

Bachelorthesis

Konzeption und Entwicklung eines Auswertungstools für die Logdateien des JRockit Garbage Collectors

von

Raffael Schmid

Schule: Hochschule für Technik, Zürich
Betreuer: Mathias Bachmann
Experte: Marco Schaad
Zeitraum: 22. Juni 2011 - 22. Dezember 2011

Abstract

Die vorliegende Bachelorthesis befasst sich mit der Anforderungsanalyse, Konzeption und Implementation einer Software zur Analyse von Garbage Collection Logdateien der JRockit Virtual Machine.

Inhaltsverzeichnis

I	Projektbeschreibung	8
1	Aufgabenstellung	9
1.1	Ausgangslage	9
1.2	Ziele der Arbeit	9
1.3	Aufgabenstellung	9
1.4	Erwartete Resultate	10
1.5	Geplante Termine	10
II	Umsetzung	11
2	Analyse der Aufgabenstellung	12
2.1	Übersicht	12
2.1.1	Erarbeitung der Grundlagen	12
2.1.2	Stärken-Schwächen-Analyse verschiedener Rich Client Frameworks	13
2.1.3	Durchführung einer Anforderungsanalyse für den Software-Prototypen	13
2.1.4	Auswahl der zu verwendenden Frameworks	13
2.1.5	Konzeption und Implementation	13
2.1.6	Bewertung	13
3	Vorgehen Performanceanalyse	14
3.1	Motivation	14
3.2	Ablauf	14
3.3	Suche nach dem Dominating Consumer	15
3.3.1	Hohe relative Systemlast	15
3.3.2	Hohe CPU- respektive Core-Last	16
3.3.3	Effizienter Objekt-Lebenszyklus	16
3.4	Garbage Collection Tuning	17
3.4.1	Durchsatz	17
3.4.2	Pausenzeiten	18
3.4.3	Speicherverbrauch	18

4	Grundlagen Garbage Collection	19
4.1	Funktionsweise	19
4.1.1	Reference counting	20
4.1.2	Tracing techniques	20
4.2	Ziele der Garbage Collection	20
4.3	Eingliederung von Garbage Collection Algorithmen	21
4.3.1	Serielle versus Parallele Collection	21
4.3.2	Konkurrierend versus Stop-the-World	21
4.3.3	Kompaktierend, Kopierend	21
4.4	Grundlage der Algorithmen	21
4.4.1	Mark & Sweep Algorithmus	22
4.4.2	Mark & Copy Algorithmus[7]	22
4.4.3	Mark & Compact Algorithmus[6]	22
4.4.4	Tri-Coloring Mark and Sweep	22
4.5	Generational Garbage Collection	23
5	Grundlagen Garbage Collection in der JRockit Virtual Machine	25
5.1	Grundlage	25
5.1.1	Old Collection	25
5.1.2	Intergenerationale Referenzen	26
5.2	Übersicht der Garbage Collection Algorithmen auf der JRockit Virtual Machine	27
5.2.1	Garbage Collection Modi	27
5.2.2	Garbage Collection Algorithmen	28
5.3	Garbage Collection Logdateien	28
5.3.1	Aktivierung Log Ausgaben	29
5.3.2	Log Module	31
6	Anforderungsanalyse	32
6.1	Einleitung	32
6.1.1	Zweck	32
6.1.2	Systemumfang	32
6.1.3	Stakeholder	34
6.1.4	Glossar	35
6.1.5	Referenzen	35
6.1.6	Übersicht	35
6.2	Allgemeine Übersicht	35
6.2.1	Architekturbeschreibung	35
6.2.2	Nutzer und Zielgruppen	35
6.2.3	Randbedingungen	36
6.3	Funktionale Anforderungen	37
6.4	Qualitätsanforderungen	39
6.4.1	Software	39
6.4.2	Basisframework	40
6.5	Use Cases	42

6.5.1	Übersicht	42
6.5.2	Beschreibung	43
7	Auswahl Frameworks und Komponenten	53
7.1	Einleitung	53
7.2	Rich Client Framework	54
7.2.1	Übersicht	54
7.2.2	Auswertung Rich Client Frameworks	55
7.2.3	Entscheid	55
7.3	Charting Bibliothek	55
7.3.1	Übersicht	55
7.3.2	Auswertung Charting-Bibliothek	56
7.3.3	Entscheid	56
8	Konzept	57
8.1	Allgemein	57
8.1.1	Ausgangslage	57
8.1.2	Lizenzierung der Software	57
8.2	Architektur	57
8.2.1	Problemstellung	57
8.2.2	Übersicht	58
8.2.3	Projektstruktur	58
8.2.4	Ablauf Garbage Collection Analyse	60
8.3	Installation (UC-01)	62
8.4	Update (UC-02)	62
8.5	Datei importieren (UC-03)	62
8.6	Importierte Dateien speichern (UC-03.1)	63
8.7	Datei einlesen (UC-04)	63
8.7.1	Domänenmodell	63
8.8	Profil erstellen (UC-07)	64
8.8.1	Domänenmodell	64
8.8.2	Charts definieren (UC-07.1)	65
8.8.3	Profil speichern (UC-07.2)	65
8.8.4	Profil exportieren, importieren (UC-07.3/4)	65
8.8.5	Hilfesystem (UC-08)	66
8.9	Garbage Collection Logdatei parsen (FRQ-05)	66
8.9.1	Parser	66
8.9.2	MemoryModuleParser	68
8.9.3	Auswertung Logdatei	68
8.9.4	Domänenmodell JRockit Garbage Collection	69
8.10	Standardauswertung anzeigen (FRQ-06)	71
8.10.1	Anzeige Übersicht Garbage Collection (FRQ-06.1)	71
8.10.2	Chart Anzeige Heap Benutzung (FRQ-06.2)	72
8.10.3	Chart Anzeige Dauer Garbage Collection (FRQ-06.3)	72
8.11	Qualitätsanforderungen	72
8.11.1	Testabdeckung (QRQ-S-02)	72

<i>INHALTSVERZEICHNIS</i>	5
8.12 Eclipse Rich Client Framework	73
8.12.1 Speicherung des Zustands einer View	73
8.12.2 Internationalisierung (QRQ-S-03)	73
8.12.3 Usability (QRQ-S-04)	73
8.13 Infrastruktur	74
8.13.1 Build-Automatisierung	74
8.13.2 Versionskontrolle	74
8.13.3 Continous Integration	75
8.13.4 Issue Tracker	75
9 Implementation	76
9.1 Funktionale Anforderungen	76
9.1.1 Installation (FRQ-01) und Update (FRQ-02)	76
9.1.2 Datei importieren (FRQ-03)	76
9.1.3 Datei einlesen (FRQ-04)	76
9.1.4 Garbage Collection Logdatei parsen (FRQ-05)	77
9.1.5 Standardauswertung anzeigen (FRQ-06)	77
9.1.6 Profil erstellen (FRQ-07)	77
9.1.7 Hilfesystem (FRQ-08)	77
9.2 Qualitätsanforderungen Software	77
9.2.1 Erweiterbarkeit (QRQ-S-01)	77
9.2.2 Testabdeckung (QRQ-S-02)	77
9.2.3 Internationalisierung (QRQ-S-03)	77
9.2.4 Usability (QRQ-S-04)	78
9.2.5 Korrektheit (angezeigte Werte) (QRQ-S-05)	78
10 Review und Ausblick	79
10.1 Was leistet das Tool?	79
10.1.1 Offline-Analyse	79
10.1.2 Log Module	79
10.1.3 JRockit Version	79
10.2 Ausblick	80
10.2.1 Analyseumfangs JRockit R28	80
10.2.2 Analyseumfangs JRockit R27	80
10.2.3 G1 Algorithmus	80
Glossar	81
Abkürzungen	84
Listings	87
Abbildungsverzeichnis	88
Tabellenverzeichnis	89

III Anhang 90**A Bedienungsanleitung 91**

A.1	Installation der Software	91
A.2	Update	92
A.3	Dashboard	92
A.4	Import einer Garbage Collection Logdatei	93
A.5	Profile	94
A.5.1	Profil erstellen	95
A.5.2	Profil exportieren	95
A.5.3	Profil importieren	96
A.6	Garbage Collection Analyse	97
A.6.1	Standardauswertung	97
A.6.2	Benutzerdefinierte Auswertung (Profile)	100

B Informationen 102

B.1	Inhalt Datenträger	102
B.2	Repository	102

Einleitung

Bei der Performanceanalyse einer Software kann es notwendig sein, das Verhalten der Garbage Collection genauer zu betrachten. Diese Auswertung kann mit der Software JRocket Mission Control im laufenden Betrieb oder danach auf der Basis von resultierenden Logdateien gemacht werden. Im zweiten Fall gibt es allerdings - zumindest für die JRocket Virtual Machine (Release 28) keine Werkzeuge zur Automation dieser Aufgaben. Die vorliegende Bachelorthesis zeigt das Konzept und die Implementation einer Analysesoftware für Garbage Collection Logs der JRocket Virtual Machine. Die Software soll so erweiterbar sein, dass sie auch für andere Garbage Collection Algorithmen erweitert werden kann.

Die Bachelorthesis ist in drei Teile gegliedert: Teil eins definiert die von der Schulleitung abgenommene Aufgabenstellung, Teil zwei befasst sich mit der Umsetzung, von der Anforderungsanalyse bis zur Implementation. Im dritten Teil befindet sich der Anhang.

Teil I

Projektbeschreibung

Kapitel 1

Aufgabenstellung

1.1 Ausgangslage

Für die Ermittlung von Java Performance-Problemen braucht es Wissen über die Funktionsweise der Java Virtual Machine (JVM), deren Ressourcenverwaltung (Speicher, I/O, CPU) und das Betriebssystem. Die Verwendung von Tools zur automatisierten Auswertung der Daten kann in den meisten Fällen sehr hilfreich sein. Die Auswertung von Garbage Collection Metriken kann im laufenden Betrieb durch Profiling (online) gemacht werden, sie ist aber bei allen JVMs auch via Logdatei (offline) möglich. Die unterschiedlichen Charakteristiken der Garbage Collectors bedingen auch unterschiedliche Auswertungs- und Einstellungsparameter. JRockit ist die Virtual Machine des Weblogic Application Servers und basiert entsprechend auch auf anderen Garbage Collector Algorithmen als die der Sun VM. Aktuell gibt es noch kein Tool, welches die Daten der Logs sammelt und grafisch darstellt.

1.2 Ziele der Arbeit

Ziel der Bachelorthesis ist die Konzeption und Entwicklung eines Prototypen für die Analyse von Garbage Collection Logdateien der JRockit Virtual Machine. Die Software wird mittels einer Java Rich Client Technologie implementiert. Zur Konzeption werden die theoretischen Grundlagen der Garbage Collection im Allgemeinen und der JRockit Virtual Machine spezifisch erarbeitet und zusammengestellt.

1.3 Aufgabenstellung

Im Rahmen der Bachelorthesis werden vom Studenten folgende Aufgaben durchgeführt:

1. Studie der Theoretischen Grundlage im Bereich der Garbage Collection (generell und spezifisch JRockit Virtual Machine)
2. Stärken- / Schwächen-Analyse der bestehende Rich Client Frameworks (Eclipse RCP Version 3/4, Netbeans)
3. Durchführung einer Anforderungsanalyse für einen Software-Prototyp.
4. Auswahl der zu verwendenden Frameworks
5. Konzeption und Spezifikation des Software-Prototypen (auf Basis des ausgewählten Rich Client Frameworks), der die ermittelten Anforderungen erfüllt.
6. Implementation der Software
7. Bewertung der Software auf Basis der Anforderungen

1.4 Erwartete Resultate

Die erwarteten Resultate dieser Bachelorthesis sind:

1. Detaillierte Beschreibung der Garbage Collection Algorithmen der Java Virtual Machine im Generellen und spezifisch der JRockit Virtual Machine.
2. Analyse über Stärken und Schwächen der bestehenden (state of the art) Java Rich Client Technologien
3. Anforderungsanalyse des Software Prototyps
4. Dokumentierte Auswahlkriterien und Entscheidungsgrundlagen
5. Konzept und Spezifikation der Software
6. Lauffähige, installierbare Software und Source-Code
7. Dokumentierte Bewertung der Implementation

1.5 Geplante Termine

Kick-Off:	Juni 2011
Design Review:	August 2011
Abschluss-Präsentation:	Ende November 2011

Teil II

Umsetzung

Kapitel 2

Analyse der Aufgabenstellung

2.1 Übersicht

Das Ziel dieser Arbeit ist die Konzeption und Implementation einer Software für die Analyse von Garbage Collection Logdateien der JRockit Virtual Machine. Deren grundlegendes Ziel ist es, die grosse Datenmenge schnell und übersichtlich darzustellen. Um die Anforderungen an diese Software zu ermitteln, braucht es im Bereich der Performanceanalyse, der Grundlage der Garbage Collection und über die JRockit VM einige zu erarbeitende Grundlagen. Anschliessend wird eine Stärken-/Schwächen-Analyse der bestehenden Richt Client Frameworks gemacht, aufgrund welcher nach der Aufnahme der Anforderungen der Entscheid für das zu verwendende Framework getroffen werden kann. Im Anschluss an diese Tätigkeiten folgt die Konzeption und Implementation der Software. Die genaueren Beschreibung der einzelnen Teilaufgaben befindet sich in den folgenden Abschnitten.

