Review Probe Specification

Praktikum „Software Performance Engineering mit Eclipse"

Moritz Baum, Michael Faber, Philipp Merkle
Betreut von Anne Martens

Lehrstuhl für Software Design und Qualität (SDQ), Institut für Datenorganisation und Programmstrukturen (IPD), Uni Karlsruhe (TH)

1 Einleitung

1.1 Motivation

Um auf Basis des PCM Software-Performanz Analysen durchzuführen, kann ein Simulationsframework, wie z.B. SimuCom, verwendet werden. Dabei werden innerhalb eines Simulationslaufs verschiedene Messungen durchgeführt, wie etwa die Messung von Antwortzeiten, HDD-Auslastungen und CPU-Auslastung. Diese Messdaten dienen dann z.B. als Grundlage für die Visualisierung innerhalb von SimuBench. Bisher sind die Stellen, an denen gemessen wird, sowie die Art der jeweiligen Messung starr vorgegeben. Der Benutzer hat keinen Einfluss darauf, welche Messdaten erhoben werden. Insbesondere werden häufig nicht benötigte Messungen durchgeführt. Ebenso können neue Messmethoden nicht mit vertretbarem Aufwand umgesetzt werden.

Um dies zu ändern, soll im Rahmen der ProbeSpecification die freie Konfiguration eigener Messpunkte ermöglicht werden. Die Konfiguration umfasst die Position eines Messpunkts sowie die Art der Messung. Dazu wurde ein umfangreiches Framewerk ausgearbeitet, das im Folgenden beschrieben wird. Die ProbeSpecification umfasst dabei das Setzen von Messpunkten an beliebigen Stellen innerhalb der Simulation, die Spezifikation welche Metriken gemessen werden sollen sowie die Verarbeitung, Filterung, Aggregation und Persistierung der Messungen. Sämtliche Vorgänge sollen dabei vom Benutzer komfortabel bedienbar sein und zugleich komplexe Verarbeitungen ermöglichen. Die exakten Vorgaben können der Anforderungsanalyse entnommen werden.

1.2 Konzepte

Zur Messung beliebiger Werte innerhalb der Simulation dienen Probes. Eine Probe misst dabei jeweils einen festgelegten Wert. Welcher Wert gemessen wird, hängt vom Typ der Probe ab (z.B. den aktuellen CPU-Bedarf oder den aktuellen Zeitpunkt). Die Probe legt somit fest, was gemessen wird.

Eine Messung soll an beliebigen Stellen innerhalb der Simulation möglich sein, also beispielsweise innerhalb des Kontrollflusses einer SEFF. Zur Festlegung, wo gemessen werden soll, dienen ProbeSets. Ein ProbeSet beinhaltet eine

oder mehrere Probes und ist einem beliebigen Element des zugrundeliegenden Software-Architektur-Modells (z.B. PCM) zugeordnet. Beispielsweise könnte ein ProbeSet den aktuellen Zeitpunkt und die aktuelle CPU-Auslastung (spezifiziert durch entsprechende Probes) jedesmal nach einem Schleifendurchlauf innerhalb einer SEFF messen.

Zur Erzeugung beliebig komplexer Messergebnisse dient schließlich ein Calculator. Dieser erlaubt beliebige Operationen auf den Messungen einer oder mehrerer ProbeSets. Beispielsweise kann ein Calculator jeweils die Differenz von zwei Zeitpunkten aus unterschiedlichen ProbeSets bilden, um die dazugehörige Antwortzeit zu generieren. Durch die Verknüpfung verschiedener ProbeSets wird also die Semantik der Messung bestimmt.

Nach der Verarbeitung der Messwerte durch den Calculator sollen weitere Modifikationen der Ergebnisse möglich sein. Sofern vom Benutzer gewünscht, kann das Messergebnis unterschiedlichen Filtern zugeführt werden, die in beliebiger Reihenfolge angeordnet werden können. Ein solcher Filter könnte beispielweise die ersten 100 Messergebnisse eine Messreihe verwerfen und erst ab einem bestimmten Zeitpunkt weiterleiten, wie etwa ein Warmup-Filter, um die Auswertung erst zu beginnen, nachdem die Simulation angelaufen ist.

Daraufhin können die Messdaten entweder aggregiert oder in Rohform abgespeichert werden. Dazu dienen unterschiedliche Recorder, die wiederum eine variable Form der Persistierung der Daten erlauben. Standardmäßig vorgesehenes Persistenzframework ist dabei das EDP2.

