EDP2 in PerOpteryx

Praktikumsbericht

Shengjia Feng
Praktikum Software Quality Engineering mit Eclipse I Wintersemester 2014/15
Matrikelnummer: 1629650
Betreuerin: Dr.-Ing. Anne Koziolek

Inhaltsverzeichnis

1 Einleitung

1.1 PerOpteryx

PerOpteryx ist ein Framework, welches Softwaredesigns im frühen Stadium auf unterschiedliche Kriterien hin optimieren kann. Diese Kriterien können zum Beispiel die Performance bei einem bestimmten Use Case sein oder auch die Allokation von Ressourcen.

Die Grundlage der Designanalyse bietet dabei unter anderem der Palladio Simulator, der auf Basis von mehreren Modellen die Software simulieren kann. Aus den ermittelten Performance-Ergebnissen in PerOpteryx kann dann die Pareto-optimale Lösung für die getesteten Kriterien gefunden werden.

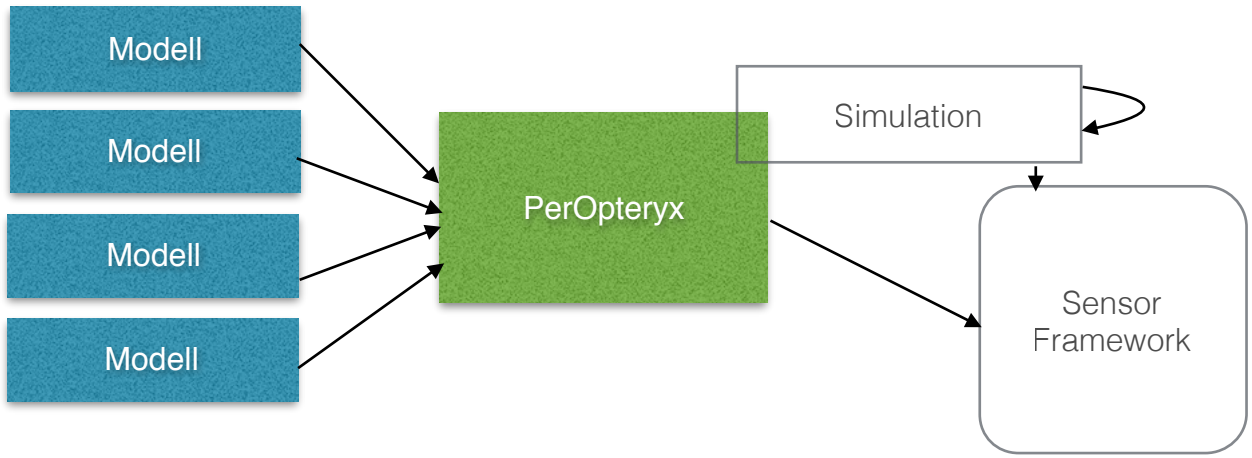

Abb. 1 : Aufbau PerOpteryx

Abb. 1 verdeutlicht den Aufbau und die Funktionsweise von PerOpteryx. Es werden (mindestens) 4 Modelle eingegeben für einen Use Case gemäß den Anforderungen des Palladio Simulators. Das sind zum Beispiel das Allocation Model (Modellierung des Deployments auf Hardware), die SEFFs (*Service Effect Specification*, modelliert die Vorgänge in einer Komponente), das Usage Model (Modellierung der Benutzung) und das Component Model, welches die Abhängigkeiten zwischen Komponenten modelliert.
Es werden anhand dieser Modelle mit einstellbaren Freiheitsgraden wiederholt Simulationen durchgeführt und die Ergebnisse dieser Simulationen im Sensor Framework gespeichert. Diese Ergebnisse werden dann von PerOpteryx wieder abgefragt, um diese Ergebnisse entsprechend weiter zu verarbeiten.

