Probe Specification
Anforderungsanalyse

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

Version 0.2, 04.06.2009
Praktikum Software Performance Engineering mit Eclipse
SDQ, IPD, Uni Karlsruhe

1 Ausgangssituation und Zielsetzung

1.1 Ausgangssituation

- Sensoren werden automatisch während der M2T-Transformation nach SimuCom erstellt
- Keine Möglichkeit, nicht interessierende Sensoren zu deaktivieren
- Zusätzliche Sensoren nur mittels Modifikation der M2T-Transformation und des SimuCom-Frameworks möglich
- Verwendung des SensorFramework 1.0
- Sensoren nur möglich an Service Calls, Usage Scenarios, Processing Resources, Communication Link Resources

1.2 Zielsetzung

- Manuelle Erstellung von Probes
  - Must-have: Baumeditor zur Erstellung von Probes/Recordern
  - Nice to have: Grafischer Editor, evtl. Erweiterung vorhandener GEF-Editoren
- Aktivieren/Deaktivieren vorhandener Calculators
- Zusätzliche Probe-Typen möglich per Implementierung einer *AbstractProbe*
- Verwendung des EDP² aka SensorFramework 2.0

- Probes an mehreren Stellen möglich, insbesondere auch innerhalb von SEFFs
- Unabhängigkeit von SimuCom: Hierdurch beispielsweise auch Codeprototypen etc. messbar (Messpunkte entsprechend der ProbeSpec setzen)

2 Überblick

Anhand der angehängten Abbildung 1 sollen zunächst die einzelnen Funktionen und Begrifflichkeiten anhand eines Beispiels (ResponseTimeCalculator) verdeutlicht werden.

Die ProbeSets (Messpunkte) bestehen aus mehreren Probes (Messgeber), die bestimmen, welche Daten an der jeweiligen Stelle erhoben werden. Die gemessenen Daten werden zusammen mit den entsprechenden IDs (z.B. ThreadID) zu einem Tupel zusammengefasst. Die Messwerte werden als JScience-Objekte gespeichert (hier durch eckige Klammern angedeutet).

Ein Calculator verknüpft die Werte der ProbeSets zu einem Messergebnis und achtet darauf, dass nur Werte mit passenden IDs verknüpft werden. Der Calculator sorgt auch für die semantischen Informationen, die die Art des Messergebnisses beschreiben.

Der Calculator übergibt die so anfallenden Tupel dann an eine Pipe & Filter Architektur, die für die Persistierung im EDP² zuständig ist. Diese soll durch ein einheitliches Interface leicht durch neue Filter und Aggregatoren erweiterbar sein. Das Interface soll auch ermöglichen andere Quellen als den Calculator für die Pipe & Filter Architektur zu verwenden. Hier wäre zum Beispiel ein EDP2Reader vorstellbar, um eine nachträgliche Offline Aggregation zu realisieren.

In dem weiteren Verlauf werden die einzelnen Elemente weiter spezifiziert.

3 Funktionale Anforderungen

/LF010/ Spezifikation von Messpunkten (Konzept: *Probes*)

1. Eine *Probe* misst einen einzelnen Wert in Verbindung mit dessen Einheit (Anmerkung: Weder die PCM Tools, noch SimCom unterstützen derzeit die Verwendung von Einheiten. Im PCM-Metamodell sind Einheiten bereits berücksichtigt).
2. Durch den konkreten Typ der verwendeten Probe wird spezifiziert, welche Daten gemessen werden sollen. Eine Probe gibt somit an, **was** gemessen werden soll.
3. Spezifische Probes sollen unabhängig vom PCM sein, so dass diese mit beliebigen Software-Architektur-Modellen auf Basis von Ecore verwendbar sind.
4. Es gibt mindestens eine abstrakte Probe. Konkrete Probes sind Erben und legen fest, welche Daten erhoben werden sollen. Beispiele für Daten sind: Zeit, Datum, Auslastung, Thread Anzahl.

5. Folgende Probe-Typen werden benötigt

   ResourceStateProbe: Misst den Status einer Ressource: IDLE, BUSY mit 1 Job, BUSY mit n Jobs.

   PassiveResourceStateProbe: Misst die Auslastung von passiven Ressourcen.

   StoExProbe: Misst das Resultat eines stochastischen Ausdrucks.

   ResourceDemandProbe: Misst den Bedarf eine bestimmten Ressource. Die Einheit wird dabei im entsprechenden SEFF spezifiert.