2.1.1 Erarbeitung der Grundlagen

Die Erarbeitung der Grundlagen dient als Basis in zwei Bereichen:

- **Anforderungsanalyse:** Aus Sicht des Analysten respektive dem Benutzer dieser Software ist es essentiell, dass er die richtigen Daten der Garbage Collection, die richtige Sicht auf diese Daten und die richtigen Filter-Funktionen vorfindet. Voraussetzung für diese Anforderungen sind die Kenntnisse der verschiedenen Garbage Collection Algorithmen insbesondere die der JRockit VM.
- **Konzept:** Die Konzeption des Domänen-Modells und des Parseprozesses der Logdateien bedingt eine genaue Kenntnis der verschiedenen Strategien und Ausgabeformate.

2.1.2 Stärken-Schwächen-Analyse verschiedener Rich Client Frameworks

Aufgrund einiger Anforderungen (siehe Anforderungsanalyse) muss die Applikation als Rich Client implementiert werden. Als Basis kommen die Frameworks Netbeans und Eclipse in Frage. Die Stärken-Schwächen-Analyse soll den Entscheid für die eine dieser Plattformen herleiten.

2.1.3 Durchführung einer Anforderungsanalyse für den Software-Prototypen

Die Anforderungen werden zusammen mit einem Performance-Analysten ermittelt und nach [12, 4.3.2 Angepasste Standardinhalte] dokumentiert. Laut dem IEEE¹ und [12, 4.5 Qualitätskriterien für das Anforderungsdokument] müssen Anforderungen folgende Kriterien erfüllen:

- Eindeutigkeit und Konsistenz
- Klare Struktur
- Modifizierbarkeit und Erweiterbarkeit
- Vollständigkeit
- Verfolgbarkeit

2.1.4 Auswahl der zu verwendenden Frameworks

Basierend auf der Stärken-Schwächen-Analyse wird ein Entscheid für das jeweilige Framework gewählt. Nebst der Auswahl der Rich Client Plattform muss auch ein Entscheid hinsichtlich Charting-Bibliothek gefällt werden.

2.1.5 Konzeption und Implementation

Basierend auf den Anforderungen wird das Konzept erstellt und anschliessend die Software implementiert.

2.1.6 Bewertung

Die Bewertung der Software wird auf Basis der Anforderungen gemacht.

¹Institute of Electrical and Electronic Engineers

Kapitel 3

Vorgehen Performanceanalyse

3.1 Motivation

Die Performance respektive Leistungsfähigkeit einer Applikation stellt einer der bedeutendsten Qualitätsanforderungen dar. Sie kann nicht nur zu verärgerten Benutzern, sondern auch zu einer Verlangsamung der Business-Prozesse führen. Die Motivation für eine Performanceanalyse entsteht, wenn die in Service Level Agreement oder Anforderungsanalyse spezifizierten **Qualitätsanforderungen nicht erfüllt** sind. Optimal wäre, wenn diese Anforderung während des Entwicklungsprozesses kontinuierlich geprüft würde.

3.2 Ablauf

Die Performanceanalyse ist ein iterativer Prozess und sollte so lange dauern, bis die Anforderungen erfüllt und der Kunde zufrieden ist. Sie besteht aus folgenden vier Schritten[3]:

1. Identifikation der neuralgischen Punkte des Systems
2. **Suche nach dem Dominating Consumer**¹
3. **Sammeln von Detaildaten**
4. Lösen des Problems

Damit die Analyse der Garbage Collection zum richtigen Zeitpunkt gemacht wird, sind insbesondere die Punkte zwei und drei wichtig. Sie werden in den folgenden beiden Abschnitten genauer definiert.

¹Kirk Pepperdine bezeichnet die Aktivität, welche dominiert wie die CPU gebraucht wird, als Dominating Consumer .

3.3 Suche nach dem Dominating Consumer

Die Suche nach dem Dominating Consumer kann nach folgendem Schema durchgeführt werden:

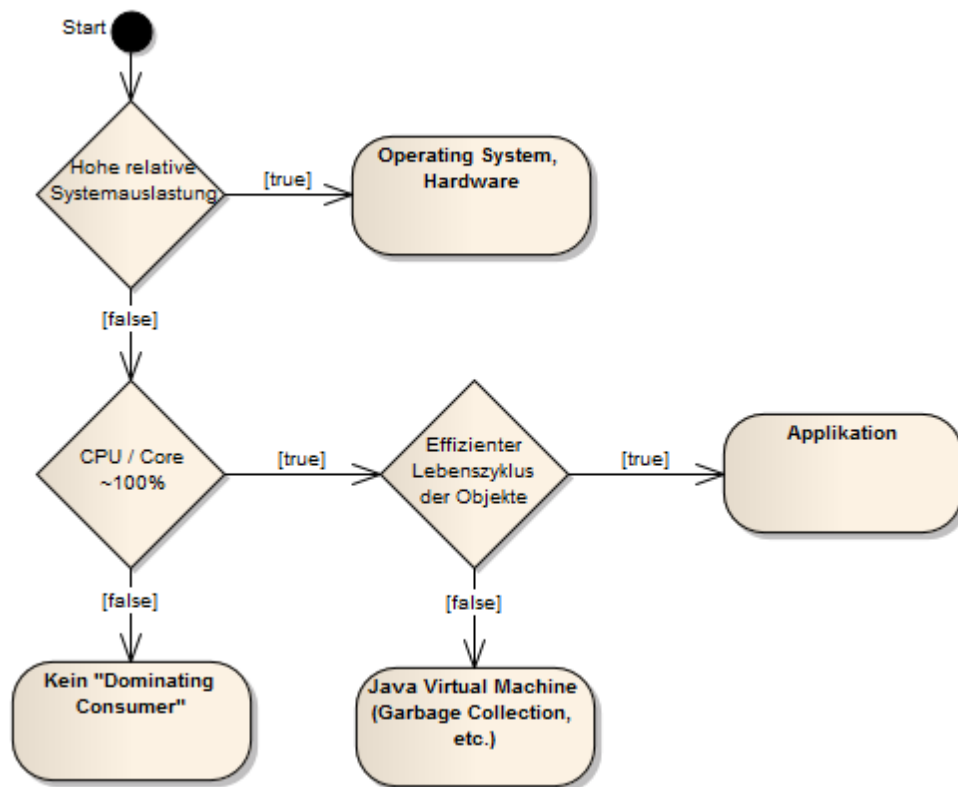


Abbildung 3.1: Suche nach dem Dominating Consumer

3.3.1 Hohe relative Systemlast

Die erste Frage, die sich bei der Suche nach dem Dominating Consumer stellt, ist, ob die hohe CPU-Auslastung² durch die Applikation oder das System verursacht wird. Je nach Betriebssystemen gibt es dafür unterschiedliche Werkzeuge:

- **Windows:** Task Manager / Task Performance (Leistung). Für die Anzeige der Systemzeit muss die Anzeige der Kernelzeit eingeblendet werden: View / Show Kernel Times
- **Linux / Unix:** vmstat zeigt die Ausnutzung der CPU: vmstat 5

²CPU kann auch einer oder ein paar Cores bedeuten.

Hohe Systemauslastung kann unterschiedliche Gründe haben:

- **Exzessives Context-Switching**³: Mehrere Prozesse können sich die gleiche CPU (CPU-Kern) nur deshalb teilen, weil beim Wechsel von Prozess A auf B ein Context-Switch gemacht wird, so dass Thread B am selben Ort weiterarbeitet wo er beim letzten Mal war. Gründe für exzessives Context-Switching können beispielsweise nicht adäquat gewählte Locks sein. Wenn ein Thread aufgrund der Synchronisation angehalten werden muss.⁴
- **Hohe I/O Festplatte**
- **Hohe I/O Netzwerk**

3.3.2 Hohe CPU- respektive Core-Last

Sofern die im Abschnitt 3.3.1 beschriebenen Punkte nicht zutreffen, stellt sich die Frage, ob die Auslastung der CPU überhaupt gross ist. Ist das nicht der Fall, gibt es womöglich gar keinen Dominating Consumer - es muss dann herausgefunden werden, was die Threads weg von der CPU haltet. Es gibt dafür laut [11] unterschiedliche Gründe:

- **Dead Locks**: Threads schliessen sich gegenseitig aus.
- **Applikation skaliert nicht**: Applikation hat schlechte multi-core Eigenschaften.
- **Langsame Disks / Netzwerke**
- **Zu kleine Connection- und Thread-Pools**
- **Aurufe auf langsame externe Systeme**

3.3.3 Effizienter Objekt-Lebenszyklus

Sofern die relative Systemlast klein und die CPU-Auslastung gross ist, muss der Lebenszyklus der Objekte angeschaut werden. Dies kann auf unterschiedliche Arten getan werden:

- **Memory Analyse**: Es wird angeschaut, wie alt die Objekte in den verschiedenen Bereiche (Young Collection, Old Collection) sind. Wieviele Garbage Collection Zyklen sie überlebt haben.
- **Garbage Collection**: Wann und wie oft werden Garbage Collections durchgeführt, wie lange haben sie gedauert, wieviel Speicher haben sie befreit.

³Kontextwechsel

⁴Symptomatisch zeigt dann vmstat, das System ist nicht im Leerlauf, aber die Kernelzeiten (cpu/sy) dominieren die Zeit des Benutzers (cpu/us).

3.4 Garbage Collection Tuning

Mit der Analyse des Dominating Consumers entscheidet sich auch, in welchem Bereich anschliessend weitere Detaildaten gesammelt werden. Auch beim Garbage Collection Tuning gilt grundsätzlich: "never change a running system". Man ändert also nur etwas, wenn es Probleme bei der Garbage Collection respektive die Anforderungen daran nicht erfüllt sind. Mit dem Tuning will man in der Regel drei unterschiedliche Ziele erreichen[8]:

- Verbesserung des Durchsatzes (siehe Abschnitt 3.4.1)
- Geringere Pausenzeiten (siehe Abschnitt 3.4.2)
- Geringerer Speicherverbrauch (siehe Abschnitt 3.4.3)

Diese drei Ziele bilden eine Dreiecksbeziehung, zum Beispiel führt eine aggressive Optimierung des Durchsatzes in der Regel zu längeren Stop-the-World⁵ Pausen. Tuning bedeutet nun, zwei der Ziele konstant halten, während das dritte durch Anpassung der Konfiguration verbessert wird. Sofern die Optimierung nicht ausreicht, müssen einer oder beide anderen Parameter aufgegeben werden, um das Ziel zu erreichen.

3.4.1 Durchsatz

Die relative Zeit der CPU welcher der Anwendung zur Verfügung steht nennt man Durchsatz (englisch *Throughput*). Es handelt sich also um einen Prozentsatz im Vergleich mit der gesamten CPU-Zeit. Grundsätzlich ist wird der Durchsatz folgendermassen ausgerechnet:

$$\begin{aligned} \text{Durchsatz} &= 100 * (1 - \text{Relative Zeit in Garbage Collection}) \\ \text{Durchsatz} &= 100 * (1 - \frac{\text{Zeit in Garbage Collection}}{\text{Gesamtdauer der Messung}}) \end{aligned}$$

Im Prinzip ist diese Formel ziemlich ungenau und berücksichtigt nur die Garbage Collection Zeiten während einer Stop-the-World Pause. Insofern ist sie nur für eine Single-Core CPU genau. Bei einer Multi-Core-Maschine müsste der Througput für jeden einzelnen Core ausgewertet und damit der Gesamt-Throughput berechnet werden.

Durchsatz-Tuning spielt in der Praxis oft nur eine untergeordnete Rolle[8]. Eine wirkliche Steigerung des Durchsatzes kann man nur mit einer aggressiven Optimierung erzielen, sie führt zu verlängerten Pausenzeiten, was nur bei Batch-Applikationen toleriert werden kann. Gleichzeitig ist der Austausch oder das Zuschalten von weiteren CPUs sehr rasch der kostengünstigere Weg.