1.2 EDP2 und Sensor Framework

EDP2 steht für „Experiment Data & Persistency Framework“ und dient dazu, Ergebnisse sogenannter Experimente zu persistieren, also zu speichern. Experimente sind dabei Simulationen unterschiedlicher Konstellationen von Input-Modellen. Die Ergebnisse werden in geeigneten Strukturen gespeichert und können dementsprechend ausgelesen werden. Das Sensor Framework kann als ältere Version des EDP2 gesehen werden und dient dem gleichen Zweck. De interne Aufbau der Datenhaltung ist jedoch grundverschieden. Da die interne Struktur des Sensor Frameworks in diesem Praktikum nur wenig von Bedeutung ist bzw. eine grobe Sicht bereits genügt, wird darauf im Folgenden nicht weiter eingegangen.

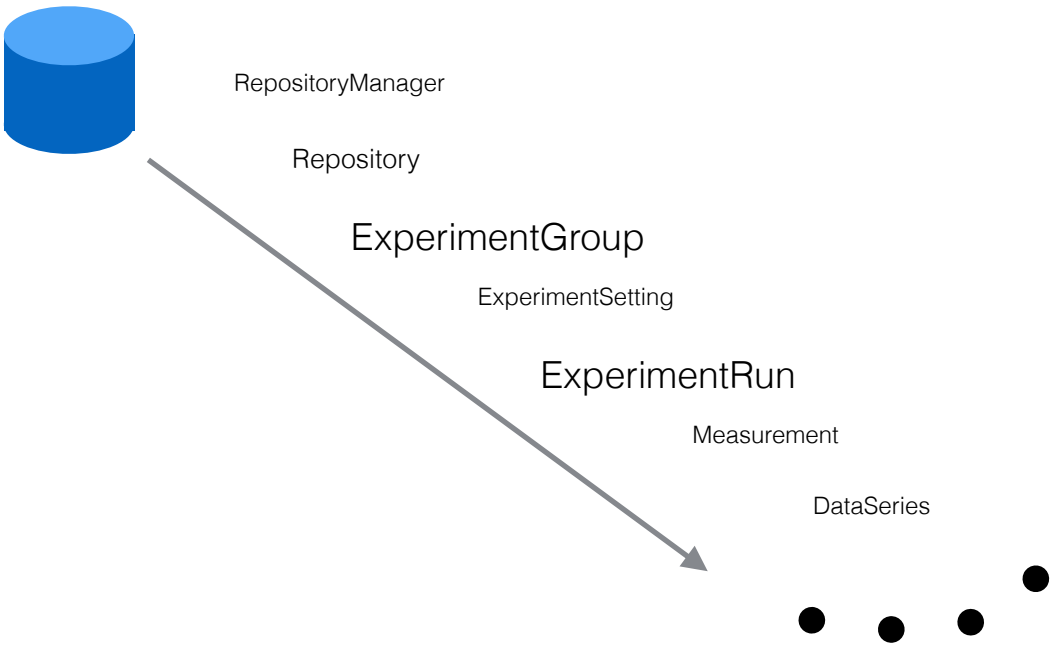

Abb. 2 : Interner Aufbau des EDP2

In Abb. 2 wird die interne Speicherstruktur des EDP2 skizziert. Die zentrale Speichereinheit ist das Singleton *RepositoryManager*, welches eine Menge an Repositories halten kann. Ein Repository kann mehrere *ExperimentGroups* enthalten. Dem Namen entsprechend enthält eine *ExperimentGroup* die Ergebnisse mehrerer Experimente, idealerweise logisch verwandt, was jedoch nicht direkt gefordert ist durch EDP2.
Jede Konstellation der Input-Modelle gemäß der Modifikationsfreiheiten, für die Experimente - also Simulationen in diesem Fall - durchgeführt wird, wird in EDP2 durch ein *ExperimentSetting* ausgedrückt. Erst ein *ExperimentSetting* enthält letztendlich Durchläufe bzw. Durchführungen der Experimente, die *ExperimentRuns.* Das hat den Vorteil, dass für zwei Durchführungen eines Settings nicht multiple Instanzen von *ExperimentSetting* erstellt werden.
Jeder *ExperimentRun* enthält dann wiederum *Measurements,* die tatsächlichen Messungen. Jedes Measurement besteht dabei aus 2 *DataSeries* im regulären Fall, nämlich eine *DataSeries* für die Messzeitpunkte und eine für die gemessenen Werte. Je nach Ziel der Messung können hier auch weitere Dimensionen der Messergebnisse hinzugefügt werden. Die *DataSeries* enthalten dann die Werte, ähnlich einer einfachen Liste.

