- **We can talk about:**
  - Execution time = an *intrinsic property of a component*
  - Response time = an *extrinsic property of a system*
Global System Model

User View (Extrinsic View)

System View (Intrinsic View)

System

Components

Container

Resources

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

**Currently very simple model:**

Service = Single Operation

```
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
```
Service – Measurement Specification

EXTENDS RealTime

CONSTANT ResponseTimeBound
ASSUME (ResponseTimeBound ∈ Real) ∧ (ResponseTimeBound > 0)

VARIABLES ResponseTime, inState, unhandledRequest, Start

Serv ⋁ INSTANCE Service

Init ⋁ Start = 0 ∧ ResponseTime = 0

StartNext ⋁ Serv!StartRequest ⇒ Start' = now

RespNext ⋁ Serv!FinishRequest ⇒ ResponseTime' = now − Start

Next ⋁ StartNext ∧ RespNext

vars ⋁ (inState, unhandledRequest, Start, ResponseTime)

RespSpec ⋁ Init ∧ □[Next]vars

Service ⋁ Serv!Service
∧ RTnow(vars)
∧ RespSpec
∧ □(ResponseTime ≤ ResponseTimeBound)

Definition of response time measurement

© Steffen Zschaler, 2004
Component

AccExec = 0
SegStart = 0
ExecutionTime = 0

Idle

StartRequest
SegStart = now
AccExec = 0

HandlingRequest

InEnvironment

SwitchToOther
AccExec += now - SegStart

FinishRequest
ExecutionTime = AccExec + now - SegStart

SwitchBack
SegStart = now
Resource

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

```
MODULE RMSScheduler

EXTENDS Reals

CONSTANT TaskCount
ASSUME (TaskCount ∈ Nat) ∧ (TaskCount > 0)

CONSTANT Periods
ASSUME Periods ∈ [{1 .. TaskCount} → Real]

CONSTANT Wcets
ASSUME Wcets ∈ [{1 .. TaskCount} → Real]

VARIABLES MinExecTime, AssignedTo, now

TimedCPUSched ≜ INSTANCE TimedCPUScheduler

Schedulable ≜ LET usage ≜ [k ∈ {1 .. TaskCount} ↦ (Wcets[k]/Periods[k])] IN
Sum(usage) ≤ (TaskCount * (sqrt(TaskCount, 2) − 1))

RMS Scheduler ≜ TimedCPUSched! TimedCPUScheduler
∧ Schedulable ⇒ □ TimedCPUSched! ExecutionTimesOk
```
Container

\[ \text{ContainerPreCond} \triangleq \begin{align*}
\text{ExecutionTime} & \leq \text{ResponseTime} \\
\land (\text{TaskCount} = 1 \land \\
\text{Periods} = [1 \rightarrow \text{ResponseTime}] \land \\
\text{Wcets} = [1 \rightarrow \text{ExecutionTime}] \land \\
\text{CPUCanSchedule}(\text{TaskCount}, \text{Periods}, \text{Wcets}) \land \\
\text{ComponentMaxExecTime}(\text{ExecutionTime}) \land \\
\text{MinInterrequestTime}(\text{ResponseTime})
\end{align*} \]

\[ \text{ContainerPostCond} \triangleq \text{ServiceResponseTime}(\text{ResponseTime}) \]

\[ \text{Container} \triangleq \text{ContainerPreCond} \Rightarrow \text{ContainerPostCond} \]

- **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

System \triangleq MyComponent(20)
\land MyCPU(TaskCount, Periods, Wcets)
\land MyContainer(20, ResponseTime, TaskCount, Periods, Wcets)

ExternalService \triangleq Environment(RequestPeriod) \Rightarrow Service(ResponseTime)

IsFeasible \triangleq System \Rightarrow ExternalService
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:
  

- 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*