Grundsätzlich gilt für alle Konzepte, dass komfortabel benutzerdefinierte Varianten implementiert werden können, so dass auch spezifische Messungen vorgenommen werden können.

2 Review Anleitung

2.1 Installation und Konfiguration benötigter Software

Installationsanleitung Eclipse

1. Java SE JDK 6 installieren
2. Eclipse Modeling Tools 3.4 installieren.
   Download: http://92.51.139.183/probespec/downloadEclipse.asp
3. Eclipse starten. Help -> Software Updates... wählen
4. Folgende Features auswählen:
   (a) Ganymede Update Site / Models and Model Development / Net4j Signalling Platform Runtime
   (b) Ganymede Update Site / Models and Model Development / Net4j Signalling Platform SDK
5. Install... wählen. Anschließend Finish auswählen

Konfigurationsanleitung Eclipse

1. Workspace herunterladen und entpacken.
   Download: http://92.51.139.183/probespec/WorkspaceEclipse.zip
2. Eclipse starten. Den zuvor entpackten Workspace auswählen.

Konfigurationsanleitung Rational Software Architect (RSA) Dieser Schritt kann auch übersprungen werden. Alle RSA-Modelle sind im Anhang abgebildet.

1. RSA-Workspace herunterladen.
   Download: http://92.51.139.183/probespec/WorkspaceRSA.zip
2. RSA starten. Den zuvor entpackten Workspace auswählen.

2.2 Durchführung

Gegenstand des Reviews ist der manuell erzeugte Code sowie das Metamodell der ProbeSpecification.

Metamodell Review Zur Ansicht des Metamodells kann der Rational Software Architect benutzt werden. Alternativ sind alle Modelle im Anhang abgebildet.
Fragestellungen:

- Wurden die Anforderungen korrekt umgesetzt, im Speziellen auch die Anforderung nach Erweiterbarkeit?
- Sind die Konzepte Probe, ProbeSet, Calculator und Recorder sinnvoll modelliert?

Code Review Der für das Review relevante Code befindet sich in den folgenden Plug-In Projekten:

de.uka.ipd.sdq.pipe Beinhaltet die *Pipes and Filters*, die Messergebnisse weiterreichen und verarbeiten.

de.uka.ipd.sdq.probespec.framework Beinhaltet die Umsetzung des Frameworks.

Die Einarbeitung in den Code kann ausgehend von einem Testfall geschehen, der die Funktionsweise des Frameworks sowie der Pipes and Filters darstellt. Der Testfall wird in Abschnitt 3 beschrieben.
Fragestellungen:

- Sind die umgesetzten Konzepte klar und verständlich?
- Lässt sich das Framework komfortabel um neue Messmethoden erweitern?
- Ist es möglich alle denkbaren Messmethoden zu implementieren?
- Ist der Code effektiv und effizient?

3 Testfall zur Veranschaulichung der Funktionsweise

Der Testfall *CalculatorAndPipesTest.java* soll die Funktionsweise des Frameworks veranschaulichen. Es wird dabei ein sehr einfaches Architekturmodell, bestehend aus zwei Modellelementen verwendet, die durch je ein ProbeSet annotiert sind. An den beiden Messpunkten (ProbeSets) befindet sich jeweils eine

CurrentTimeProbe. Mittels eines ResponseTimeCalculators soll dann die Differenz der zwei Zeitpunkte bestimmt werden und schließlich über eine Filterkette in das EDP² geschrieben werden.

In dem Testfall wird dabei ein fiktiver Request mit der ContextID 1 vermessen. ProbeSet_1 misst diesen bei t=100ms, ProbeSet_2 bei t=154ms. Beide ProbeSets veröffentlichen diese Ergebnisse auf dem zentralen Blackboard des Frameworks. Der zuvor als Observer registrierte ResponseTimeCalculator wird dann über die entsprechenden Veröffentlichungen informiert und reagiert sobald die Ergebnisse beider ProbeSets vorliegen (hier also dem Ergebnis von ProbeSet_2), indem er die Differenz berechnet und das Ergebnis an die Filterkette übergibt.

In diesem Testfall besteht die Filterkette aus einem ExampleFilter mit nachgeschaltetem Recorder, der die Rohdaten in das EDP² schreibt.