2 Motivation

Die vorher bestehende Version des PerOpteryx Frameworks speicherte die Simulationsergebnisse im Sensor Framework. Dieses entspricht dem Grundnutzen her dem EDP2 Framework, jedoch mit geringerem Funktionsumfang. Da das Sensor Framework aus diesem Grund veraltet ist, sollte im PerOpteryx die Datenhaltung durch das neuere EDP2 abgelöst werden. Unter anderem sind bereits nützliche Funktionen basierend auf EDP2 implementiert wie die Visualisierung der darin gespeicherten Daten[1].

Auf dem bisherigen Stand verfügte das Simulationsmodul „SimuCom“ in PerOpteryx bereits die Datenhaltung in EDP2 in sofern, dass die Experiment-Ergebnisse nach in einer EDP2 Repository-Struktur gespeichert werden. Allerdings konnten diese Ergebnisse bisher nicht ausgelesen werden und waren daher für die weitere Analyse unbrauchbar. Zudem wird die in Abschnitt 1.2 beschriebene, umfangreiche Speicherstruktur von EDP2 nicht ausgenutzt als direkte Folge der Abbildung vom SensorFramework in das EDP2. Daher kann man hier nur davon ausgehen, *dass* die Daten in einem EDP2 Repository gespeichert werden, allerdings nicht optimal.

[1] Result Visualization and Design Decision Support for the PCM, Diplomarbeit von Timo Rohrberg, 2010

Deswegen wurde in diesem Praktikum der Aufbau der Datenhaltung in EDP2 analysiert, damit die darin gespeicherten Daten korrekt ausgelesen werden können. Zudem wurde dies im SimuCom Projekt entsprechend implementiert, sodass EDP2 neben dem herkömmlichen Sensor Framework als alternatives Persistenz-Framework zur Verfügung steht.

3 Entwurf

Das relevante Projekt für den Einbau des EDP2 ist *de.uka.ipd.sdq.dsexplore.analysis.simucom*. In diesem Projekt werden die einzelnen, sich wiederholenden Simulationen angestossen und die Ergebnisse dieser wieder „eingesammelt“ zur Weiterverarbeitung.

Da die Ergebnisse wie in Schnitt 2 beschrieben bereits in das EDP2 Repository reingeschrieben werden von den Simulationen, müssen weder Simulationsanstoß noch die Simulation selbst verändert werden. Die Klasse *SimuComAnalysis* muss daher nur an wenigen Stellen verändert werden:

- Bei der Überprüfung **vor** einem Simulationsstart wird überprüft, ob das gewünschte Experiment bereits existiert. Falls dieses existiert, werden die gespeicherten Ergebnisse als *SimuComAnalysisResult* zurückgegeben. Bei alternativer Verwendung von Sensor Framework und EDP2 muss die Suche nach dem Projekt verändert werden, da die Speicherstruktur beider Persistenzframeworks sich unterscheiden.

- Für diverse Funktion wie der oben genannten müssen gespeicherte Simulationsergebnisse aus dem EDP2 geholt und als *SimuComAnalysisResult* zurückgegeben werden. Hier ist ebenfalls eine Änderung notwendig, da die Daten unterschiedlich ermittelt werden.

3.1 Entwurfsansätze für *SimuComAnalysisResult*

Die Klasse *SimuComAnalysisResult* enthält Such- und Verarbeitungsalgorithmen für Simulationsergebnisse, die nicht generisch funktionieren, das heißt unter anderem auch nicht direkt auf das EDP2 übertragbar sind. Daher muss diese Klasse modifiziert werden, möglich waren folgende Muster:

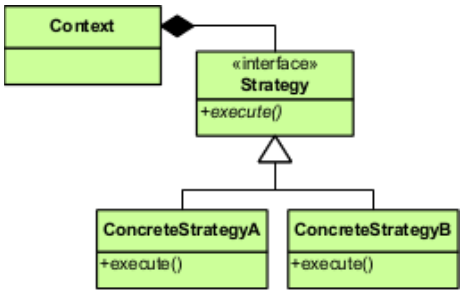

Abb. 3 : Strategie-Pattern[2]

