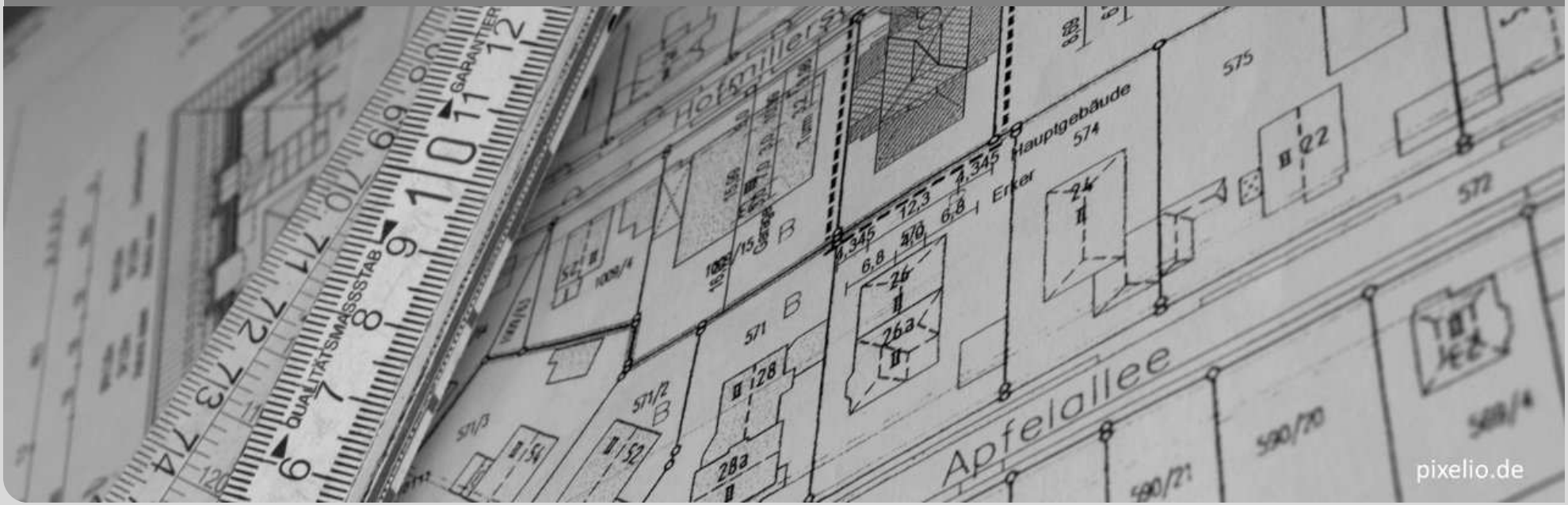
Probe Specification

SDQ Lerngruppe 2010

Philipp Merkle, Anne Martens | 04.02.10

CHAIR FOR SOFTWARE DESIGN AND QUALITY



Outline

Outline

Motivation – General

- Specify measuring points / track by a model
- Simplify measurings in simulations, benchmarks, ...
- Integrates persistence framework(s) – soon

Motivation – SimuCom

- Current situation
  - Where and what to measure hard coded
  - Persists measurements using SensorFramework
  - Some things cannot be measured
    - e.g. duration between two calls
  - What if some measurements are not needed?
- SimuCom with ProbeSpecification
  - Where and what defined by a model
  - Persistence: EDP2 applicable
  - Allows deactivation of certain measurements

Motivation – SimuCom

- Current situation
  - Where and what to measure hard coded
  - Persists measurements using SensorFramework
  - Some things cannot be measured
    - e.g. duration between two calls
  - What if some measurements are not needed?
- SimuCom with ProbeSpecification
  - Where and what defined by a model
  - Persistence: EDP2 applicable
  - Allows deactivation of certain measurements

Outline

Overview

- Probe Specification
  - Meta-Model
    - Model specifies the measuring track
  - Framework
    - Supports concrete measurings
- Pipes-and-Filters
  - Filtering, aggregation, persistency of measurements
  - Meta-Model
    - Model specifies a filter chain
  - Framework
    - Provides mechanism to build and use filter chains

Overview

- Probe Specification
  - Meta-Model
    - Model specifies the measuring track
  - Framework
    - Supports concrete measurings
- Pipes-and-Filters
  - Filtering, aggregation, persistency of measurements
  - Meta-Model
    - Model specifies a filter chain
  - Framework
    - Provides mechanism to build and use filter chains

Probe Specification

- Meta-Model (EMF ECore)
  - Models define where and what to measure
  - Refers to another meta-model (e.g. PCM) containing “measurable” elements
    - Measurable (PCM): UsageScenario, ExternalCallAction, EntryLevelSystemCall, ...
    - Condition: ECore based too
- Framework
  - Simplifies collecting and calculating measurements
  - Should support various measuring environments
    - SimuCom, ProtoCom, . . .
  - Depends on Pipes-and-Filters
    - Delegates results to persistence framework

Probe Specification

- Meta-Model (EMF ECore)
  - Models define where and what to measure
  - Refers to another meta-model (e.g. PCM) containing “measurable” elements
    - Measurable (PCM): UsageScenario, ExternalCallAction, EntryLevelSystemCall, ...
    - Condition: ECore based too
- Framework
  - Simplifies collecting and calculating measurements
  - Should support various measuring environments
    - SimuCom, ProtoCom, . . .
  - Depends on Pipes-and-Filters
    - Delegates results to persistence framework

Pipes-and-Filters

- Meta-Model (EMF ECore)
  - Models define concrete filter chains
- Framework
  - Provides *PipesAndFiltersManager*
    - Simplifies building filter chains
    - Manages data flow source $\longrightarrow$ sink
  - Works with arbitrary persistence framework
    - SensorFramework, EDP2, ...
    - Implement custom *WriteStrategy*
  - Independent of Probe Specification $\Rightarrow$ reusable!