⁵Zeiten in welchen die Prozessoren der Anwendung keine Rechenzeiten zur Verfügung stellen.

3.4.2 Pausenzeiten

In der Regel geht es beim Garbage Collection Tuning um das Tuning der Pausenzeiten, dies weil man die Symptome wie Stottern und einer ungleich reaktionsfreudigen Anwendung beseitigen möchte.

In der Regel startet man mit der Analyse einzelner Garbage Collections. Die Gründe für die Initiierung einer Garbage Collection können sehr unterschiedlich sein und müssen entsprechend behandelt werden:

- **Erreichen eines Schwellenwerts des Young- oder Old-Spaces:** Anpassung der Grösse des Young- / Old-Spaces, Vergrösserung des Heaps, etc.
- **Heap hat maximale Grösse erreicht:** Anpassung der Heap-Grösse, Beseitigung von Memory Leaks, etc.
- **Allokation von Objekten ist nicht möglich:** Vergrösserung des Young-Space, Beseitigung der Fragmentierung, etc.
- **Von der Applikation innitiert (`System.gc()`):** Von der Applikation ausgelöste Garbage Collections können mittels einem Flag ignoriert werden.

3.4.3 Speicherverbrauch

Dieses Tuningziel verliert insofern an relevanz, weil 64-Bit-JVMs bei grossen Applikationen schon sehr verbreitet sind. Mit den 32-Bit Adressen kam man im Bereich des Arbeitsspeichers schon eher an die Grenze (ein Teil des adressierbaren Bereichs von 4GB bei 32-Bit-Adressen wird auch noch Für das Betriebssystem verwendet). Eine Optimierung im Bereich des Speicherverbrauchs hat in der Regel nichts mit Garbage Collection Tuning zu tun, sondern ist eine Analyse der Objektpopulation im Bereich der Applikation.

Kapitel 4

Grundlagen Garbage Collection

Schon seit den ersten Programmiersprachen ist das Aufräumen von verwendeten Ressourcen und Speicher ein wichtiges Thema. Im Unterschied zu den ersten Sprachen, bei denen das Memory Management in der Verantwortung des Entwicklers war (explizit), findet allerdings das Recycling von Memory bei Sprachen der dritter Generation automatisch statt und macht Operatoren wie “free” unwichtig. Bei Formen dieser automatischen Speicherverwaltung spricht man von Garbage Collection¹. In den meisten neueren Laufzeitumgebungen gibt es zusätzlich das Prinzip des adaptiven Memory Managements, hier werden Feedbacks und Heuristiken dazu verwendet, um die Strategie der Garbage Collection anzupassen. Probleme die nur beim expliziten Memory Management auftreten sind Dangling References² (Dangling Pointers) und Space Leaks³. Trotzdem sind Memory Leaks auch bei automatischer Speicherverwaltung noch möglich, nämlich dann wenn Memory noch referenziert ist, auch wenn es nicht mehr gebraucht wird. Der folgende Abschnitt beschreibt die Grundlagen der Java Garbage Collection. Das nächste Kapitel zeigt dann die Details dazu spezifisch für die JRockit Virtual Machine.

4.1 Funktionsweise

Alle Techniken der Garbage Collection zielen darauf ab, die “lebenden” von den “toten” Objekten zu unterscheiden. Es werden also primär die Objekte gesucht, die nicht mehr referenziert sind. Laut [4, S. 77] gibt es grundsätzlich zwei unterschiedliche Strategien: Referenz-Zählung (Reference Counting) und

¹vom englischen Wort “garbage collector” für Müll-, Abfallsammler

²Man spricht von Dangling Pointers oder Dangling References, wenn ein Pointer auf ein Objekt im Memory freigegeben wurde, obwohl es noch gebraucht wird.

³Man spricht von Space Leaks, wenn Memory alloziert und nicht mehr freigegeben wurde, obwohl es nicht mehr gebraucht wird.[9]

Tracing-Techniken (Tracing Techniques).

4.1.1 Reference counting

Beim Reference counting[4, S. 77] behält die Laufzeitumgebung jederzeit den Überblick, wie viele Referenzen auf jedes Objekt zeigen. Sobald die Anzahl dieser Referenzen auf 0 gesunken ist, wird das Objekt für die Garbage Collection freigegeben. Trotzdem der Algorithmus relativ effizient ist, wird er aufgrund der folgenden Nachteile nicht mehr verwendet:

- Sofern zwei Objekte einander referenzieren (zyklische Referenz), wird die Anzahl Referenzen nie 0.
- Es ist relativ aufwendig, die Anzahl Referenzen immer auf dem aktuellsten Stand zu halten, insbesondere in nebenläufigen (concurrent) Systemen.

4.1.2 Tracing techniques

Bei den Tracing-Techniken[4, S. 77] werden vor jeder Garbage Collection die Objekte gesucht, auf welche aktuell noch eine Referenz zeigt. Die anderen werden zur Garbage Collection freigegeben. Diese Art von Garbage Collection Algorithmen verwenden ein Set (Root Set), bestehend aus von globalen Variablen und den Stack aller Threads referenzierten Objekten, als Startpunkt für die Traversierung aller Objekte.

4.2 Ziele der Garbage Collection

Für Garbage Collectors gibt es folgende Bedingungen[9, S. 4]:

- **Sicherheit:** Garbage Collectors dürfen nur Speicher/Objekte freigeben, der effektiv nicht mehr gebraucht wird,
- **Umfassend:** Garbage Collectors müssen Speicher/Objekte die nicht mehr gebraucht werden nach wenigen Garbage Collection Zyklen freigegeben haben.

Wünschenswert sind zudem folgende Punkte[9, S. 4]:

- **Effizienz:** Die Anwendung soll vom laufenden Garbage Collector möglichst nicht beeinträchtigt sein:
 - keine langen Pausen⁴
 - einen durch den Garbage Collector möglichst geringen Ressourcenverbrauch

⁴man spricht von Stop-the-World wenn zwecks Garbage Collection die Anwendung gestoppt wird und ihr damit keine Ressourcen zur Verfügung stehen

- **Wenig Fragmentierung:** Bei starker Fragmentierung ist die Allokation von Speicher nicht mehr effizient möglich. Wenn der zu allozierende Speicher für ein Objekt grösser ist als die grösste Speicherlücke, kommt es zu einem Out-of-Memory-Error, obwohl insgesamt noch genügend Speicher vorhanden ist.

4.3 Eingliederung von Garbage Collection Algorithmen

Bei der Selektion von Garbage Collection Algorithmen gibt es entscheidende Kriterien[9, S. 5]:

4.3.1 Serielle versus Parallele Collection

Auf einem Multi-Core System kann die Garbage Collection parallelisiert werden. Dies bringt zwar einen Overhead, wirkt sich aber in der Regel trotzdem in verkürzten Garbage Collection Pausen aus.

4.3.2 Konkurrierend versus Stop-the-World

Wenn aufgrund der Garbage Collection der Heap der Laufzeitumgebung blockiert (freeze) werden muss, führt das implizit zum Stopp (Stop-the-World) der Anwendung. Diese Pausen sind beispielsweise für Webapplikationen sehr unerwünscht, können aber für Backendprozesse durchaus toleriert werden.

4.3.3 Kompaktierend, Kopierend

Fragmentierung tritt grundsätzlich beim befreien von Speicher auf und ist ein unerwünschter Nebeneffekt, da sie zur Verlangsamung der Speicherallokation führt. Sie kann durch das Kompaktieren der überlebenden Objekte oder das Kopieren in einen anderen Bereich minimiert werden.

4.4 Grundlage der Algorithmen

Der Prozess der Garbage Collection beginnt bei allen Algorithmen mit der Marking-Phase. Für jedes Objekt des Root Sets⁵ werden rekursiv die transitiv abhängigen Objekte auf dem Heap bestimmt. Alle nicht im Resultat enthaltenen Objekte sind tot und können bereinigt werden.

⁵Objekte im Heap die aus den Call-Stacks der aktuellen Threads referenziert werden, globale ("static final" definierte) Variablen mit Referenzen auf den Heap)

4.4.1 Mark & Sweep Algorithmus

Beim Mark & Sweep Algorithmus[7] wird nach der oben beschriebenen Mark-Phase der Speicher der nicht mehr referenzierten Objekte freigegeben. In den meisten Implementationen bedeutet dies das Einfügen dieses Objekts in eine sogenannte Free-List.

Nachteil dieses Algorithmus

Der Speicher wird aufgrund der vielen Freigaben stark fragmentiert.

4.4.2 Mark & Copy Algorithmus[7]

Mark & Copy ist ein Algorithmus, bei dem der resultierende Heap nach der Collection nicht fragmentiert ist. Der Heap wird in einen “from space” und einen “to space” unterteilt. Objekte die noch immer am Leben sind, werden im Anschluss an die Markierung vom “from space” in den “to space” kopiert.

4.4.3 Mark & Compact Algorithmus[6]

Für die Old Collection kann es Sinn machen, einen Mark & Compact Algorithmus zu verwenden (siehe Generational Garbage Collection). Nach der Markierungs-Phase und der Bereinigung der “toten” Objekte wird der Speicher kompaktiert. Der Algorithmus hat zwar einen erhöhten Ressourcenbedarf, es wird aber im Gegensatz zum Mark & Copy kein Speicher verschwendet.

4.4.4 Tri-Coloring Mark and Sweep

Eine Variation des Mark & Sweep Algorithmus ist der Tri-Coloring Mark & Sweep Algorithmus. Er lässt sich besser parallelisieren[4, S. 79]. Im Gegensatz zur normalen Version des Mark & Sweep Algorithmus wird anstelle eines Mark-Flags ein ternärer Wert genommen der den Wert “weiss”, “grau” und “schwarz” annehmen kann. Der Status der Objekte wird in drei Sets nachgeführt. Das Ziel des Algorithmus ist es, alle weissen Objekte zu finden. Schwarze Objekte sind die, die garantiert keine weissen Objekte referenzieren und die Grauen sind die, bei denen noch nicht bekannt ist, was sie referenzieren. Der Algorithmus funktioniert folgendermassen:

1. Als erstes haben alle Objekte das Flag weiss.
2. Die Objekte des Root-Sets werden grau markiert.
3. Solange es graue Objekte hat, werden rekursiv die Nachfolger dieser Objekte grau markiert.
4. Sobald alle Nachfolger des Objekts grau markiert sind, wird das aktuelle Objekt auf den Status schwarz geändert.

Der Vorteil des Tri-Coloring Mark & Sweep basiert auf folgender Invariante: **Kein schwarzes Objekt zeigt jemals direkt auf ein Weisses.** Sobald es keine grauen Objekte mehr gibt - sprich das Set der grauen Objekte leer ist, können die weissen Objekte gelöscht werden.

4.5 Generational Garbage Collection

Innerhalb einer Anwendung kann man die Objekte in Short Living, Medium Living und Long Living Objekte⁶ kategorisieren. Es gibt viele Objekte die nicht lange leben (Objekte mit der Lebensdauer einer Methode) und wenig Objekte mit langer Lebensdauer, wie Teile eines Caches. Aus diesem Grund ist es sinnvoll, die Garbage Collection Strategie der Lebensdauer der Objekte anzupassen. Der Speicher je nach Virtual Machine in unterschiedliche Bereiche aufgeteilt.

Die HotSpot Virtual Machine tut dies folgendermassen[5]:

- **Young Collection:** Die Young Collection ist nochmals unterteilt in Eden Space (Bereich für neue Objekte) und zwei Survivor Spaces (from-space als aktiver Bereich, to-space als der Bereich in den die Objekte während der Collection kopiert werden).
- **Old Collection:** In der Old Collection befinden sich die Objekte, die eine gewisse Anzahl an Young-Generation Collections (Minor Collections) überlebt haben und dann durch eine Promotion hierhin verschoben wurden.
- **PermGen:** Der PermGen Bereich ist keine Generation, sondern ein Non-Heap-Bereich, und wird von der VM für eigene Zwecke verwendet. Dieser Bereich ist eine Eigenheit der Oracle HotSpot Virtual Machine. Hier werden beispielsweise Class-Objekte inklusive Bytecode für alle geladenen Klassen und JIT-Informationen⁷. gespeichert.

Und bei der JRockit Virtual Machine ist es so:

- **Nursery**⁸: Die Nursery entspricht der Young Collection bei der HotSpot Virtual Machine und ist der Bereich der jungen Objekte. Bei JRockit ist es möglich, eine zusätzliche Keep-Area innerhalb der Nursery zu haben. Diese Keep-Area ist der Platz der Objekte mittlerer Lebensdauer. Ein Objekt wird also zuerst von der Nursery in die Keep-Area promoted - was den Vorteil hat, dass sie, wenn nicht mehr referenziert, immer noch einer Young Collection unterliegen - erst dann gelangen sie nach einer nochmaligen Young Collection in die Old Collection.
- **Old Generation:** Die Old Generation beinhaltet wie bei der HotSpot Virtual Machine die Objekte mit langer Lebensdauer.