1. Strategie
   Die Framework-spezifischen Teile von *SimuComAnalysisResult* werden ausgelagert nach dem Entwurfsmuster „Strategie“. Es wird ein Interface angelegt, welche die benötigten, spezifischen Funktionen definiert. Dieses wird dann jeweils für Sensor Framework und EDP2 implementiert. Die Vorteile des Entwurfsmusters, unter anderem die leichte Austauschbarkeit sowie Erweiterbarkeit werden mit dieser Lösung ermöglicht. Allerdings muss dieses neue Interface eingeführt werden und abgewägt werden, ob diese genannten

[2] Bildquelle: http://en.wikipedia.org/wiki/Strategy_pattern - Abruf am 25. April 2015

Vorteile in Zukunft von Bedeutung sein könnten, also ob in Zukunft weitere Alternativen parallel angeboten werden. Abb. 3 zeigt ein UML-Diagramm mit dem groben Strategie-Pattern. Der *Context* ist in diesem Fall die Klasse *SimuComAnalysisResult*, welche als Strategy-Interface ein Interface benutzt, das Zugriffe auf die Datenhaltung ermöglicht bzw. bestimmte Daten abruft. Die konkreten Strategien sind dann die auf Sensor Framework bzw. auf EDP2 spezialisierten Suchalgorithmen, welche die geforderten Daten ermitteln.

2. Vererbung

   Die zweite Möglichkeit ist es, direkt die ursprüngliche *SimuComAnalysisResult*-Klasse abstrakt zu deklarieren und zwei erbende Klassen zu erstellen, welche jeweils das Ergebnis aus EDP2 und aus dem Sensor Framework darstellt. Die Framework-spezifischen Teile der ursprünglichen Klasse werden zunächst in die Sensor Framework Unterklasse verschoben, da diese bereits auf den Sensor Framework zugeschnitten sind. Diese Teile müssen dann analog für EDP2 nachgebildet werden, indem die abgefragten Daten in der EDP2 Struktur ausfindig gemacht und extrahiert werden.
   Ursprünglich kann *SimuComAnalysis* ein bestimmtes Ergebnis-Set bereits selber suchen und daraus ein *SimuComAnalysisResult* erstellen. Da der Suchalgorithmus nun entsprechend dem Persistenzframework ausgewählt werden muss, wird der Teil der Suche in die entsprechende *SimuComAnalysisResult*-Unterklasse delegiert. Es wird eine neue, statische Funktion eingeführt in jeder der Unterklassen, die diese Suchaufgabe erfüllt und ein Objekt der jeweiligen Unterklasse erstellt, während in der aufrufenden Methode in *SimuComAnalysis* immer noch ein generisches *SimuComAnalysisResult*-Objekt erwartet.

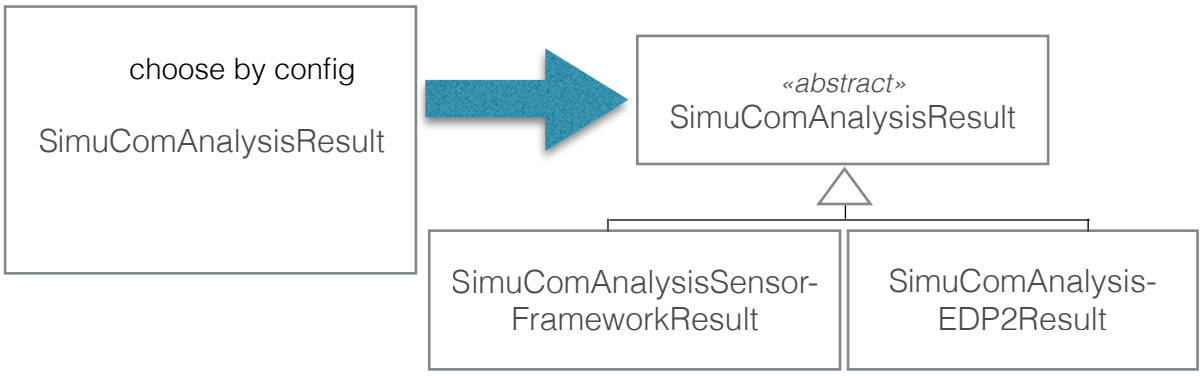

Abb. 4 : Entwurf mit Vererbung

Abb. 4 zeigt die Umwandlung der Klasse *SimuComAnalysisResult* und die Struktur inklusive der zwei neu eingeführten Unterklassen.