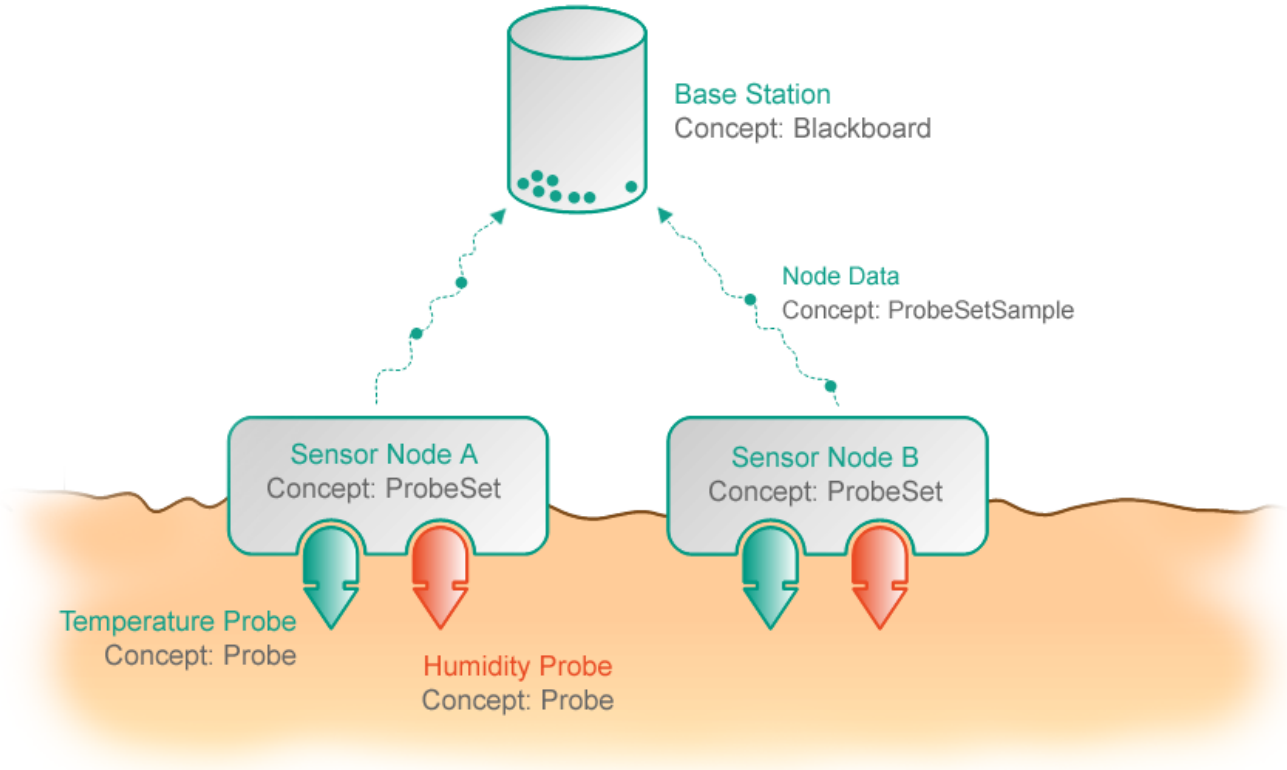
Pipes-and-Filters

- Meta-Model (EMF ECore)
  - Models define concrete filter chains
- Framework
  - Provides *PipesAndFiltersManager*
    - Simplifies building filter chains
    - Manages data flow source $\longrightarrow$ sink
  - Works with arbitrary persistence framework
    - SensorFramework, EDP2, ...
    - Implement custom *WriteStrategy*
  - Independent of Probe Specification $\Rightarrow$ reusable!

Outline

Concepts – Illustration

- Szenario: Agriculture
  - Several deployed sensor nodes which measure *temperature* and *humidity*
  - Base Station collects node data

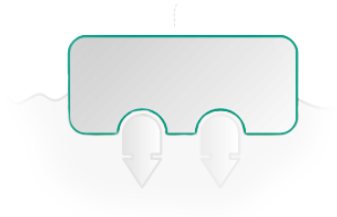


Probe – “Messfühler”

- Measures a single value + unit pair (JScience: “Measure”)
  - A measured value/unit pair is called ProbeSample
- Specifies **what** to measure, depending on probe type
  - Knows how to take a measurement
- Probe types
  - Current Time, CPU State, CPU Demand, . . .

ProbeSet – “Messpunkt”

- Encapsulates one or several Probes
- Measures a tuple of value + unit pairs
  - Dependent on contained probes
  - Multidimensional measurements possible
  - A set of measures is called ProbeSetSample
- Specifies **where** to take a measurement
- Example ProbeSet: (*CurrentTime*, *CPUState*)



Calculator

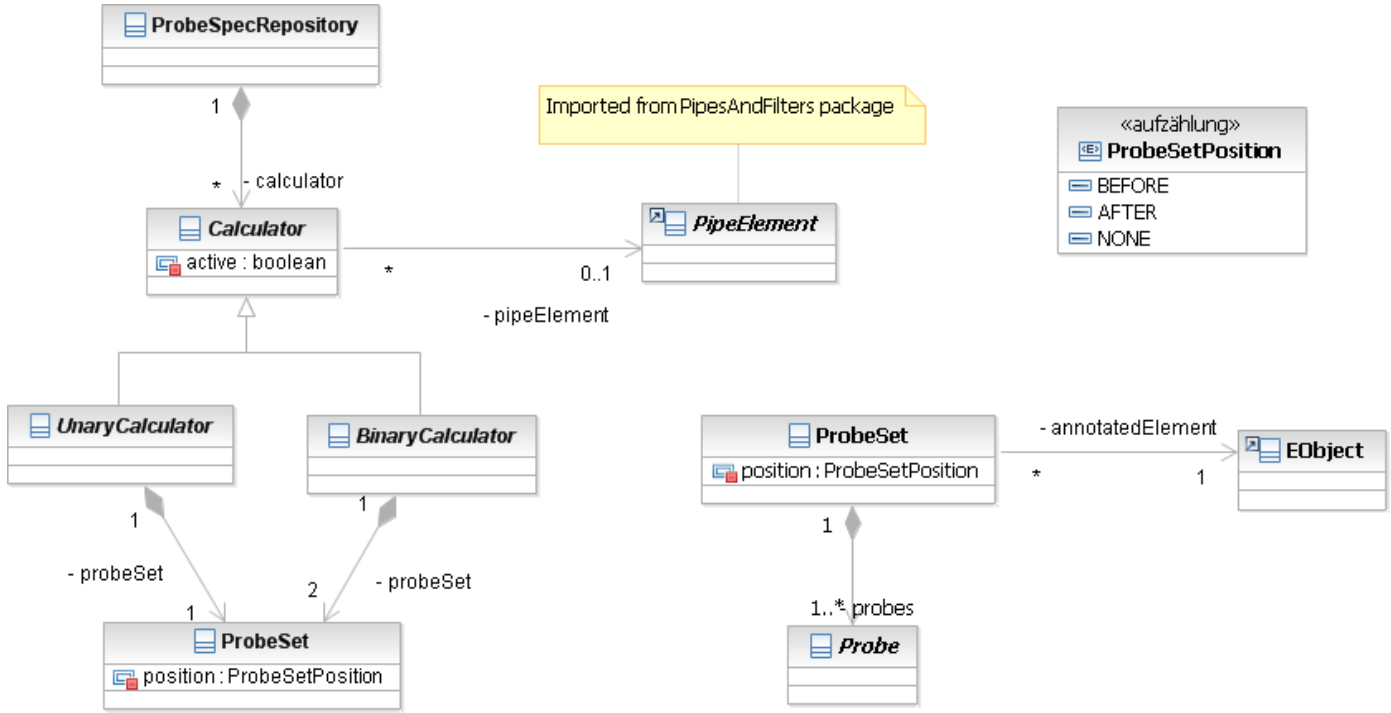
- Motivation: Measure (time) differences
- Input: ProbeSetSample(s)
  - i.e. the measurements from one or several ProbeSets
- Output: The measuring result (value + unit)
- Example: CPUStateCalculator
- Trivial calculator: Pass through
- Calculator types
  - Response Time, Waiting Time, CPU Demand, . . .