⁶Objekte mit kurzer, mittel und langer Lebensdauer.

⁷Die Just-in-Time (Kompilierung) führt zu einer Veränderung respektive Optimierung des Bytecodes.

⁸Nursery bedeutet im übertragenen Sinn Kindergarten

Für weitere Details zu den JRokit Garbage Collection Eigenheiten siehe Kapitel Grundlagen Garbage Collection in der JRokit Virtual Machine.

Kapitel 5

Grundlagen Garbage Collection in der JRockit Virtual Machine

5.1 Grundlage

Die Grundlage der JRockit Garbage Collection bildet der Tri-Coloring Mark & Sweep Algorithmus (siehe Abschnitt Tri-Coloring Mark and Sweep). Er wurde hinsichtlich besserer Parallelisierbarkeit und der optimalen Verwendung der Anzahl Threads optimiert. Die Garbage Collection der JRockit VM kann mit oder ohne Generationen arbeiten - so gibt es die beiden Algorithmen “Concurrent Mark & Sweep” und “Parallel Mark & Sweep” in beiden Ausführungen Generational und Single:

- Generational Concurrent Mark & Sweep
- Single Concurrent Mark & Sweep
- Generational Parallel Mark & Sweep
- Single Parallel Mark & Sweep

5.1.1 Old Collection

Im Unterschied zum normalen Tri-Coloring Algorithmus verwendet der Algorithmus der JRockit, egal ob parallel oder concurrent, zwei Sets für die Markierung der Objekte. In einem werden die grauen und schwarzen Objekte gespeichert, im anderen die weissen. Die Trennung zwischen Grau und Schwarz wird gemacht indem die grauen Objekte in Thread-Local Queues jedes Garbage Collection Threads gespeichert werden.

Die Verwendung von Thread-lokalem Speicher hat hinsichtlich den folgenden Punkten einen Vorteil:[4, S. 79]:

- Thread-lokaler Speicher führt zu einer besseren Parallelisierbarkeit.
- Thread-lokaler Speicher kann prefetched werden, was die Geschwindigkeit des Algorithmus als Ganzes erhöht.

Bei der Verwendung der Concurrent Algorithmen wird parallel dazu auch Speicher für neue Objekte alloziert. Diese neuen Objekte werden in einem sogenannten Live-Set getrackt, damit sie in der Sweep Phase - obwohl nicht markiert - nicht gelöscht werden.

5.1.2 Intergenerationale Referenzen

Die Idee der Generational Garbage Collection ist, dass während einer Young Collection nur ein Teilbereich des Heaps aufgeräumt wird. Referenzen in die Old Collection werden dabei nicht verfolgt. Den Referenzen von der Old in die Young Collection muss trotzdem nachgegangen werden, sie können auf unterschiedliche Weise entstehen:

- **Ein Objekt wird durch eine Promotion von der Young Generation in die Old Generation verschoben:** Die Dabei entstehenden intergenerationellen Referenzen werden im sogenannten **Remembered Set** nachgeführt.
- **Es findet eine Zuweisung der Referenz durch die Applikation statt:** Diese Problematik wird durch Write Barriers und der Aufteilung des Heaps in sogenannte Cards¹ gemacht. Bei der Änderung einer Referenz (die dann von der Old in die Young Collection zeigt) wird auf der entsprechenden Card das "Dirty Bit" gesetzt. Objekte innerhalb von dirty Cards werden nach der Markierungsphase nochmals überprüft.

Concurrent Mark & Sweep

Beim Concurrent Mark & Sweep handelt es sich eigentlich um einen Mostly Concurrent Mark & Sweep Algorithmus. Das heisst er findet nicht in allen Phasen konkurrierend zur Applikation statt. Die Markierung dieses Algorithmus ist in vier Phasen aufgeteilt:

- **Initial Marking (Nicht konkurrierend):** Hier wird das Root Set zusammengestellt.
- **Concurrent Marking (konkurrierend):** Mehrere Threads gehen nun diesen Referenzen nach und markieren Sie als lebende Objekte.
- **Preclean (konkurrierend):** Änderungen im Heap während den vorherigen Schritten werden nachgeführt, markiert.
- **Final Marking (Nicht konkurrierend):** Änderungen im Heap während der Precleaning Phase werden nachgeführt, markiert.

¹Eine Card ist typischerweise ein ungefähr 512 Byte grosser Bereich auf dem Heap.

Die Sweep Phase findet ebenfalls konkurrierend zur Applikation statt. Im Gegensatz zur HotSpot VM ist sie aber in zwei Schritte aufgeteilt. Als erstes wird die erste Hälfte des Heaps von toten Objekten befreit. Während dieser Phase können Threads Speicher in der zweiten Hälfte des Heaps allozieren. Nach einer kurzen Synchronisationspause findet das Sweeping auf dem zweiten Teil des Heaps statt.

Parallel Mark & Sweep

Beim parallelen Mark & Sweep findet die Garbage Collection parallel mit allen verfügbaren Prozessoren statt. Dazu werden aber alle Threads der Applikation gestoppt.

Kompaktierung

Aufgrund der im Speicher stattfindenden Fragmentierung wird bei jeder Old Collection eine Kompaktierung des Speichers durchgeführt. Während der Sweep Phase werden die Objekte auf dem Heap umherkopiert, so dass danach wieder grössere Speicherbereiche am Stück existieren.

5.2 Übersicht der Garbage Collection Algorithmen auf der JRockit Virtual Machine

5.2.1 Garbage Collection Modi

Die Auswahl der Garbage Collection Strategie muss auf der Basis der Anforderungen an die Applikation gemacht werden. Auf der JRockit VM gibt es drei verschiedene Modi:

- **Durchsatz (throughput):** Optimiert die Garbage Collection hinsichtlich möglichst grossem Durchsatz. Um dieses Ziel zu erreichen, werden parallele Algorithmen verwendet. Sie laufen nicht-konkurrierend mit der Applikation und führen zu kurzen Pausenzeiten (Stop-the-World Pausen). In der restlichen Zeit stehen der Applikation allerdings sehr viel Ressourcen zur Verfügung.
- **Pausenzeit (pause time):** Die Optimierung der Pausenzeiten hat zur Folge, dass die Applikation durch möglichst wenig Stop-the-World Pausen angehalten wird. Das ist insbesondere für Client-Server Applikationen wichtig. Das Ziel wird mit der Verwendung eines Concurrent Garbage Collection Algorithmus erreicht.
- **Determinismus (deterministic):** Optimiert den Garbage Collector auf sehr kurze und deterministische Garbage Collection Pausen.

Alias	Aktivierung	Beschreibung	Einstellungsmöglichkeiten
-------	-------------	--------------	---------------------------

throughput	-Xgc:throughput	Der Garbage Collector wird auf maximalen Durchsatz der Applikation eingestellt. Er arbeitet so effektiv wie möglich und erhält entsprechend viele Java-Threads, was zu kurzen Pausen der Applikation führen kann.	
pausetime	-Xgc:pausetime	Der Garbage Collector wird auf möglichst kurze Pausen eingestellt. Das bedeutet, dass ein konkurrierender Algorithmus verwendet wird, der insgesamt etwas mehr CPU Ressourcen benötigt.	-XpauseTarget=value (default 200msec)
deterministic	-Xgc:deterministic	Der Garbage Collector wird auf eine möglichst kurze und deterministische Pausenzeit eingestellt.	-XpauseTarget=value (default 30msec)

Tabelle 5.1: Übersicht der Garbage Collection Modi

5.2.2 Garbage Collection Algorithmen

Alias	Aktivierung	Generation	Pause	Durchs.	Heap	Mark	Sweep
singlecon, singleconcon	-Xgc:singlecon -Xgc:singleconcon	-	++	–	single	konk.	konk
gencon, genconcon	-Xgc:gencon -Xgc:genconcon	Old, Young	++	–	gen	konk.	konk.
singlepar, singleparpar	-Xgc:singlepar -Xgc:singleparpar	-	–	++	single	parallel	parallel
genpar, genparpar	-Xgc:genpar -Xgc:genparpar	Old, Young	–	++	gen	parallel	parallel
genconpar	-Xgc:genconpar	Old, Young	+	+	gen	konk.	parallel
genparcon	-Xgc:genparcon	Old, Young	+	+	gen	parallel	konk.
singleconpar	-Xgc:singleconpar	-	+	+	single	konk.	parallel
singleparcon	-Xgc:singleparcon	-	+	+	single	parallel	konk.

Tabelle 5.2: Übersicht der Garbage Collection Algorithmen

5.3 Garbage Collection Logdateien

Die Auswertung der Garbage Collection findet auf Basis von Logdateien statt. Das Format dieser Logdateien hängt mitunter auch von den aktivierten Log-

Modulen der JRockit Virtual Machine ab. Der Aufbau der Logdateien und die in der Analyse verwendeten Daten werden in diesem Abschnitt genauer beleuchtet.

5.3.1 Aktivierung Log Ausgaben

Defaultmässig macht die JRockit keine Angaben darüber, wie sie die Garbage Collection durchführt. Um an diese Informationen zu gelangen, muss das entsprechende Log-Module beim Start der Applikation aktiviert werden. Die Ausgaben werden dann auf die Standard Ausgabe geschrieben - können aber optional mit dem Argument `-Xverboselog:logdatei.log` in eine Datei umgeleitet werden. Das Format für die Kommandozeile sieht folgendermassen aus:

```
1 -Xverbose:<modul>[=log_level]
```

Listing 5.1: Format Aktivierung Log Modul

Um das Memory Log Modul (gibt Informationen über die Garbage Collection aus) zu aktivieren:

```
1 -Xverbose:memory
```

Listing 5.2: Garbage Collection Log (Info)

Die Umleitung der Ausgaben in eine separate Logdatei kann mit dem Flag `-Xverboselog:${dateiname}` eingestellt werden:

```
1 -Xverbose:memory -Xverboselog:gc.log
```

Listing 5.3: Garbage Collection Log (Info) - Umleitung in gc.log

Zusätzlich kann pro Log-Modul der Log-Level eingestellt werden:

```
1 -Xverbose:memory=debug -Xverboselog:gc.log
```

Listing 5.4: Einstellung des Log-Levels

Die folgenden Log Level können aktiviert werden: quiet, error, warn, info, debug, trace. Für mehr Informationen siehe [10].

Die ersten Zeilen einer Log-Datei mit eingeschaltetem Memory Modul sehen folgendermassen aus:

```

1 [INFO ][memory ] GC mode: Garbage collection optimized for throughput, strategy: Generational Parallel Mark
  & Sweep.
2 [INFO ][memory ] Heap size: 65536KB, maximal heap size: 1048576KB, nursery size: 32768KB.
3 [INFO ][memory ] <start>--<end>: <type> <before>KB-><after>KB (<heap>KB), <time> ms, sum of pauses <pause>
  ms.
4 [INFO ][memory ] <start> - start time of collection (seconds since jvm start).
5 [INFO ][memory ] <type> - OC (old collection) or YC (young collection).
6 [INFO ][memory ] <end> - end time of collection (seconds since jvm start).
7 [INFO ][memory ] <before> - memory used by objects before collection (KB).
8 [INFO ][memory ] <after> - memory used by objects after collection (KB).
9 [INFO ][memory ] <heap> - size of heap after collection (KB).
10 [INFO ][memory ] <time> - total time of collection (milliseconds).
11 [INFO ][memory ] <pause> - total sum of pauses during collection (milliseconds).
12 [INFO ][memory ] Run with -Xverbose:gcpause to see individual phases.
13 [INFO ][memory ] [OC#1] 0.843-0.845: OC 428KB->78423KB (117108KB), 0.003 s, sum of pauses 1.614 ms, longest
  pause 1.614 ms.
14 [INFO ][memory ] [OC#2] 1.393-1.442: OC 78449KB->156488KB (233624KB), 0.049 s, sum of pauses 47.104 ms,
  longest pause 47.104 ms.
15 [INFO ][memory ] [YC#1] 1.494-1.496: YC 156524KB->156628KB (233624KB), 0.002 s, sum of pauses 1.670 ms,
  longest pause 1.670 ms.
16 [INFO ][memory ] [YC#2] 1.496-1.496: YC 156652KB->156755KB (233624KB), 0.001 s, sum of pauses 0.605 ms,
  longest pause 0.605 ms.
17 [INFO ][memory ] [YC#3] 1.497-1.497: YC 156780KB->156884KB (233624KB), 0.001 s, sum of pauses 0.602 ms,
  longest pause 0.602 ms.
18 [INFO ][memory ] [YC#4] 1.497-1.498: YC 156908KB->157011KB (233624KB), 0.001 s, sum of pauses 0.592 ms,
  longest pause 0.592 ms.