   SEFFParameterProbe: Misst die konkrete Ausprägung eines Parameters.

   CurrentTimeProbe: Misst den aktuellen Zeitpunkt.

6. Ein Messwert soll nur dann erhoben werden, wenn mindestens ein zugehöriger Calculator aktiviert ist, um den Performance-Overhead des Probe Frameworks klein zu halten.

/LF015/ Spezifikation von Messpunkten (Konzept: *ProbeSet*)

1. Mittels sogenannter *ProbeSets* soll spezifiziert werden können, an welchen Stellen eines Software-Architektur-Modells welche Daten gemessen werden sollen. Wähend einer Software-Performanz-Analyse werden auf Basis der spezifizierten ProbeSets Messwerte derjenigen Modellelemente erhoben, die mit einem ProbeSet versehen sind. ProbeSets geben somit an, **wo** gemessen werden soll.

2. Ein ProbeSet fasst mehrere Probes zu einem Messpunkt zusammen. Welche Daten an einem Messpunkt erhoben werden, ist durch die beinhalteten Probes bestimmt.

3. ProbeSets beinhalten folgende Informationen: Verweis auf mindestens eine Probe sowie das annotierte Ecore-Modellelement.

4. Ein Messwert-Tupel eines ProbeSets beinhaltet die Messwerte der einzelnen Probes, eine *ThreadID* und eine optionale *RequestID*. Ein Messergebnis wird somit durch folgendes Tupel repräsentiert: $(probeRes_1, ..., probeRes_n, threadID, requestID)$, wobei $probeRes_i$ für das das von Probe i gelieferte Resultat steht.

5. Die Messposition (BEFORE, AFTER) eines ProbeSets bezogen auf das annotierte Modellelement soll spezifiziert werden können. Hiermit lässt sich z.B. für ein ProbeSet an einem ServiceCall festlegen, ob die Messung vor Aufruf oder nach Beendigung des ServiceCalls durchgeführt werden soll. Entwickler von Completions-Transformationen müssen eine sinnvolle Weiterverwendung der Messposition sicherstellen.

/LF020/ Berechnung von Messergebnissen aus Messpunkten (Konzept: *Calculator*)

1. Ein *Calculator* verknüpft die Tupel, die von ProbeSets kommen zu einem mit Semantik angereicherten Messergebnis.

2. Bei der Verknüpfung muss der Calculator darauf achten nur Werte mit passenden IDs zu verknüpfen. In dieser ersten Version der ProbeSpec soll das ID Konzept anhand von ThreadIDs exemplarisch verdeutlicht werden.

3. Das neue Messergebnis-Tupel wird dann an die Pipe & Filter Architektur übergegeben, um im EDP² persistiert zu werden.

4. Calculators sollen für einzelne Simulationsläufe aktiviert oder deaktiviert werden können. Hierfür darf der Simulationscode nach jeder (De-)Aktivierung neu generiert werden, d.h. eine erneute M2T-Transformation ausgeführt werden.

5. Calculator-Typen (die Architektur soll es ermöglichen diese Menge je nach Bedarf einfach zu erweitern):
   - ResourceStateCalculator
   - PassiveResourceCalculator
   - StoExCalculator
   - ResourceDemandCalculator
   - SEFFParameterCalculator
   - ResponseTimeCalculator
   - WaitingTimeCalculator

/LF030/ Ergebnissaggregation über Messpunkten (Konzept: *Filter und Recorder*)

1. Das Speichern der Messergebnisse in ein Persistenzframework wird durch verschiedene *Recorder* vorgenommen.

2. Als Persistenzframework soll das EDP² verwendet werden.

3. Zum Anpassen der Ergebnisse stehen verschiedene *Filter* zur Verfügung. Diese können nach dem Pipes-and-Filters-Muster in beliebiger Reihenfolge angeordnet werden, um eine hohe Flexibilität zu gewährleisten.