4 Implementierung

Die implementierte Änderung entspricht der in Abschnitt 3.1 Punkt 2 vorgestellte Vererbung. Dieser Entwurf wurde übernommen, da der Einsatz einer Strategie in diesem Fall nicht nötig war. Zudem sind sich die Entwurfsalternativen sehr ähnlich und die Unterklassen können leicht auch in konkrete Strategien konvertiert werden, sollte das in Zukunft nötig werden. Aus diesem Grund hat es an diesem Punkt keine signifikanten Vor- und Nachteile beider Entwurfsalternativen.

4.1 Änderungen in *SimuComAnalysis*

Wie bereits in den vorherigen Abschnitten beschrieben, muss auch in der Klasse *SimuComAnalysis* kleine Änderungen vorgenommen werden, da nun auch ein Teil der dort implementierten Funktionen an die Ergebnis-Klassen delegiert werden wie die Suche nach einem

bestimmten Datensatz. Ebenfalls muss zwischen Sensor Framework und EDP2 unterschieden werden, indem aus der Konfiguration gelesen wird.

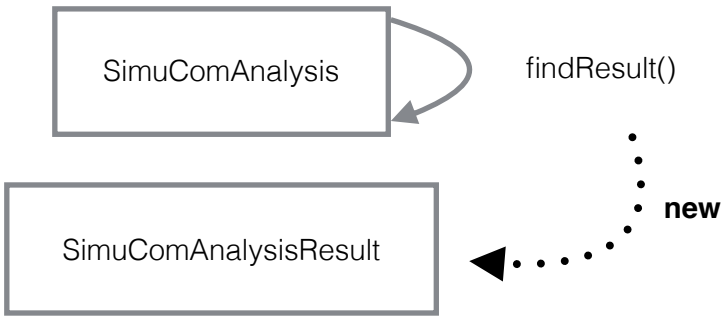

Abb. 5 : Erstellung eines Ergebnis-Objekts nach „altem“ Verfahren

Abb. 5 zeigt die bisherige Implementierung: In *SimuComAnalysis* wird eine Methode aufgerufen, die nach einem Ergebnis-Set sucht und dieses - falls vorhanden - zurückgibt. Sonst wird `null` zurückgegeben. Diese Suche wird nun delegiert, nachdem aus der Konfiguration das gewählte Framework ausgelesen wurde (siehe Abb. 6).