```

Listing 5.5: Garbage Collection Log

5.3.2 Log Module

Die folgende Tabelle beschreibt die für die Auswertung der Garbage Collection relevanten Module (alphabetisch sortiert).

Modul	Beschreibung	Relevanz
alloc	Informationen betreffend Speicher Allokation und “Out-of-Memory”	niedrig
compaction	zeigt abhängig vom Garbage Collection Algorithmus Informationen zur Kompaktierung	niedrig
gcheuristic	zeigt Informationen und Entscheidungen zur Garbage Collection Heuristik	mittel
gcpause	zeigt wann welche Pausen zwecks Garbage Collection gemacht wurden, wie lange sie gedauert haben.	mittel
gcreport	zeigt verschiedene Auswertungen (Anzahl Collections, Anzahl promotete Objekte, maximal promotete Objekte per Zyklus, totale Zeit, Durchschnittszeit, Maximale Zeit) der Garbage Collection zum aktuellen Lauf	niedrig
memory	zeigt Informationen zum Memory Management System wie Start-, Endzeitpunkt der Collection, Speicher bevor, nach Collection, Grösse des Heaps, etc.	sehr hoch
memdbg	zeigt debug Informationen zu Speicher belange	hoch
systemgc	zeigt Garbage Collections die durch System.gc() gestartet wurden.	niedrig

Tabelle 5.3: Beschreibung der verschiedenen relevanten Log Modulen

Kapitel 6

Anforderungsanalyse

6.1 Einleitung

6.1.1 Zweck

Die Anforderungsanalyse dient als Basis für die folgenden Abschnitte:

- **Architektur und Konzept:** Die dokumentierten Anforderungen dienen als Grundlage für die Architektur des Systems und das Konzept.
- **Implementation:** Die Implementation der Anwendung richtet sich nach den ermittelten Anforderungen.
- **Verifikation:** Der implementierte Software-Prototyp wird anhand der in diesem Abschnitt ermittelten Anforderungen bewertet.

6.1.2 Systemumfang

Der folgende Abschnitt beschreibt die wesentlichen Teile innerhalb des Systems und des Systemkontexts.

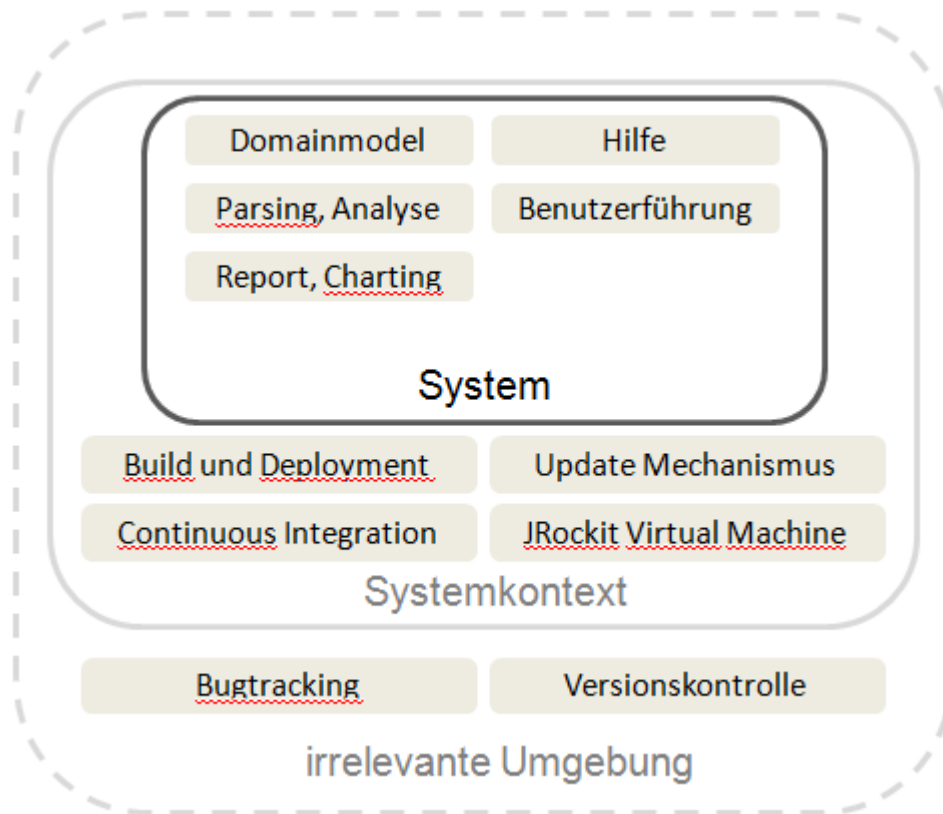


Abbildung 6.1: System und Systemkontext

System

Das System besteht aus den Teilen Datenmodell, Domäne, Log Parsing, Garbage Collection Analyse, Report, Charting, Hilfesystem¹, Benutzerführung.

Systemkontext

Zum Systemkontext gehören folgende nicht veränderbare Komponenten:

- **Build und Deployment, Continuous Integration:** Der Build der Software für neue Updates oder Releases wird zentral auf einem Server durchgeführt. Der Source-Code wird aus der Versionskontrolle ausgecheckt, die binären Pakete werden gebildet, es wird ein für den jeweiligen Update-Mechanismus notwendiges Packet erstellt und auf den Update-Server gestellt.

¹Dem Benutzer wird sowohl eine generelle wie auch eine kontextbezogene Hilfe angeboten.

- **Update Mechanismus:** Die Software wird in der Version 1.0 released. Anschliessend können Updates (Minor-, Major-Versionen) direkt - ohne den manuellen Download der Software - durchgeführt werden.
- **JRockit Virtual Machine:** Die Schnittstelle zur JRockit Virtual Machine findet über deren Logdateien statt. Die genauere Beschreibung befindet sich unter Garbage Collection Logdateien.

Irrelevante Umgebung

- **Bugtracking:** Sobald die Software stabil läuft und an Tester herausgegeben wird, wird für die Verwaltung der Fehler (Bugs) ein Bugtracker verwendet.
- **Versionskontrolle:** Der Source-Code der Applikation wird in einer Source-Code-Verwaltung abgelegt. Diese dient als Backup und zur Versionierung der einzelnen Artefakte. Die beiden Werkzeuge (Bugtracker, Versionskontrolle) arbeiten normalerweise eng zusammen, so dass Bug Fixes mit der eingetragenen Version in Verbindung bleiben.

6.1.3 Stakeholder

Für die Anforderungsanalyse sind nur die mit Asteriks gekennzeichneten Stakeholder-Rollen relevant. Die anderen Rollen können erst dann berücksichtigt werden, wenn die Software eine weitere Verbreitung wahrnehmen kann. Verschiedene Rollen können auch von einer Person wahrgenommen werden.

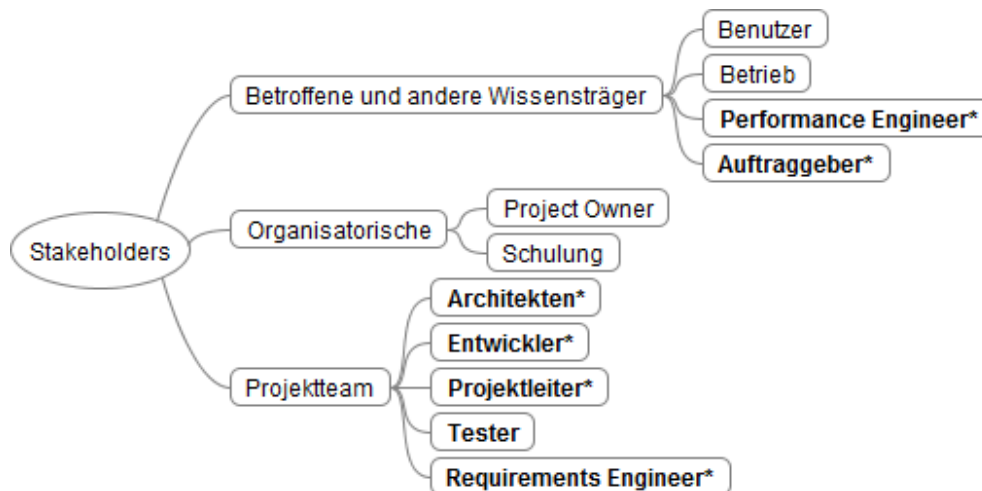


Abbildung 6.2: Übersicht der Stakeholder

6.1.4 Glossar

Das Glossar befindet sich auf Seite 81.

6.1.5 Referenzen

Die Referenzen innerhalb des Abschnitts Anforderungsanalyse befinden sich im Abschnitt Literaturverzeichnis.

6.1.6 Übersicht

Die Anforderungsanalyse ist folgendermassen aufgebaut: Beginnen tut sie mit der Allgemeinen Übersicht. Hier sind Architektur, Nutzer- und Zielgruppen, Randbedingungen und Annahmen dokumentiert. Danach sind die Use Cases der Software beschrieben. Aus diesen leiten sich dann die funktionalen Anforderungen und schliesslich die Qualitätsanforderungen ab.

6.2 Allgemeine Übersicht

6.2.1 Architekturbeschreibung

Die Software wird als Plugin programmiert. Der Entwickler kann sich die Software als Erweiterung in seiner Entwicklungsumgebung installieren. Sobald die Software installiert ist, wird für die Arbeit keine Verbindung ins Internet mehr benötigt. Die Applikation ist auf allen gängigen Betriebssystemen (Linux, Mac OSX, Windows) lauffähig.

6.2.2 Nutzer und Zielgruppen

Performance Engineers

Normalerweise Software Entwickler die viel Erfahrung, Wissen, Fähigkeiten haben und über die entsprechenden Werkzeuge verfügen, um die Ursachen für Performance-Probleme zu finden. Dabei handelt es sich nicht nur um Wissen im Bereich der Softwareentwicklung, sondern auch im Bereich des Servers und des Betriebssystems (Speichermanagement, I/O, etc.). Durch ihr breites Wissen sind sie mit der Unterstützung von Charts, Statistiken und Reports schnell in der Lage, Ursache von Performanceproblemen zu finden.

Java Entwickler

Im Gegensatz zu Performance Engineers beschäftigen sich Java Entwickler vor allem mit der Entwicklung von Anwendungen und verfügen nicht direkt über Knowhow im Bereich der Performanceanalyse. Als gut ausgebildete Ingenieure sind sie aber mit Hilfe von Werkzeugen und Dokumentation in der Lage, Performance Problemen innert nützlicher Frist auf den Grund zu gehen.

6.2.3 Randbedingungen

Das Format der Logdatei der JRocket Virtual Machine hat sich im Bereich des Memory Managements von der Version R27 auf die Version R28 geändert. Die Kompatibilität mit Versionen älter als R28 wird vorerst nicht gegeben sein.

6.3 Funktionale Anforderungen

Die folgende abschliessende Liste der Funktionalen Anforderungen an die Analysesoftware (stand 17. November 2011) wurde hinsichtlich Korrektheit, Machbarkeit, Notwendigkeit geprüft und entsprechend Priorisiert:

Identifik.	Vers.	Titel	Beschreibung	Quelle	Abnahmekriterium	Prio.
FRQ-01	1.0	Installation	Software muss als Erweiterung in der Entwicklungsumgebung ² installiert werden können.	Raffael Schmid (Project Owner)	Entwickler mit durchschnittlichen Kenntnissen benötigen für die Installation in eine bestehende Entwicklungsumgebung dauert weniger als 5 Minuten.	gross
FRQ-02	1.0	Updaten	Die Software kann mit geringem Aufwand aktualisiert werden.	Raffael Schmid (Project Owner)	Entwickler mit durchschnittlichen Kenntnissen für den Update weniger als 3 Minuten.	mittel
FRQ-03	1.0	Garbage Collection Logdatei importieren	Logdateien können importiert werden und werden anschliessend in einem Fenster dargestellt. Importierte Logdateien sind persistent und sind auch nach einem Neustart der Analysesoftware verfügbar.	Raffael Schmid (Project Owner)	-	gross

²Die Wahl der Entwicklungsumgebung respektive des Frameworks befindet sich im Abschnitt Auswahl Frameworks und Komponenten.