Calculator: Discussion

- Aren't the ProbeSetSamples already results?
  - Consider a response time measurement
    - 2 ProbeSets needed: At start position, at end position (e.g. of a service call)
    - Resulting ProbeSetSamples: $(time_{start})$, $(time_{end})$
    - Calculating $time_{end} - time_{start}$ produces result

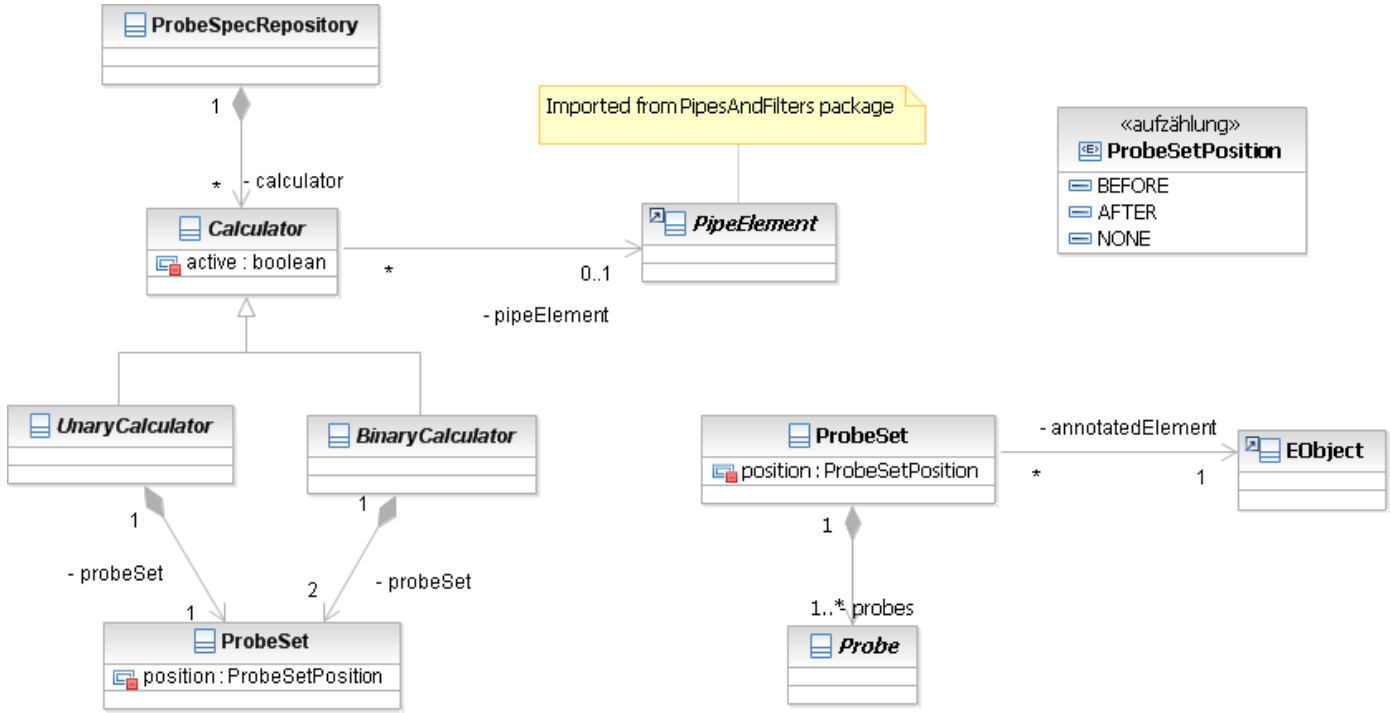
Outline

Meta-Model – Overview



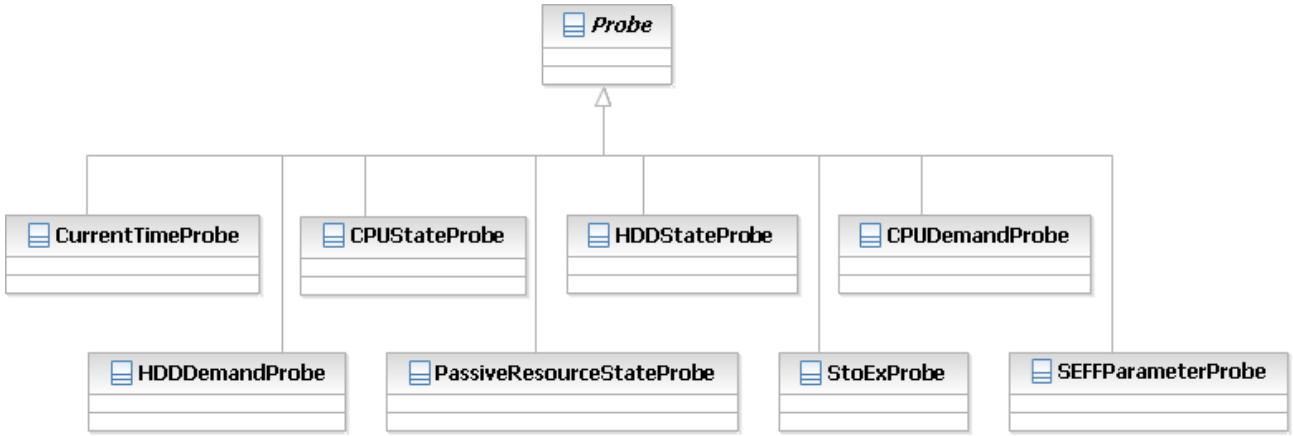
- Why a *Position* attribute?
  - Consider e.g. an *ExternalCallAction*: Position specifies whether to measure before or after the call