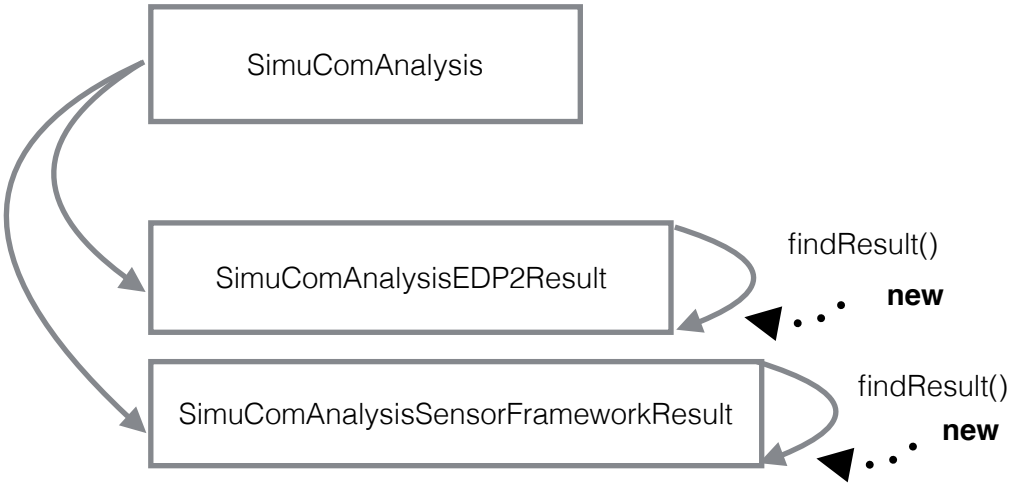

Abb. 6 : Erstellung eines Ergebnis-Objekts nach „neuem“ Verfahren

Nach dem „neuen“ Verfahren werden die in *SimuComAnalysis* ursprünglich durchgeführten Suchen und Checks als statische Methoden in den jeweiligen Ergebnis-Klassen implementiert. Die einzige, in der Klasse verbliebene Funktion ist der „Switch“ zwischen den Frameworks. Dieser wird bereits auf der Analysis-Ebene gemacht und die entsprechende Klasse angefragt, damit keine weitere Klasse noch zusätzlich auf die Konfiguration zugreifen muss.

Die betroffenen Methoden in *SimuComAnalysis* sind:

- `isExperimentRunDoesNotExist`
- `retrieveSimuComResults`

Die dazu „passenden“, statischen Methoden in den Klassen *SimuComAnalysisEDP2Result* und *SimuComAnalysisSensorFrameworkResult* sind:

- `isExperimentRunExisting`
- `findExperimentRunAndCreateResult`

4.2 Änderungen in *SimuComAnalysisResult*

Die Klasse *SimuComAnalysisResult* ist nun als `abstract` deklariert, da die spezifischen Teile an die Unterklassen delegiert werden. Einige (längere) Methoden delegieren dabei nur einen Abschnitt an die Unterklassen, damit möglichst viel gemeinsame Vorbereitung möglich ist. Zusätzlich wurden die Attribute, die von den Unterklassen genutzt oder beschrieben werden, auf `protected` statt `private` gesetzt.

Vollständig delegierte Methoden:

- `calculateUnivariateStatistic`
- `calculateThroughput`
- `getUsageScenarioMeasurements`
- `getLatestRun`
- `extractTimestamp`
- `retrieveUtilisationFromSensor`
- `determineConfidenceInterval`
- `getSensorForUsageScenario`
- `getSensorForResource`
- `measurementsToDoubleArray`
- `getUtilisationOfResource`

Teilweise delegierte Methoden:

- `retrieveServiceResults`
- `retrievePassiveResourceUtil`

Die teilweise delegierten Methoden haben jeweils eine passende, abstrakte Methode mit zugehörigem Namen, welche den Framework-spezifischen Rest der Methode in der Unterklasse implementiert. Diese sind bei EDP2 aus Komplexitätsgründen nicht implementiert, wie auch die Methode `getUtilisationOfResource`, was jedoch den Programmablauf im Allgemeinen nicht behindert.

Nicht alle der vollständig delegierten Methoden haben eine äquivalente Funktion in *SimuComAnalysisEDP2Result*, da die Ergebnisse bei EDP2 in einer unterschiedlichen Form vorliegen und z.B. nicht mehr zwingend zu einem `double`-Array konvertiert werden müssen.

4.3 *SimuComAnalysisEDP2Result*

Die Klasse *SimuComAnalysisEDP2Result* dient als EDP2-Pendant zu *SimuComAnalysisSensorFrameworkResult* und erbt ebenfalls von *SimuComAnalysisResult*. Hier werden die Framework-spezifischen Methoden neu implementiert, damit die gleiche Funktionalität angeboten werden kann unter Nutzung von EDP2.

Der wesentliche Unterschied zu den Sensor Framework Algorithmen liegt in der Weise, wie die Daten aus dem Persistenzframework entnommen werden. Aus diesen Datenpunkten können dann andere Größen wie das arithmetische Mittel oder der Median errechnet werden. Das Extrahieren der Datenpunkte geschieht in der Methode `getValuesForUsageScenario()`. Das ist direkt schon möglich, da der relevante *ExperimentRun* bereits bei Objekterstellung vorliegt. Dabei wird davon ausgegangen, dass jede *ExperimentGroup* nur ein *ExperimentSetting* enthält. Aus den *ExperimentRuns* wird dann der Runs gewählt, der am aktuellsten ist (vgl. `getLatestRun()` in *SimuComAnalysisSensorFrameworkResult*).