FRQ-04	1.0	Garbage Collection Logdatei einlesen	Geöffnete Garbage Collection Logdateien werden ins Memory gelesen.	Raffael Schmid (Project Owner)	Der Einleseprozess bei einer Datei mit 100000 Zeilen dauert weniger als 2 Sekunden.	gross
FRQ-05	1.0	Garbage Collection Logdatei parsen	Die eingelesenen Garbage Collection Logdatei wird geparkt. Aus den Daten wird ein Domänenmodell aufgebaut.	Raffael Schmid (Project Owner)	Das Parsen einer Logdatei mit 100000 Zeilen dauert nicht länger als 8 Sekunden.	gross
FRQ-06	1.0	Standardauswertung anzeigen	Der Benutzer kann eine vordefinierte Anzeige öffnen.	Raffael Schmid (Project Owner)	-	gross
FRQ-07	1.0	Profil (Benutzerdefinierte Auswertung) erstellen	Benutzer kann Profil erstellen und darauf entsprechende Charts konfigurieren. Das Profil kann gespeichert sowie exportiert und importiert werden.	Adrian Hummel (Performance Analyst)	-	klein
FRQ-08	1.0	Hilfesystem	Dem Benutzer werden eine indexbasierte und eine kontextsensitive Hilfe zur Verfügung gestellt.	Raffael Schmid (Project Owner)	Die Hilfe ist in Deutsch und Englisch verfügbar.	klein

Tabelle 6.1: Funktionale Anforderungen

6.4 Qualitätsanforderungen

6.4.1 Software

Identifik.	Vers.	Titel	Beschreibung	Quelle	Abnahmekriterium	Prio.
QRQ-S-01	1.0	Erweiterbarkeit	Nebst der Analyse von Garbage Collection Logs der JRockit Virtual Machine sollen später auch andere Formate unterstützt sein.	Raffael Schmid (Project Owner)	Erweiterung um ein weiteres Logformat soll den Aufwand von 5 PT ³ nicht überschreiten.	mittel
QRQ-S-02	1.0	Testabdeckung	Zur Qualitätskontrolle muss es eine angemessene Testabdeckung geben.	Raffael Schmid (Project Owner)	Angestrebte Coverage: 80%	klein
QRQ-S-03	1.0	Internationalisierung	Die Sprachelemente der Software (Labels, Titel, Texte) werden als Ressourcen definiert, was die spätere Erweiterung ermöglicht.	Raffael Schmid (Project Owner)	-	klein
QRQ-S-04	1.0	Usability	Schnelles Einlesen von grossen Dateien, Benutzerfeedback über ein Progressbar (Monitor).	Raffael Schmid (Project Owner)	Der Import einer Log-Datei von 100000 Zeilen dauert kürzer als 10 Sekunden. Dem Benutzer wird ein Monitor bereitgestellt.	mittel

³Personentage

QRQ-S-05	1.0	Korrektheit (angezeigte Werte)	Die berechneten und angezeigten Werte sind exakt.	Raffael Schmid (Project Owner)	berechnete und angezeigte Werte haben eine Genauigkeit von mindestens einem Zehntel (0.1).	gross
QRQ-S-06	1.0	Grösse Softwarepaket		Raffael Schmid (Project Owner)	Die grösse der gesamten Software soll 10 Megabyte nicht überschreiten.	mittel

6.4.2 Basisframework

Identifik.	Vers.	Titel	Beschreibung	Quelle	Abnahmekriter.	Prio.
QRQ-F-01	1.0	Verbreitung	Die Software wird als Erweiterung für eine Entwicklungsumgebung bereitgestellt. Die Verbreitung der Software spielt eine grosse Rolle.	Raffael Schmid (Project Owner)	-	gross
QRQ-F-02	1.0	Plattform-unabhängig	Software soll auf den gängigsten Betriebssystemen Windows und Apple OSX laufen.	Raffael Schmid (Project Owner)	Framework läuft auf den Plattformen Windows und Mac OSX.	gross
QRQ-F-03	1.0	Lokalisation	Framework muss Unterstützung für Lokalisation bereitstellen.	Raffael Schmid (Project Owner)	Framework bietet Unterstützung für die Mehrsprachigkeit.	klein

QRQ-F-04	1.0	Modularisierung	Framework muss Unterstützung für Modularisierung bieten, damit die Software in unterschiedliche Komponenten aufgeteilt werden kann (siehe Erweiterbarkeit QRQ-S-01).	Raffael Schmid (Project Owner)	Framework bietet Unterstützung für Modularisierung.	mittel
FRQ-F-05	1.0	Offline Betriebsmodus	Der Anwender soll die Software auch im Offline-Modus ⁴ benutzen können.	Raffael Schmid (Project Owner)	Eigenständige Software, keine Web Applikation	gross
FRQ-F-06	1.0	Installation als Erweiterung	Software wird als Erweiterung in einer Entwicklungsumgebung installiert.	Raffael Schmid (Project Owner)	-	gross

Tabelle 6.3: Qualitätsanforderungen Basisframework

⁴Auf einem Computer der sich nicht am Netz befindet.

6.5 Use Cases

Aus den Anforderungen ergeben sich die in diesem Abschnitt definierten Use-Cases. Zur Übersicht dient das folgende UML Use Case Diagramm, die Details sind nachfolgend in tabellarischer Form nach [12, S. 78-79] definiert.

6.5.1 Übersicht

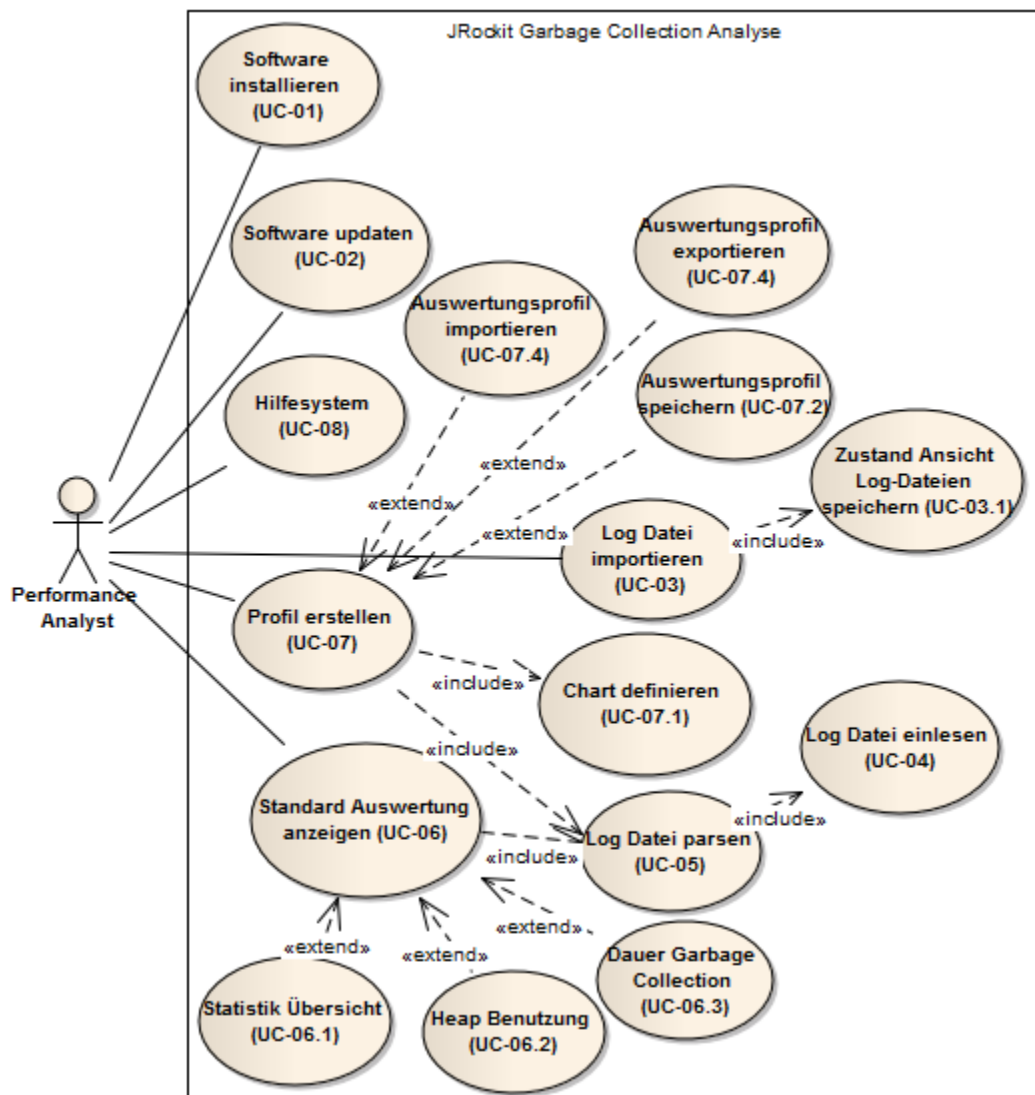


Abbildung 6.3: Systemfunktionalität als Use-Case-Diagramm

6.5.2 Beschreibung

Die definierten Use Cases leiten sich aus einer Anforderung ab, welche über die Nummer referenziert ist.

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-01
Bezeichner	UC-01
Name	Software installieren
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: mittel
Kritikalität	gross
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Der Benutzer kann die Software in seiner Entwicklungsumgebung installieren.
Akteure	Anwender, Entwicklungsumgebung
Auslösendes Ereignis	Anwender möchte eine Garbage Collection Logdatei analysieren.
Vorbedingung	Richtige Entwicklungsumgebung ist bereits ohne Analysesoftware installiert.
Nachbedingung	Es sind keine Fehler aufgetreten.
Ergebnis	Entwicklungsumgebung ist bereit für Garbage Collection Auswertungen.
Hauptszenario	<ol style="list-style-type: none"> 1. Anwender Startet Entwicklungsumgebung 2. Anwender gibt Update-Seite an 3. Anwender selektiert zu installierendes Softwarepaket 4. Softwarepaket wird installiert
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	-

Tabelle 6.4: Use-Case: Software installieren

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-02
Bezeichner	UC-02
Name	Software updaten
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: mittel

Kritikalität	Mittel
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Der Benutzer kann die Software updaten.
Akteure	Anwender, Update-Server
Auslösendes Ereignis	Anwender hat die Software bereits zu einem früheren Zeitpunkt installiert. Sofern ein neues Update vorhanden ist, möchte er dieses installieren.
Vorbedingung	Richtige Entwicklungsumgebung und Software wurden bereits in einer früheren Version installiert.
Nachbedingung	Es sind keine Fehler aufgetreten.
Ergebnis	Entwicklungsumgebung und Analysesoftware sind auf dem neusten Stand für Garbage Collection Auswertungen.
Hauptszenario	<ol style="list-style-type: none"> 1. Anwender Startet Entwicklungsumgebung 2. Anwender sucht und findet Updates für die Analysesoftware 3. Anwender selektiert eines oder mehrere dieser Softwarepakete 4. Software wird aktualisiert
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	-

Tabelle 6.5: Use-Case: Software updaten

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-03
Bezeichner	UC-03
Name	Garbage Collection Logdatei importieren
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: gering
Kritikalität	gross
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Der Benutzer importiert die sich auf dem Dateisystem befindende Logdatei.
Akteure	Anwender, Logdatei Importer
Auslösendes Ereignis	Anwender startet Garbage Collection Log Analyse
Vorbedingung	Logdatei befindet sich auf dem Rechner und ist in einem der unterstützten Formate. Die Software ist vollständig installiert und gestartet.
Nachbedingung	Es sind keine Fehler aufgetreten.

Ergebnis	Die Logdatei ist in der Ansicht Logdateien ersichtlich und kann von da im Analysefenster geöffnet werden.
Hauptszenario	<ol style="list-style-type: none"> 1. Anwender öffnet Import-Wizard 2. Anwender navigiert zum Ordner 3. Anwender selektiert Datei(en) und importiert diese
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	-
Erweiterungen	UC-03.1

Tabelle 6.6: Use-Case: Garbage Collection Logdatei importieren

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-03
Bezeichner	UC-03.1
Name	Zustand Ansicht Logdateien speichern
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: gering
Kritikalität	gering
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	In der Ansicht Logdateien befinden sich die importierten Logdateien, der Zustand dieser Ansicht wird gespeichert und bleibt über die Zeit einer Benutzersession bestehen.
Akteure	Entwicklungsumgebung
Auslösendes Ereignis	Entwickler schliesst Entwicklungsumgebung
Vorbedingung	Eine oder mehrere Logdateien wurden importiert.
Nachbedingung	Importierte Logdateien sind gespeichert.
Ergebnis	Bei einem Neustart der Entwicklungsumgebung bleiben die zuvor importierten Dateien erhalten.
Hauptszenario	Die Information, welche Dateien importiert wurde, wird gespeichert. Beim Neustart der Software wird sie genommen um die Liste der Importierten Dateien zu initialisieren.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	-

Tabelle 6.7: Use-Case: Zustand Ansicht Logdateien speichern.