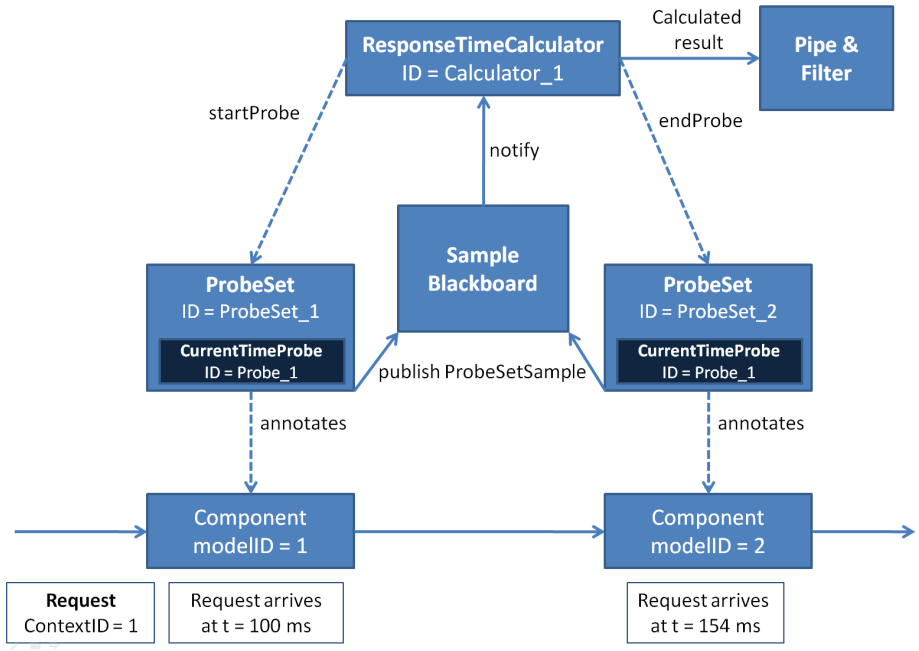

Abbildung 1. Testfall *CalculatorAndPipesTest.java*

Anhang

RSA-Modelle

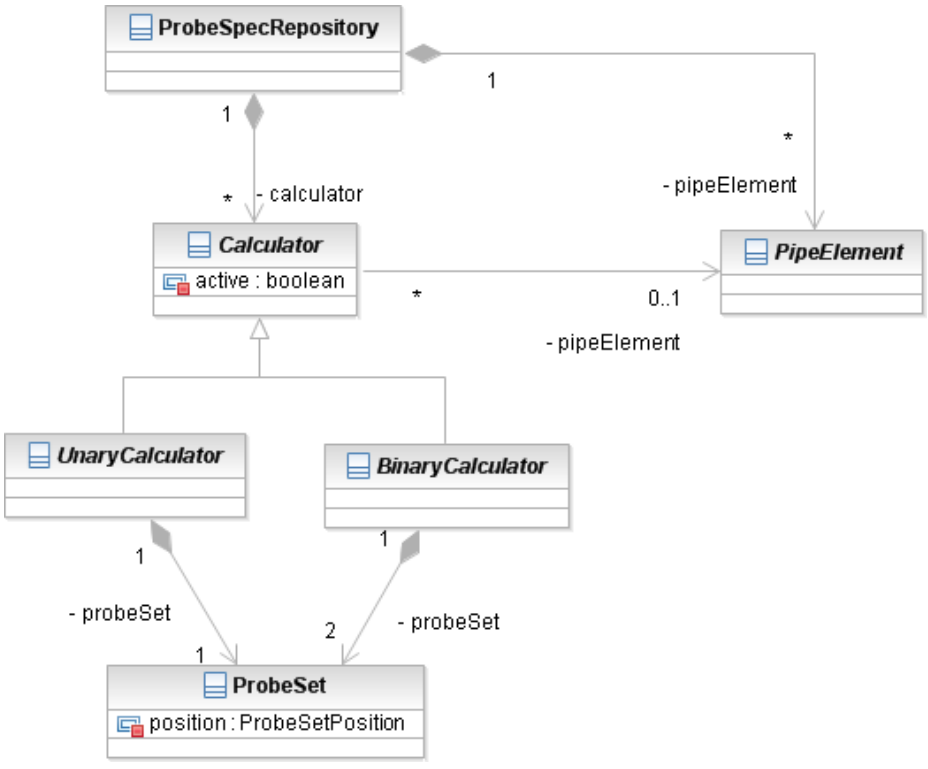

Abbildung 2. Overview

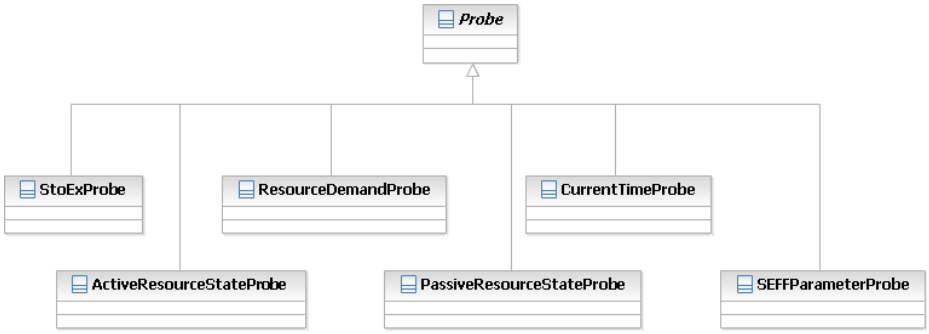

Abbildung 3. Probes

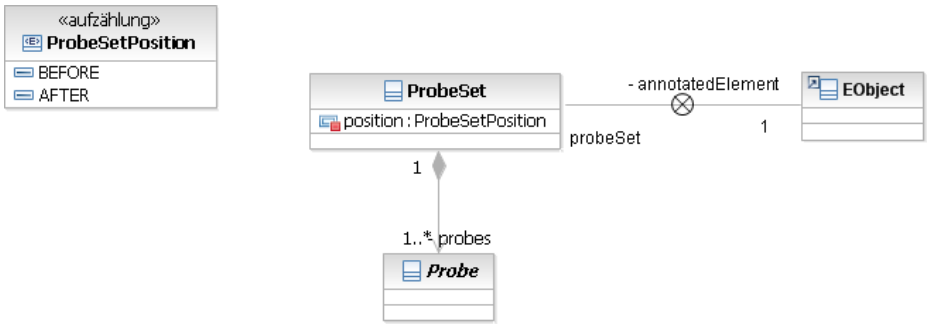