Meta-Model – Overview

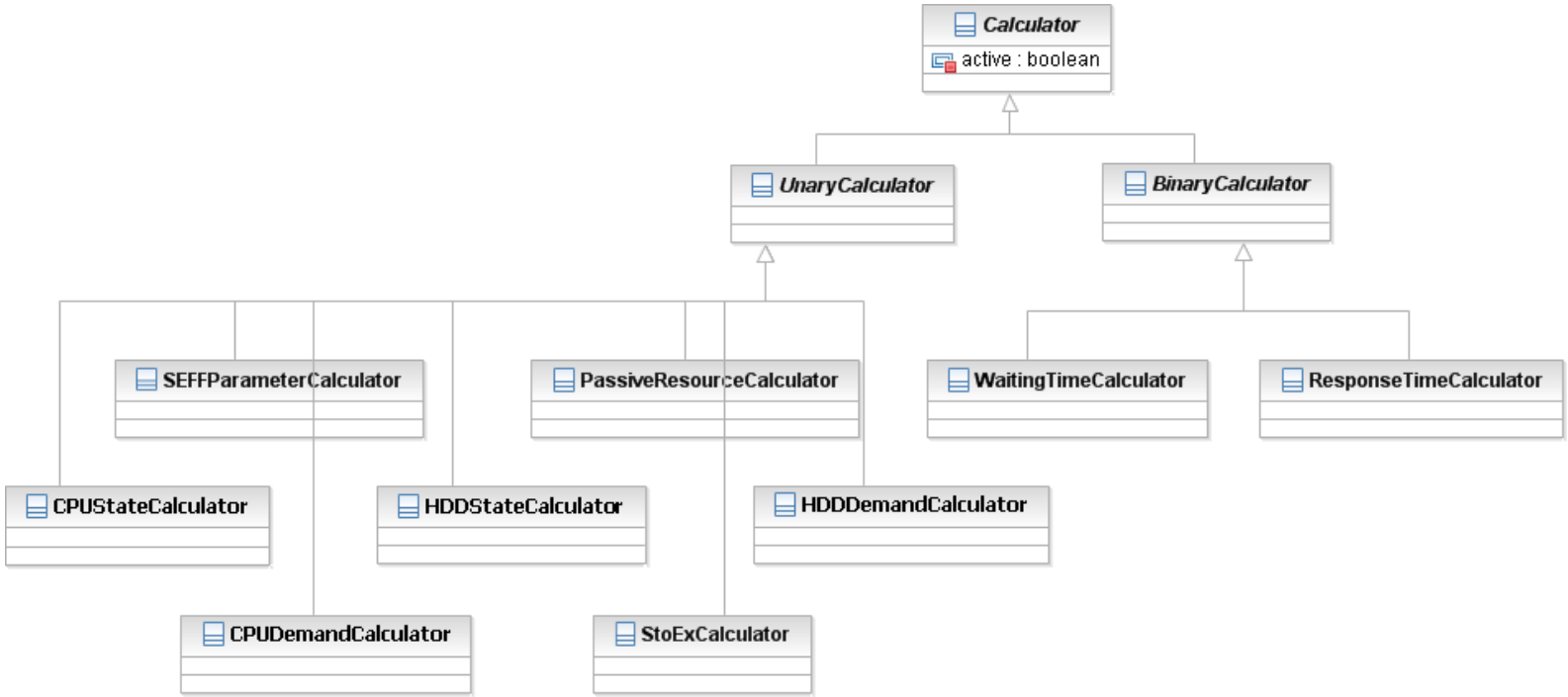


- Why a *Position* attribute?
  - Consider e.g. an *ExternalCallAction*: Position specifies whether to measure before or after the call