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-04
Bezeichner	UC-04
Name	Garbage Collection Logdatei einlesen
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: gross
Kritikalität	gross
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Der Benutzer kann eine sich auf seinem Rechner befindende Garbage Collection Logdatei einlesen. Die Daten befinden sich anschliessend noch im unstrukturierten Zustand, erst der Parseprozess bringt es in eine strukturierte Form (Domänenmodell).
Akteure	Anwender, Logdatei Importer
Auslösendes Ereignis	Anwender startet Auswertung
Vorbedingung	Logdatei wurde bereits importiert.
Nachbedingung	Es sind keine Fehler aufgetreten.
Ergebnis	Die Logdatei befindet sich im Memory und steht für das Parsen durch die spezifische Erweiterung (JRockit) zur Verfügung.
Hauptszenario	Anwender öffnet Logdatei im Analysefenster. Die Logdatei wird eingelesen und befindet sich für die weitere Verarbeitung durch den Parser im Arbeitsspeicher.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Usability (QRQ-S-04)

Tabelle 6.8: Use-Case: Garbage Collection Logdatei einlesen

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-05
Bezeichner	UC-05
Name	Garbage Collection Logdatei parsen
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: gross
Kritikalität	gross
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Die sich im Arbeitsspeicher befindenden unstrukturierten Daten werden durch den Parseprozess in eine strukturierte Form gebracht (Domänenmodell).
Akteure	JRockit Garbage Collection Log Parser
Auslösendes Ereignis	Datei wurde fertig eingelesen.
Vorbedingung	Datei befindet sich im Arbeitsspeicher.

Nachbedingung	Kein Fehler ist aufgetreten.
Ergebnis	Domänenmodell ist fertig bereit für Abfragen darauf.
Hauptszenario	Jede einzelne Zeile in der Logdatei wird analysiert, die nötigen Daten werden extrahiert. Die Daten werden in ein Domänenmodell geschrieben.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Usability (QRQ-S-04), Korrektheit (QRQ-S-05)

Tabelle 6.9: Use-Case: Garbage Collection Logdatei importieren

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-06
Bezeichner	UC-06
Name	Standardauswertung anzeigen
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: gross
Kritikalität	gross
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Für eine schnelle Übersicht steht eine Standard-Auswertung zur Verfügung. Diese soll eine kurze Übersicht über die Garbage Collection geben und beinhaltet zwei Charts (Heap Benutzung, Dauer Garbage Collection).
Akteure	Anwender, Logdatei Analyzer, Report Engine
Auslösendes Ereignis	Die Applikation hat die Datei fertig eingelesen und geparkt.
Vorbedingung	Die Logdatei wurde ohne Fehler eingelesen und befindet sich im strukturierten Format im Arbeitsspeicher.
Nachbedingung	-
Ergebnis	Dem Benutzer wird ein Analyse-Screen angezeigt.
Hauptszenario	<ol style="list-style-type: none"> 1. Applikation hat die Logdatei fertig importiert. 2. Dem Benutzer wird ein Screen mit verschiedenen Tabs angezeigt. Auf jedem Tab wird dem Benutzer eine unterschiedliche Sicht auf die Daten gezeigt.
Alternativszenarien	Benutzerdefinierte Auswertung
Ausnahmeszenarien	-
Qualitäten	Korrektheit (QRQ-S-05)
Erweiterungen	UC-06.1, UC-06.2, UC-06.3, UC-06.4

Tabelle 6.10: Use-Case: Standardauswertung anzeigen

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-06
Bezeichner	UC-06.1
Name	Anzeige Statistik Übersicht
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: gross
Kritikalität	gross
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	<p>Der Analyse-Screen wurde geöffnet, dem Benutzer zeigen sich unterschiedliche Tabs. Auf dem ersten befinden sich verschiedene statistische Auswertungen der Logdatei:</p> <ul style="list-style-type: none"> • Übersicht und Grösse der verschiedenen Bereiche auf dem Heap: Initiale Grösse, endgültige Grösse • Aktivitäten des Garbage Collectors: Anzahl Young Collections, Anzahl Old Collections • Anzahl Garbage Collector Events (Bsp: Wechsel der Garbage Collection Strategie, etc.) • Garbage Collection Zeiten (Total, Durchschnittliche, Zeit in Old Generation Garbage Collection, Prozentuale Zeit in Old Generation Garbage Collection) • Durchsatz (siehe Abschnitt 3.4.1)
Qualitäten	Korrektheit (QRQ-S-05)

Tabelle 6.11: Use-Case: Anzeige Statistik Übersicht

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-06
Bezeichner	UC-06.2
Name	Anzeige Heap Benutzung
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: gross
Kritikalität	gross
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	<p>Die Heap Usage (Heap Benutzung) zeigt dem Benutzer anhand einer Grafik, zu welchem Zeitpunkt wieviel Speicher des Heaps verwendet wurde. Zusätzlich werden die Zeitpunkte inklusive entsprechendem Typ (Old- / Young-Collection) der Garbage Collection angezeigt.</p>

Qualitäten	Korrektheit (QRQ-S-05)
------------	------------------------

Tabelle 6.12: Use-Case: Anzeige Heap Benutzung

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-06
Bezeichner	UC-06.3
Name	Anzeige Dauer Garbage Collection
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: mittel Technologisches Risiko: mittel
Kritikalität	Mittel
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Für jede Garbage Collection ist innerhalb der Logdatei einen Eintrag mit den Informationen, wie viel Speicher von toten Objekten befreit wurde und wie lange die Collection gedauert hat. In diesem Report geht es um die Darstellung dieser Daten.
Qualitäten	Korrektheit (QRQ-S-05)

Tabelle 6.13: Use-Case: Anzeige Dauer Garbage Collection

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-07
Bezeichner	UC-07
Name	Profil (benutzerdefinierte Auswertung) erstellen
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: niedrig
Kritikalität	Niedrig
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Der Benutzer kann ein eigenes Profil erstellen. Dem Profil können eigene, benutzerdefinierte Charts hinzugefügt werden (siehe UC-07.1).
Akteure	Anwender, Logdatei Analyzer, Report Engine
Auslösendes Ereignis	Die Applikation hat die Datei fertig eingelesen und geparkt.
Vorbedingung	Die Logdatei wurde ohne Fehler eingelesen und befindet sich im strukturierten Format im Arbeitsspeicher.
Nachbedingung	-
Ergebnis	Dem Benutzer wird ein benutzerdefiniertes Analysefenster angezeigt.
Hauptszenario	Unabhängig vom Zyklus der Garbage Collection Analyse, kann der Benutzer ein eigenes Profil erstellen. Ein Profil besteht initial aus einem Übersichtsfenster der Garbage Collection und kann um eigene, benutzerdefinierte Charts erweitert werden.

Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Korrektheit (QRQ-S-05)
Erweiterungen	UC-07.1, UC-07.2, UC-07.3

Tabelle 6.14: Use-Case: Profil (benutzerdefinierte Auswertung) erstellen

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-07
Bezeichner	UC-07.1
Name	Chart definieren
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: niedrig
Kritikalität	Niedrig
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Der Benutzer erstellt auf dem Profil ein neues Chart und definiert darauf unterschiedliche Serien ⁵ .
Qualitäten	-

Tabelle 6.15: Use-Case: Chart definieren

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-07
Bezeichner	UC-07.2
Name	Auswertungsprofil speichern
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: niedrig
Kritikalität	Niedrig
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Profile können gespeichert werden und bleiben über einzelne Sitzungen bestehen, ausser der Benutzer löscht es explizit.
Qualitäten	-

Tabelle 6.16: Use-Case: Profil speichern, löschen

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-07
Bezeichner	UC-07.3

⁵Eine Serie definiert welche Daten auf der X- respektive Y-Achse angezeigt werden sollen.

Name	Auswertungsprofil exportieren
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: niedrig
Kritikalität	Niedrig
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Die vom Anwender definierten Profile können in eine Datei exportiert werden.
Qualitäten	-

Tabelle 6.17: Use-Case: Auswertungsprofil exportieren

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-07
Bezeichner	UC-07.4
Name	Auswertungsprofil importieren
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: niedrig
Kritikalität	Niedrig
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Exportierte Profile können ausgetauscht und von einem anderen Benutzer in die Entwicklungsumgebung importiert werden.
Qualitäten	-

Tabelle 6.18: Use-Case: Auswertungsprofil importieren

Abschnitt	Inhalt / Erläuterung
Anforderung	FRQ-08
Bezeichner	UC-8
Name	Hilfesystem
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: mittel
Kritikalität	Mittel
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Dem Benutzer steht eine indexbasierte ⁶ und eine kontextsensitive ⁷ Hilfe zur Verfügung.
Akteure	Anwender

⁶Generelle Hilfe mit Informationen zur Garbage Collection, Vorgehensweise bei Performance Problemen, alternative Werkzeuge, etc.

⁷Hilfe zur aktuellen View oder Aktion des Benutzers

Auslösendes Ereignis	Anwender hat Plugin installiert, weiss nicht wie eine Analyse gestartet werden kann.
Vorbedingung	Entwicklungsumgebung und Software sind installiert.
Nachbedingung	-
Ergebnis	Anwender kennt Software
Hauptszenario	<ul style="list-style-type: none"> • Indexbasierte Hilfe: Der Benutzer kennt sich im Thema Garbage Collection und auf der Analyse-Software noch nicht aus. Er holt sich Hilfe über die indexbasierte Hilfe. • Kontextsensitive Hilfe: Der Benutzer befindet sich in einem Fenster oder möchte eine Aktion ausführen (Context), das Hilfesystem zeigt ihm dazu die notwendigen Informationen.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	QRQ-S-03 (Internationalisierung)

Tabelle 6.19: Use-Case: Hilfesystem

Kapitel 7

Auswahl Frameworks und Komponenten

7.1 Einleitung

Die Evaluation der Komponenten wird auf der Basis der Anforderungen gemacht. Die jeweiligen Punktezahlen werden anhand folgender Schemas ausgerechnet.

Bewertung

Die Bewertung wird aus dem Erfahrungswert und folgender Tabelle in eine Punktzahl umgewandelt:

Bewertung	sehr schlecht	schlecht	mittel	gross	sehr gross
Punktzahl	1	2	3	4	5

Tabelle 7.1: Schema Vergabe der Punkte

Gewichtung

Die Priorität berechnet sich aus der Priorität der Anforderung und folgender Tabelle:

Priorität	sehr klein	klein	mittel	gross	sehr gross
Gewicht	1	2	3	4	5

Tabelle 7.2: Schema Umwandlung Priorität in Gewicht

Berechnung der Punkte

Die Punktezahl ergibt sich jeweils aus dem Produkt von Bewertung und Gewicht:

$$\text{Punktezahl} = \text{Bewertung} \times \text{Gewicht}$$

7.2 Rich Client Framework

Grundsätzlich käme für die Entwicklung der Analyse-Software auch die Java GUI-Bibliothek Swing in Frage. Viele Anforderungen die im Zusammenhang mit Desktop-Applikationen entstehen (Deployment, Installation, Update, Modularisierung, Internationalisierung, etc.), müssen dann allerdings neu entwickelt werden. Bei der Evaluation der Bibliothek werden darum nur Rich Client Frameworks berücksichtigt. Von denen kommen aufgrund der funktionalen Anforderung FRQ-F-05 (Offline Betriebsmodus) und FRQ-F-06 (Installation als Erweiterung) Eclipse RCP und Netbeans RCP in Frage. Während Eclipse RCP insbesondere als Entwicklungsumgebung ein weit verbreitetes Framework ist und die Entwicklung aktuell in zwei verschiedenen Versionen (3.x / 4.x) vorangetrieben wird, findet man Netbeans und deren Rich Client Plattform eher selten. In der Folge werden die drei Frameworks kurz beschrieben und dann anhand der Anforderungen verglichen.

7.2.1 Übersicht

Eclipse RCP

Bis zur Version 2.1 war Eclipse bekannt als eine Open Source Entwicklungsumgebung für Programmierer. Der Vorgänger hiess Visual Age vor Java und wurde von IBM entwickelt.

Auf die Version 3.0 wurde die Architektur von Eclipse relativ stark umgestellt und modularisiert. Nun handelt es sich um einen relativ kleinen Kern, der die eigentlichen Funktionalitäten der Applikation als Plugins lädt. Der beschriebene Mechanismus basiert auf Eclipse Equinox, einer Implementation der OSGi Spezifikation. Die grafischen Benutzeroberflächen werden in SWT implementiert. Eclipse ist für 14 verschiedene Systeme und Architekturen bereitgestellt und gilt somit als plattformunabhängig[15].