4. Die Recorder nehmen dabei eine Sonderstellung unter den Filtern ein, da sie zusätzlich zur Filterung für die anschließende Übergabe der Daten an das EDP² verantwortlich sind. Da die Art der Aggregation auch die Art der Datenstruktur impliziert, bilden Filter (Verarbeitung der Daten) und Data Sink (z.B. Schreiben der Daten ins Persistenzframework) an dieser Stelle eine Einheit in Form eines Recorders. Dabei sind zunächst folgende Typen von Recordern vorgesehen:
   - Raw-Recorder (einfache Weiterleitung der ermittelten Rohdaten)
   - Aggregation-Recorder (erlauben Anpassen der Rohdaten. Zunächst sollen beispielhaft einige Basistypen implementiert werden, weitere Aggregations-Typen können später hinzugefügt oder vom Benutzer erzeugt werden. Beispiel für einen Aggregationsrecorder: Sliding-Mean-Recorder)

5. Den Recordern können in beliebiger Anzahl und Reihenfolge Filter vorgeschaltet werden, welche die Daten nach verschiedenen, benutzerdefinierten Kriterien Filtern (z.B. WarmUp-Filter)

6. Der Splitter als spezieller Filtertyp erlaubt es, den Datenfluss auf mehrere Ausgabekanäle zu verteilen. Dadurch können dieselben Daten auf unterschiedliche Weise weiterverarbeitet werden.

7. Grundsätzlich soll für alle Filtertypen durch ein einheitlich definiertes Interface die Möglichkeit bestehen, benutzerdefinierte Filter einzubinden.

8. Neben dem Datenstrom aus dem Calculator sollen auch andere Datenquellen eingebunden werden können (z.B. FileReader, EDP2Reader).

9. Die Aggregation der Daten erfolgt grundsätzlich online, die Daten werden also direkt aggregiert und weitergeleitet. Offline Aggregierung von Daten kann realisiert werden, indem Daten aus dem EDP² erneut in eine Pipe-and-Filter Struktur eingespeist werden (EDP2Reader, siehe Punkt 8).

10. Grundsätzlich soll es auch möglich sein, andere Senken anzubinden als EDP² (etwa zur Visualisierung).

11. Der Verweis von aggregierten Daten auf Roh-Daten sollte im EDP² durch Recorder entsprechend gesetzt werden.

Abbildung 2 zeigt eine beispielhafte Anordnung von Filtern.

/LF040/ Berücksichtigung von Skalenniveaus

1. Es soll die Konsistenz von Calculator Operationen bezüglich der Skalenniveaus beteiligter Probes berücksichtigt werden.

2. Inkonsistenzen sollen möglichst schon bei der Spezifikation der Messtrecken angezeigt werden.

/LF050/ Werzeug-unterstützte Erstellung der Probes

1. Es soll möglich sein, mittels Baumeditor innerhalb Eclipse die gewünschten Probes, ProbeSets, Calculators und Recorder anzulegen und zu konfigurieren.

2. Optional: Grafischer Editor zur Erstellung von Probes, ProbeSets, Calculators und Recorder; evtl. integriert in die vorhandenen GEF-Editoren. Außerdem denkbar ist ein Wizard.

/LF060/ Unabhängigkeit von SimuCom und PCM

1. Die Probe Specification soll weitestgehend unabhängig von SimuCom sein.
2. Die Probe Specification soll weitestgehend unabhängig vom PCM sein.
3. Ausgehend von der Probe Specification sollen in Zukunft auch Transformationen abgeleitet werden können, die Messmodelle für z.B. implementierte Systeme generieren können (Dieser Punkt ist nur ein Beispiel für die gegebenen Möglichkeiten, die durch die Unabhängigkeit entstehen und wird im Rahmen dieser Umsetzung nicht realisiert).

/LF070/ Framework

1. Das Framework soll über eine API die Messlogik zur Verfügung stellen.
2. Die konkreten Probe Implementierungen sollen erweiterbar sein und über das Factory Pattern realisiert werden.

/LF080/ Testfälle

1. Die korrekte Funktionsweise soll anhand von Testfällen gezeigt werden.
2. Dabei muss darauf geachtet werden, dass der probespezifische Code durch eine Transformation erzeugbar ist.
3. Testcases sollen gleichzeitig als Dokumentation der Verwendung des Frameworks dienen können.

4 Glossar

Probe: Eine Probe misst einen atomaren Wert in Verbindung mit dessen Einheit. Eine Probe gibt an, **was** gemessen wird.

ProbeSet: Ein ProbeSet fasst mehrere Probes zu einem Messpunkt zusammen. ProbeSets annotieren Modellelemente und geben somit an, **wo** gemessen wird.

Messpunkt: Siehe ProbeSet.

Messgeber: Siehe Probe.

Calculator: Ein Calculator verknüpft die Tupel, die von Probes kommen zu einem mit Semantik angereicherten Messergebnis. Das neue Messergebnis-Tupel wird dann an die Pipe & Filter Architektur übergegeben, um im EDP² persistiert zu werden.