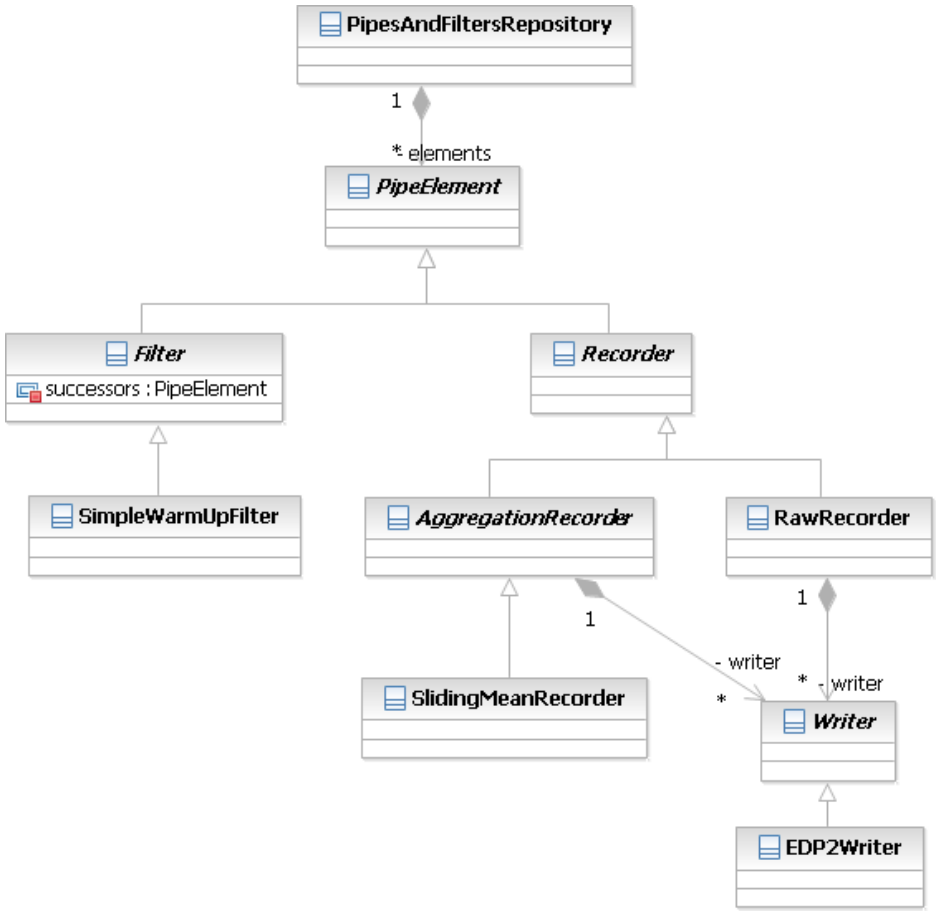
Meta-Model – Probes



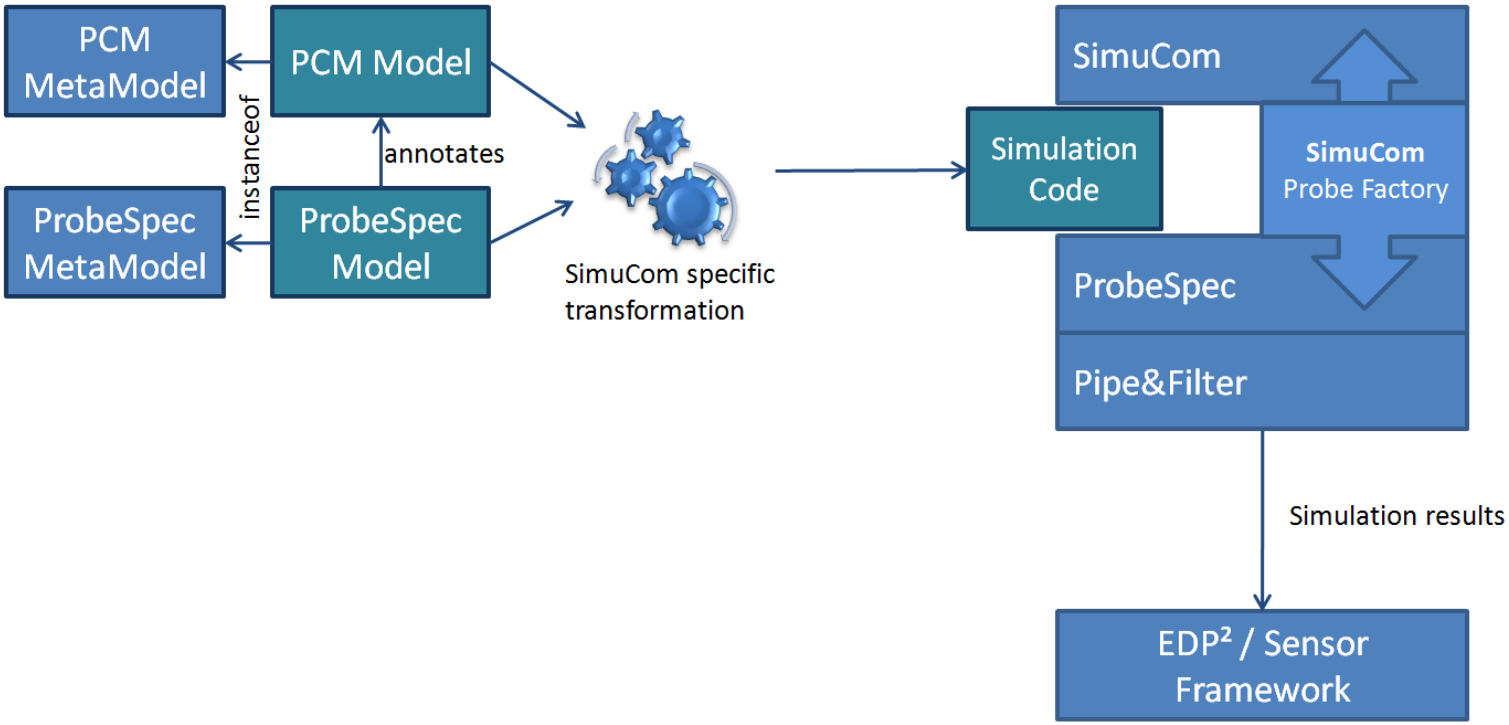
Meta-Model – Calculators



Meta-Model – Pipes-and-Filters

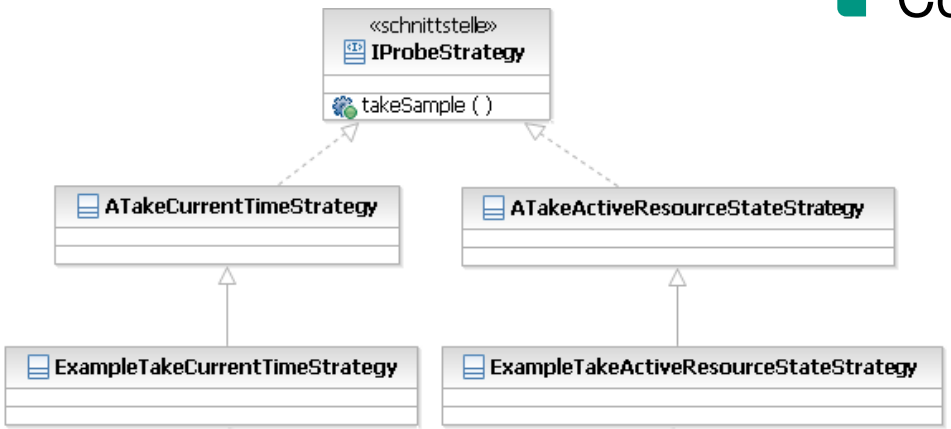


Usage Example: SimuCom



Outline

ProbeStrategies

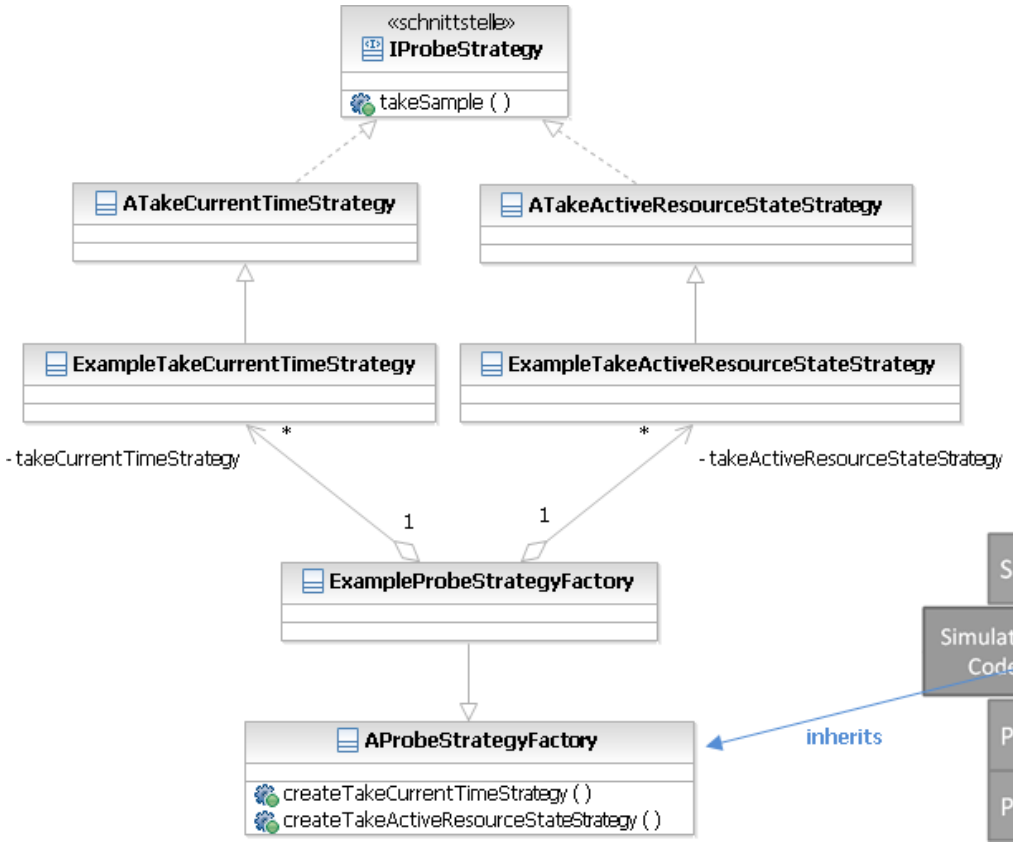


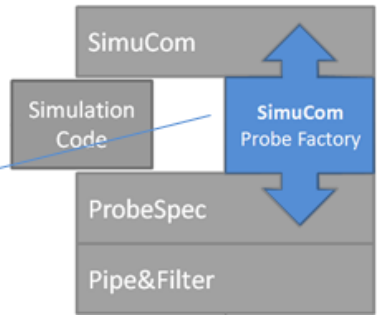
- ProbeStrategy
  - Concrete measuring method
    - Wall clock time:

      System.currentTimeMillis()
    - Simulation time:

      SimulationControl.getCurrentSimulationTime()
  - Specific implementation for each measuring environment
- Different implementations for different environments

ProbeStrategyFactory





- ProbeStrategyFactory
  - Bundles ProbeStrategies for specific measuring environments
  - One factory for each measuring environment

Blackboard

- Receives measurements
- Notifies calculators of new measurements
  - Observer Pattern
- *Context aware*