Abbildung 4. ProbeSets

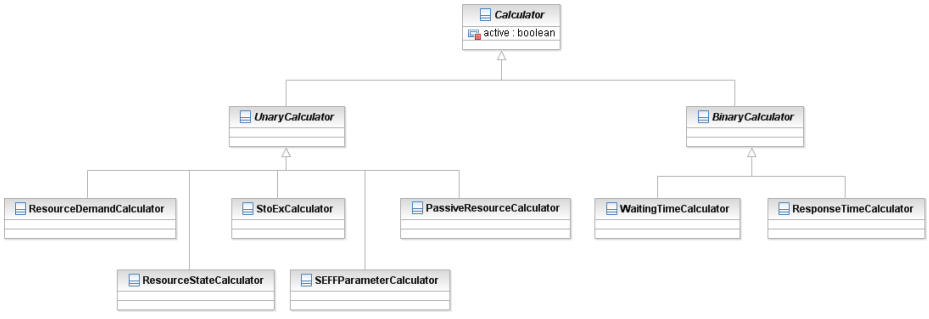

Abbildung 5. Calculators

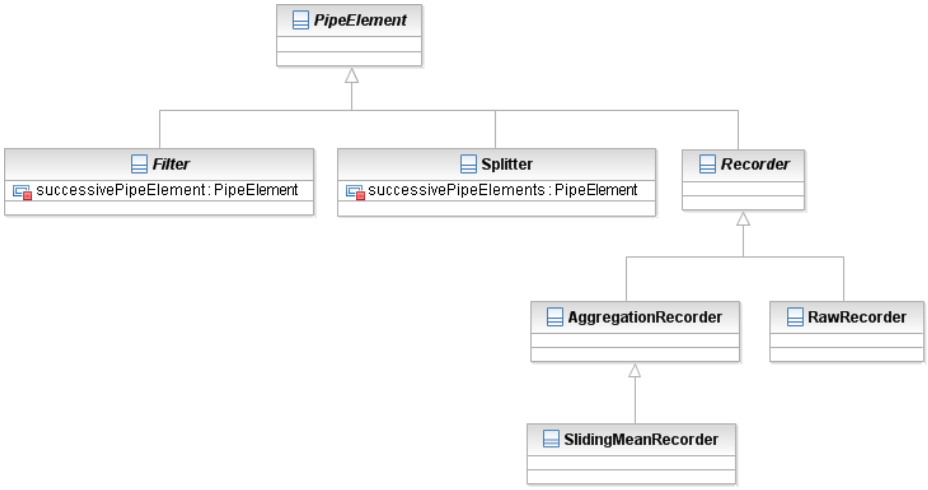

Abbildung 6. Pipes-and-Filters mit Recordern

Klassendiagramme

Abbildung 7. ProbeStrategies und ProbeStrategyFactory