Recorder: Recorder stellen als letzte Elemente der Pipe die Datensenke und somit die Schnittstelle zum EDP² dar. Die Daten können dabei in benutzerdefinierter, aggregierter Form oder als Rohdaten weitergeleitet werden. Ein aggregierender Recorder fasst somit Filter und Datensenke zusammen.

Filter: Ein Filter dient zur Filterung und anschließenden Weiterleitung eingehender Daten nach bestimmten, benutzerdefinierten Kriterien (Beispiel: WarmUpFilter)

Splitter: Ein Splitter stellt einen speziellen Filtertypen dar, der zur Aufteilung des Datenstroms auf mehrere (n>1) Ausgabekanäle dient.

Anhang

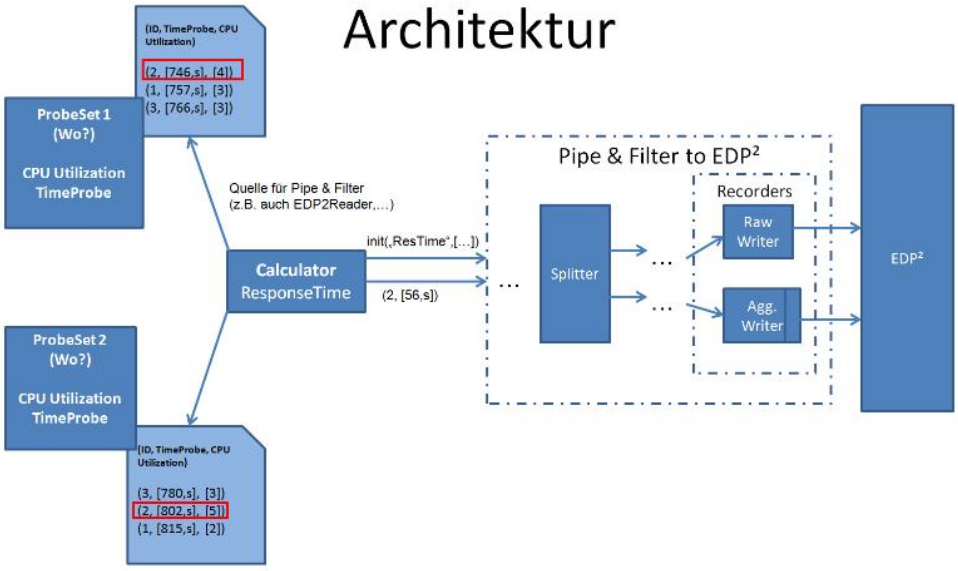

Abbildung 1: Beispielhafter Aufbau für eine Antwortzeit-Messung

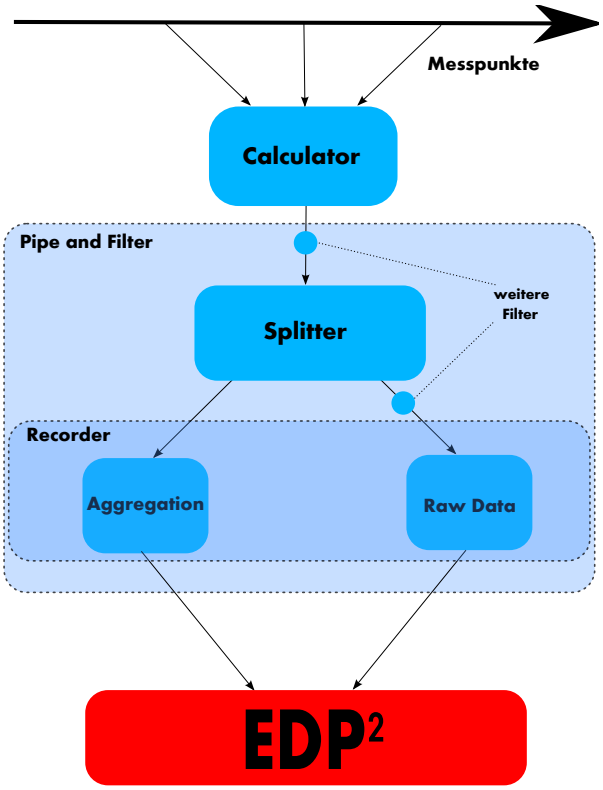

Abbildung 2: Struktur nach dem Pipe-and-Filters-Muster