  - Simulation: Which simulated user raised the measurement?
  - Code instrumentation: Which thread caused the measurement?

Blackboard – Context awareness

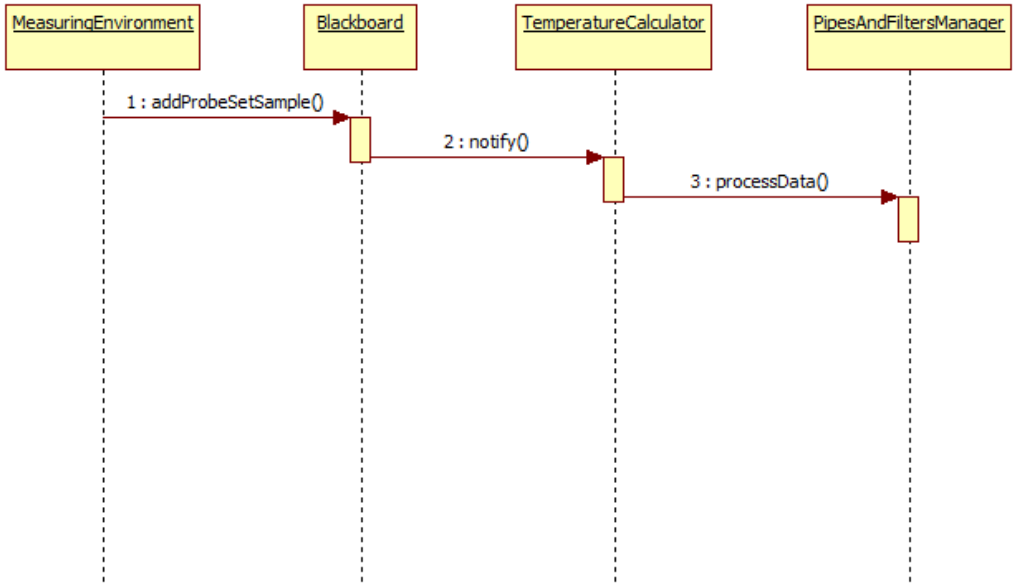
- Implemented by framework class *RequestContextID*
  - Represents context
- Present
  - Context represented simply by a String
    - e.g. *UserID* or *ThreadID*
  - Encapsulated by class *RequestContextID*
  - Does not support forks
- Planned
  - Context represented by a *stack* of Strings
  - Enables measuring e.g. response time within forks

Blackboard – Context awareness

- Implemented by framework class *RequestContextID*
  - Represents context
- Present
  - Context represented simply by a String
    - e.g. *UserID* or *ThreadID*
  - Encapsulated by class *RequestContextID*
  - Does not support forks
- Planned
  - Context represented by a *stack* of Strings
  - Enables measuring e.g. response time within forks

Blackboard – Interaction (Case 1)