Die Methode `getValuesForUsageScenario()` liest die *Measurements* für das Usage Szenario aus. *Measurements* können allerdings mehrere *MeasurementRanges* haben. Auch hier ist derzeit nur ein *MeasurementRange* angelegt, welche die gesamte Messzeit enthält. In der

Speicherstruktur, in der SimuCom derzeit die Messergebnisse in das EDP2 ablegt, haben die Messspuren nur zwei *DataSeries*, nämlich für die Zeit der Messpunkte und für die gemessenen Werte. Für statistische Größen werden hier lediglich die Werte gebraucht.

4.4 Mögliche, zukünftige Änderungen

Einige Teile von *SimuComAnalysisEDP2Result* sind noch nicht vollständig. Dazu gehören die Methoden

`retrieveServiceResultsFinish()`,
`retrievePassiveResourceUtilFinish()` und
`getUtilisationOfResource()`.

Diese Methoden dienen zur präziseren Analyse der Auslastung der Ressourcen und wurde des Umfangs wegen noch nicht implementiert.

Zusätzlich müssen in allen Methoden die Abfragen angepasst werden, sollten Änderungen in der Speicherstruktur nach Abschnitt 5 Punkt 2 gemacht werden. Beispielsweise müssen dann konstante Zugriffe auf das erste Element in einer List („`.get(0)`“) durch eine Iteration über alle Elemente ausgetauscht - oder das „richtige“ Element vorher rausgesucht werden.

5 Ausblick

Abschließend werden hier noch einige Punkte aufgelistet, die vor oder während der Analyse des EDP2 entdeckt wurden und an denen noch weitere Arbeit geleistet werden kann, um die Einbindung von EDP2 in PerOpteryx zu optimieren.

1. Einbindung von Visualisierungen basierend auf EDP2
   Die Visualisierungen aus der Diplomarbeit von Timo Rohrberg[3] können mit wenig Aufwand eingebunden werden und damit erweiterte Möglichkeiten zur Analyse der Simulationsergebnisse bieten. Diese Visualisierungen basieren bereits auf EDP2 und müssen nicht mehr grundlegend angepasst werden.

2. Neustrukturierung der Speicherstruktur
   Die derzeitige Speicherstruktur der Experimentergebnisse sieht wie folgt aus: Eine Simulation wird als eine *ExperimentGroup* gespeichert mit jeweils einem *ExperimentSetting* sowie *ExperimentRun*. Dieser *ExperimentRun* enthält dann unterschiedliche Messreihen als *Measurements*. Wie in Abschnitt 1.2 beschrieben können allerdings mehrere *ExperimentSettings* einer *ExperimentGroup* zugewiesen werden wie auch mehrere *ExperimentRuns* zu einem *ExperimentSetting*.
   Durch das derartige Gruppieren der Ergebnisse können diese auf mehreren Leveln geordnet werden, was wiederum der Suche nach bestimmten Ergebnissen hinsichtlich der Performance zu Gute kommt.
   Zusätzlich können auch *Measurements* mehrere *MeasurementRanges* enthalten, sodass die Messzeit feiner unterteilt und untersucht werden kann. Zum Beispiel können Anfangs- und Endintervalle aus der Untersuchung ausgeschlossen werden, um Kaltstart- und Shutdown-Effekte raus zu filtern.

3. Umfangreichere Ausnutzung von EDP2 Funktionen
   Das EDP2 Framework enthält mehr Funktionen als das bisher genutzte Sensor Framework. Unter anderem können zum Beispiel statistische Werte direkt aus der Speicherstruktur und nicht mehr händisch aus den Messwerten berechnet werden. Um alle

[3] Result Visualization and Design Decision Support for the PCM, Diplomarbeit von Timo Rohrberg, 2010

Zusatzfunktionen des EDP2 sinnvoll zu nutzen, müssten diese noch genauer analysiert werden auf deren Nutzungsmöglichkeiten und entsprechend eingebunden werden. Allerdings müssen an den Stellen wie in dieser Arbeit implementiert ebenso Unterscheidungen zwischen Sensor Framework und EDP2 unternommen werden, falls Sensor Framework und EDP2 parallel angeboten werden sollen.