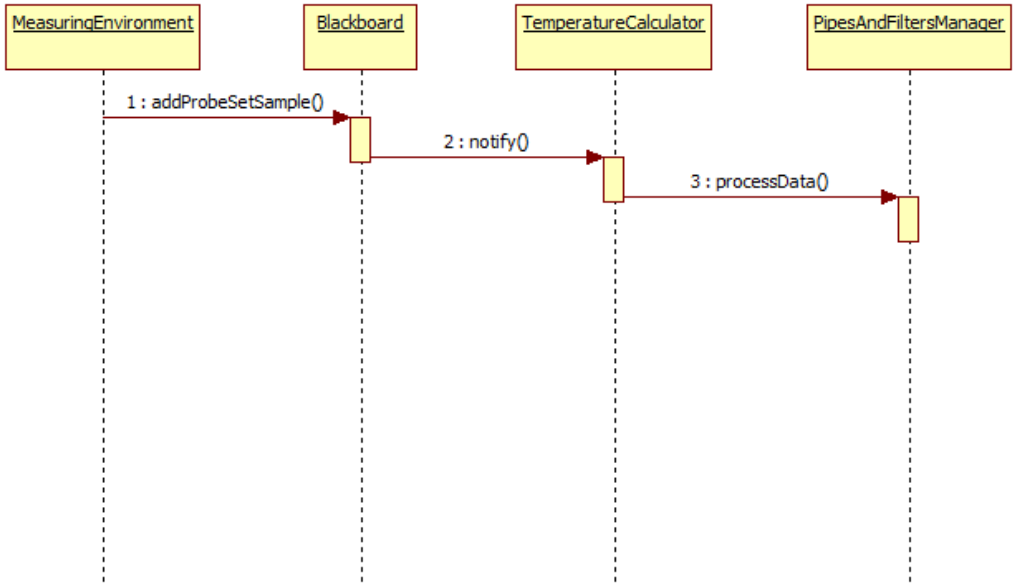
- Case 1: Unary Calculator



- Not depicted: addProbeSample() and notify() has parameter *ProbeSetSample*

Blackboard – Interaction (Case 1)

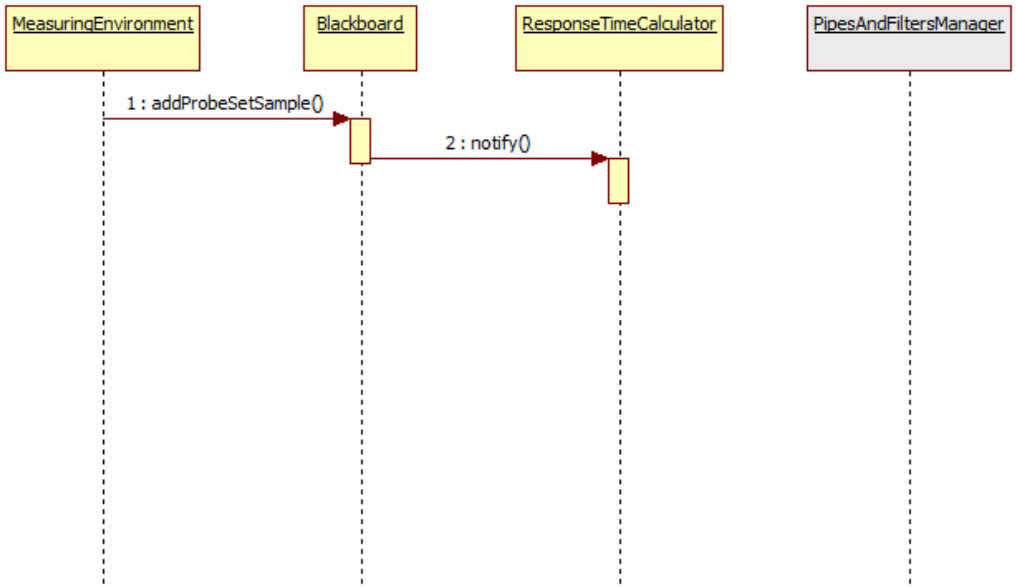
- Case 1: Unary Calculator



- Not depicted: addProbeSample() and notify() has parameter *ProbeSetSample*

Blackboard – Interaction (Case 2)

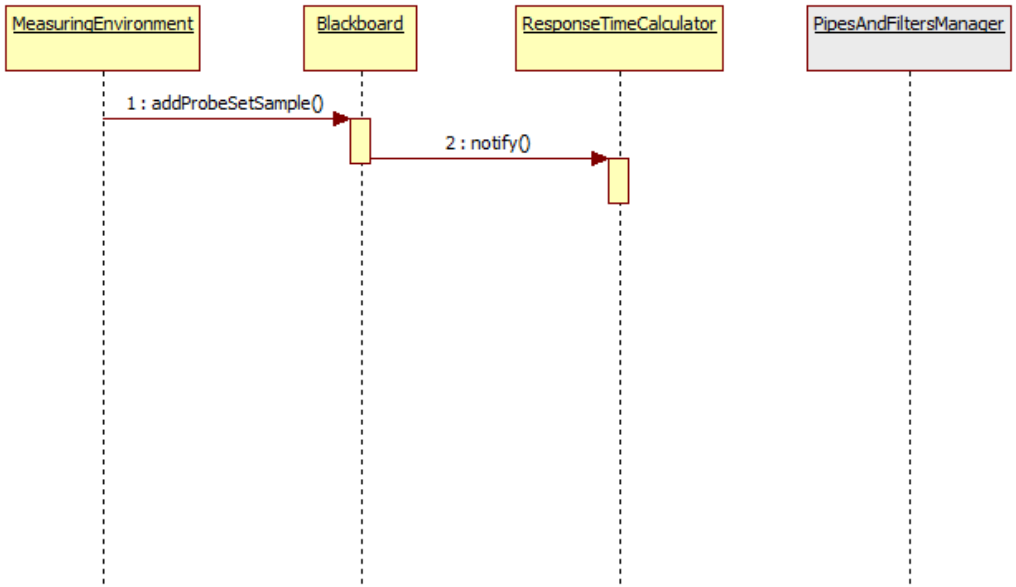
- Case 2: Binary Calculator, MeasuringEnvironment adds start ProbeSetSample to Blackboard



- ResponseTimeCalculator must wait for end ProbeSetSample (Case 3)

Blackboard – Interaction (Case 2)

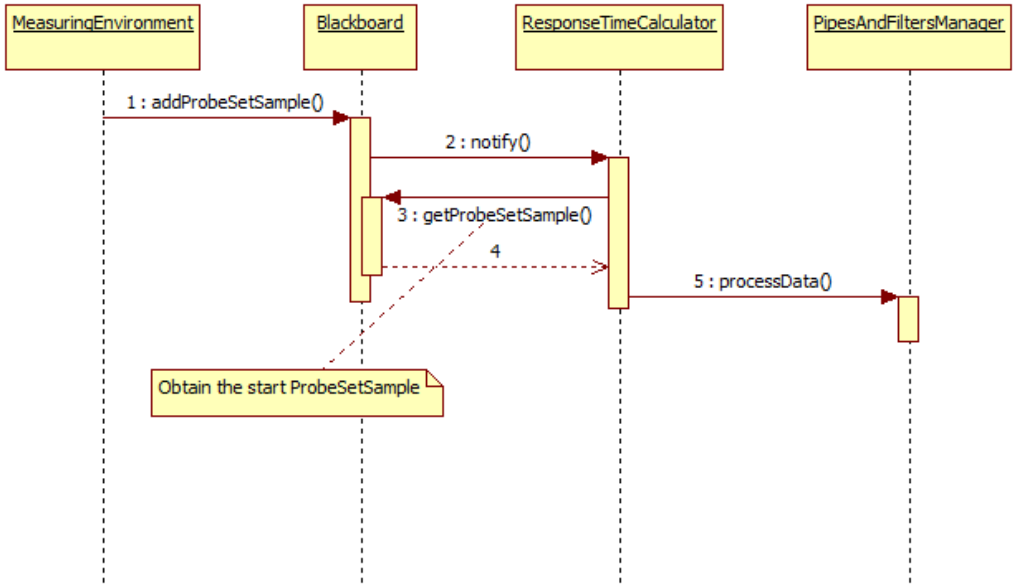
- Case 2: Binary Calculator, MeasuringEnvironment adds start ProbeSetSample to Blackboard



- ResponseTimeCalculator must wait for end ProbeSetSample (Case 3)

Blackboard – Interaction (Case 3)

- Case 3: Binary Calculator, MeasuringEnvironment adds end ProbeSetSample to Blackboard



Blackboard – Garbage Collection

- Obsolete ProbeSetSamples occupies memory
- Garbage Collection needed
- Currently: Basic implementation
  - Initial lifetime = 1 for each ProbeSetSample
  - Invoking *getProbeSetSample()* decreases lifetime by 1
  - Lifetime == 0 $\Rightarrow$ Garbage
- Desireable – future work
  - Get rid of side effects within *getProbeSetSample()*

Blackboard – Garbage Collection

- Obsolete ProbeSetSamples occupies memory
- Garbage Collection needed
- Currently: Basic implementation
  - Initial lifetime = 1 for each ProbeSetSample
  - Invoking *getProbeSetSample()* decreases lifetime by 1
  - Lifetime == 0 $\Rightarrow$ Garbage
- Desireable – future work
  - Get rid of side effects within *getProbeSetSample()*

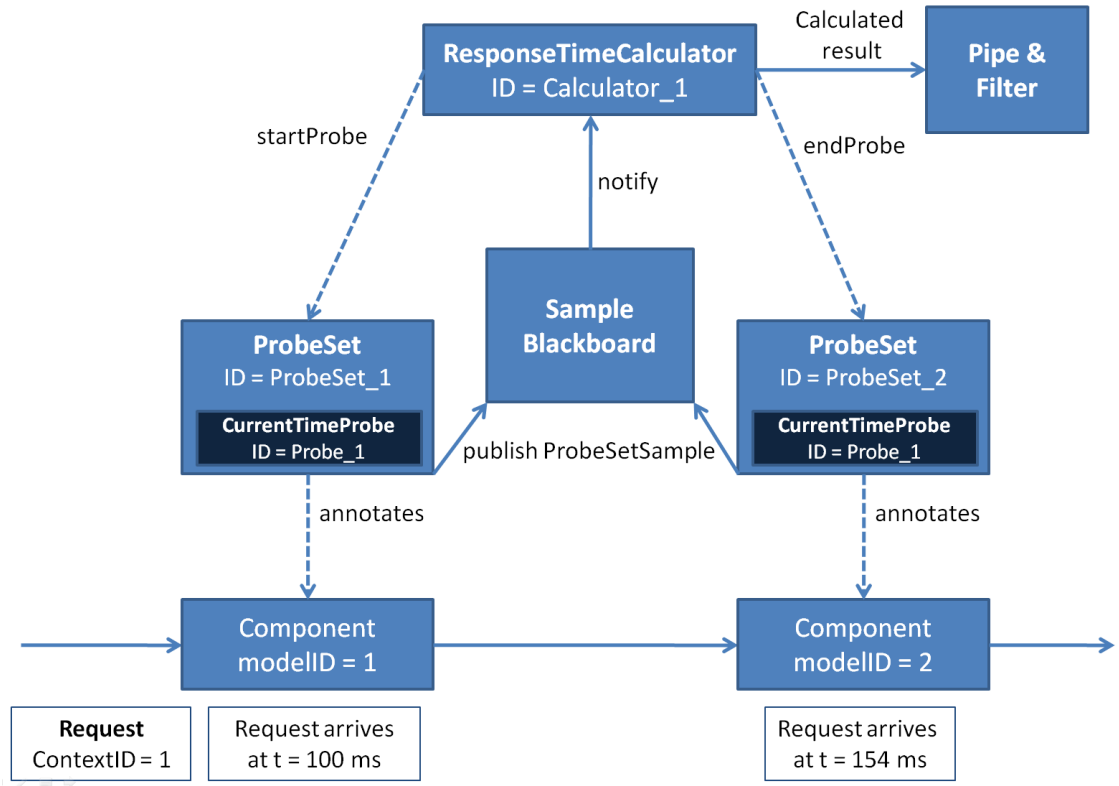
Blackboard – Garbage Collection

- Obsolete ProbeSetSamples occupies memory
- Garbage Collection needed
- Currently: Basic implementation
  - Initial lifetime = 1 for each ProbeSetSample
  - Invoking *getProbeSetSample()* decreases lifetime by 1
  - Lifetime == 0 $\Rightarrow$ Garbage
- Desireable – future work
  - Get rid of side effects within *getProbeSetSample()*

Outline

Demonstration

- Live Demo: Test Case “CalculatorAndPipesTest”



Outline

References, Further Information

- SPEEcl Praktikum 2009
  - Anforderungsanalyse – version 0.2 is final
  - Review Anleitung
  - Abschlusspräsentation
- Javadoc – there are extensive package documentations
- All documents resides in the SVN:

  /code/Palladio.ProbeSpecification/trunk/de.uka.ipd.sdq.probespec.framework/doc

Omitted

- Pipes-and-Filter
  - PipesAndFiltersManager
  - Recorder: RawRecorder vs. AggregationRecorder
  - MetaDataInit: Provides WriteStrategies details on the measurement