Die Plattform kann nun auch als Framework zur Entwicklung von Desktop Applikationen oder Plugins für die Entwicklungsumgebung verwendet werden.

Netbeans RCP

Bei Netbeans RCP handelt es sich ebenfalls um ein Framework zur Entwicklung von Desktop Anwendungen. Der Kern der Netbeans Plattform besteht ebenfalls aus einem Modul-Loader und im Bereich der grafischen Benutzeroberfläche wird Swing verwendet. Man findet Netbeans als Rich Client Framework eher selten.

7.2.2 Auswertung Rich Client Frameworks

Die erste und zweite Spalte zeigen die Anforderungen aus Abschnitt 6.4.2. Die dritte Spalte enthält das aus der Priorität abgeleitete Gewicht (siehe Abschnitt Gewichtung). Die weiteren Spalten zeigen pro Produkt die Bewertung zusammen mit dem Total der Punkte.

Anforderung	Nummer	Gewicht.	Eclipse 3.x	Eclipse 4.x	Netbeans 3.x
Verbreitung	(QRQ-F-01)	4	5 (20)	1 (4)	2 (8)
Unterstützung Plattformunabhängigkeit	(QRQ-F-02)	4	5 (20)	5 (20)	5 (20)
Unterstützung Lokalisation Support	(QRQ-F-03)	2	5 (10)	5 (10)	5 (10)
Unterstützung Modularisierung	(QRQ-F-04)	3	5 (15)	5 (15)	5 (15)
Total			65	49	53

Tabelle 7.3: Auswertung Rich Client Frameworks

7.2.3 Entscheid

Ausschlaggebend für die Wahl der Rich Client Plattform ist die Verbreitung. Trotz der Unterschiede in Architektur und den verwendeten Technologien unterscheiden sich die Frameworks im Funktionsumfang nur unwesentlich.

7.3 Charting Bibliothek

Für die grafische Darstellung der Daten (Charts) wird eine Bibliothek verwendet. Sehr oft wird dafür Eclipse BIRT oder JFreeChart eingesetzt.

7.3.1 Übersicht

BIRT

BIRT steht für Business Intelligence and Reporting Tools und stellt Business-Intelligence- und Reporting-Funktionalität für Rich Clients zur Verfügung. BIRT ist lizenziert unter der Eclipse Public License und ist daher open-source. BIRT ist wie Eclipse ein Top-Level-Softwareprojekt der Eclipse Foundation. Die Software besteht aus zwei Teilen:

- **Report Designer:** Mit dem Report Designer können Benutzer oder Administratoren ihre Reports erstellen und anpassen.
- **Charting Engine:** Die Charting Engine generiert aus den Daten und den definierten Reports, Diagrammen die Charts.

Der Umfang der Software ist mit über 100 Megabyte für kleinere Projekte eher ungeeignet.

JFreeChart

JFreeChart ist ein Open-Source-Framework mit welchem eine Vielzahl verschiedener Diagramme erzeugt werden können. JFreeChart unterstützt die Ausgabe der Diagramme in Grafiken, Swing- und RCP-Applikationen und ist mit einer Grösse von rund einem Megabyte auch für kleinere Softwareprojekte geeignet.

7.3.2 Auswertung Charting-Bibliothek

Anforderung	Nummer	Gewicht.	BIRT	JFreeChart
Grösse Softwarepacket	(QRQ-S-06)	4	1 (4)	5 (20)
Verbreitung	(QRQ-F-01)	4	4 (16)	5 (20)
Total			20	40

Tabelle 7.4: Auswertung Charting-Bibliothek

7.3.3 Entscheid

JFreeChart eignet sich insbesondere für kleinere Client-Applikationen sehr gut zur Erstellung von Diagrammen. Die Flexibilität welche bei Birt durch den Report-Designer bereitgestellt wird, wird in der Analysesoftware nicht benötigt. Aus diesen und oben aufgelisteten Gründen wird JFreeChart als Charting-Bibliothek verwendet.

Kapitel 8

Konzept

8.1 Allgemein

8.1.1 Ausgangslage

Als Rich Client Framework wird Eclipse 3.x¹ genommen. Siehe Abschnitt Auswahl Frameworks und Komponenten. In der Folge werden die für die Eclipse Plattform gebräuchlichen Begrifflichkeiten verwendet.

8.1.2 Lizenzierung der Software

Die Software wird lizenziert unter der Eclipse Public License² in der Version 1.0. Dies ist eine freie Software-Lizenz und gewährt das Recht zur freien Nutzung, Weiterverbreitung und Veränderung der Software. Die Benutzung einer Open-Source Lizenz hat insbesondere folgende Vorteile:

- An der Entwicklung von Open-Source Software können sich eine beliebige Anzahl an Entwicklern beteiligen. Der Entwicklungsaufwand kann skaliert werden.
- Jedermann kann Erweiterungen entwickeln oder Fehler beheben.

8.2 Architektur

8.2.1 Problemstellung

Aufgrund der Anforderung QRQ-S-01 (Erweiterbarkeit) muss die Analyse-Software auch hinsichtlich anderer Logformate erweiterbar sein. Die Applikation wird also nicht zwingendermassen mit der Erweiterung für JRockit Logdateien verwendet. Es könnte zu einem späteren Zeitpunkt sein, dass man damit Garbage Collection

¹die aktuelle Version ist 3.7 (stand: 31.8.2011)

²<http://www.eclipse.org/legal/epl-v10.html>

Logs der HotSpot Virtual Machine ausgewertet. Dies hat hinsichtlich Architektur einige Konsequenzen:

- Die Applikation soll in zwei Komponenten aufgeteilt werden:
 - Basissoftware
 - Erweiterung JRockit

Die Basissoftware kann unabhängig von den Erweiterungen installiert werden, für die Auswertung einer Logdatei ist allerdings die entsprechende Erweiterung zu installieren. Eine Erweiterung kann ohne Basissoftware nicht gebraucht werden.

- Die Architektur der Applikation muss es zulassen, dass zu einem späteren Zeitpunkt auch Garbage Collection Logs von anderen Virtuellen Maschinen analysiert werden können.
- Die Basissoftware stellt Extension-Points³ bereit, über welche sich die Erweiterungen registrieren.

8.2.2 Übersicht

Die im Abschnitt Problemstellung beschriebenen Konsequenzen führen dazu, dass die Applikation - obwohl es momentan erst die Erweiterung für die JRockit Garbage Collection Logs gibt - in zwei verschiedene Features aufgeteilt wird. Die beschriebenen Anforderungen können grob folgendermassen zugewiesen werden:

- Basissoftware (Core Feature)
 - Garbage Collection Log importieren
 - Garbage Collection Log einlesen
 - Profil erstellen, speichern, exportieren, importieren
 - Hilfesystem
- JRockit Extension (JRockit Extension Feature)
 - Garbage Collection Log parsen
 - Standardauswertung: Heap, Dauer Garbage Collection

8.2.3 Projektstruktur

Wie im Abschnitt Installation (UC-01) erläutert, besteht die Software aus zwei getrennten Features. Das Core-Feature ist die Basis und verantwortlich für den gesamten Import-Prozess (Import-Wizard, Leseprozess der Log-Datei, Anzeige

³Ein Extension-Point ist ein Mechanismus der von Eclipse zur Verfügung gestellt wird, damit eine Komponente (Plugin) sich bei einer anderen registrieren kann.

der Menus, Profil-Verwaltung, etc.). Die JRocket Extension ist eine für die Garbage Collection Logs der JRocket geschriebene Erweiterung. Sie ist für das Parsing der Logdateien, die Aufbereitung und Bereitstellung der Daten zuständig. Beinhaltet aber keine Basisfunktionalität.

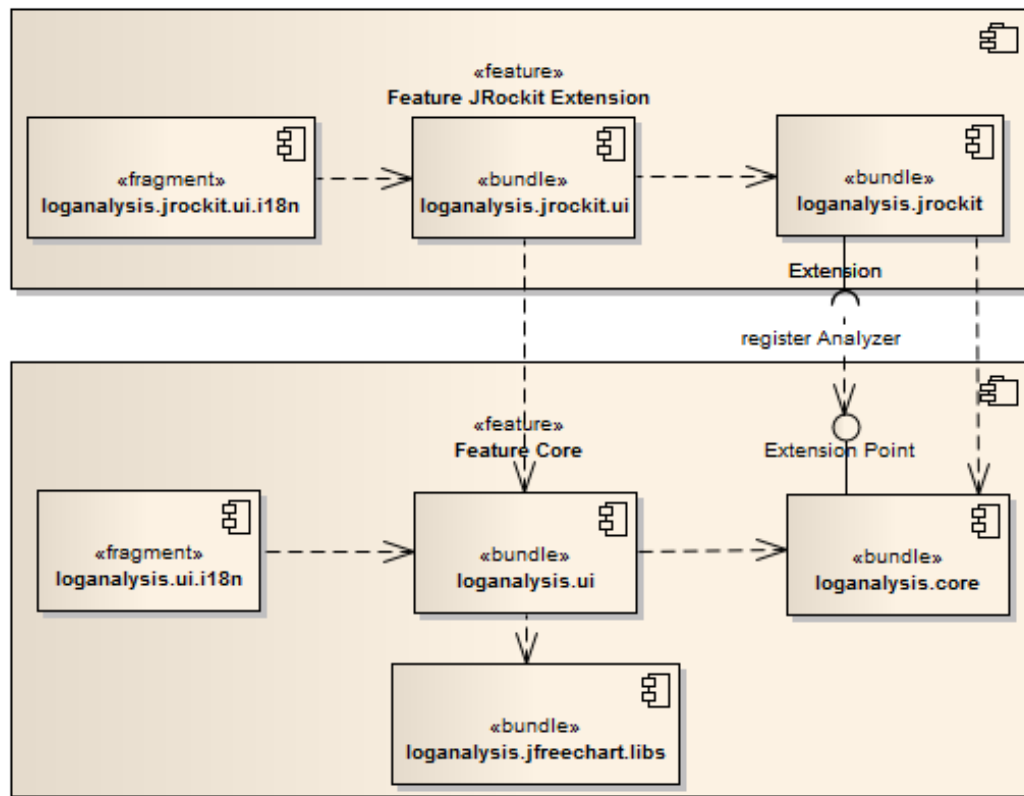


Abbildung 8.1: Architektur: Komponentendiagramm

Das Core Feature besteht aus dem Modul User Interface (“loganalysis.core.ui”) und einem von JFace und SWT⁴ unabhängigen Teil (“loganalysis.core”). Öffnet der Benutzer eine Garbage Collection Logdatei, wird diese durch das Core Feature eingelesen und an alle verfügbaren Extensions weitergeleitet. Die erste Extension welche den Inhalt der Datei versteht, öffnet seine dafür vorgesehenen Reports und Charts. Jede Extension hat ein basierend auf der Logdatei eigenes Domänen-Modell.

Weitere Projekte

Einige Plugins wurden im vorherigen Abschnitt nicht erwähnt:

⁴JFace und SWT wird in Eclipse als Library für die Präsentationsschicht verwendet.

- Test-Projekte⁵
 - `core.test`
 - `core.ui.test`
 - `jrookit.test`
 - `jrookit.ui.test`
- Features⁶
 - `loganalysis.feature`
 - `loganalysis.jrookit.feature`
- Thirdparty Bibliotheken⁷
 - `loganalysis.jfreechart.libs` (JFreeChart Library)
- Targetplattform⁸
 - `loganalysis.targetplatform` (beinhaltet die Target-Plattform)
- Update-Seite⁹
 - `loganalysis.update-site` (definiert und generiert die Update-Seite)

8.2.4 Ablauf Garbage Collection Analyse

Beim öffnen einer Analyse wird die zuständige Extension gesucht¹⁰, welche den Inhalt¹¹ der Logdatei versteht. Die Erweiterung ist für das Parsen und Aufbereiten der Daten und die Anzeige in einem Analysefenster zuständig.

⁵Der Test-Code befindet sich in eigenen Projekten (siehe Abschnitt 8.11.1).

⁶Features sind in sich lauffähige Softwarekomponenten mit definiertem Umfang.

⁷Thirdparty Bibliotheken werden bei Eclipse-Anwendungen als Plugins gepackt.

⁸Definiert gegen welche Plattform die Anwendung entwickelt wird.

⁹Definiert bereitgestellte Features

¹⁰Extensions registrieren sich via das `plugins.xml` an einem Extension-Point.

¹¹Der Inhalt der Datei wird lazy via die Methode `getContent` und einem `ContentReader` geladen.

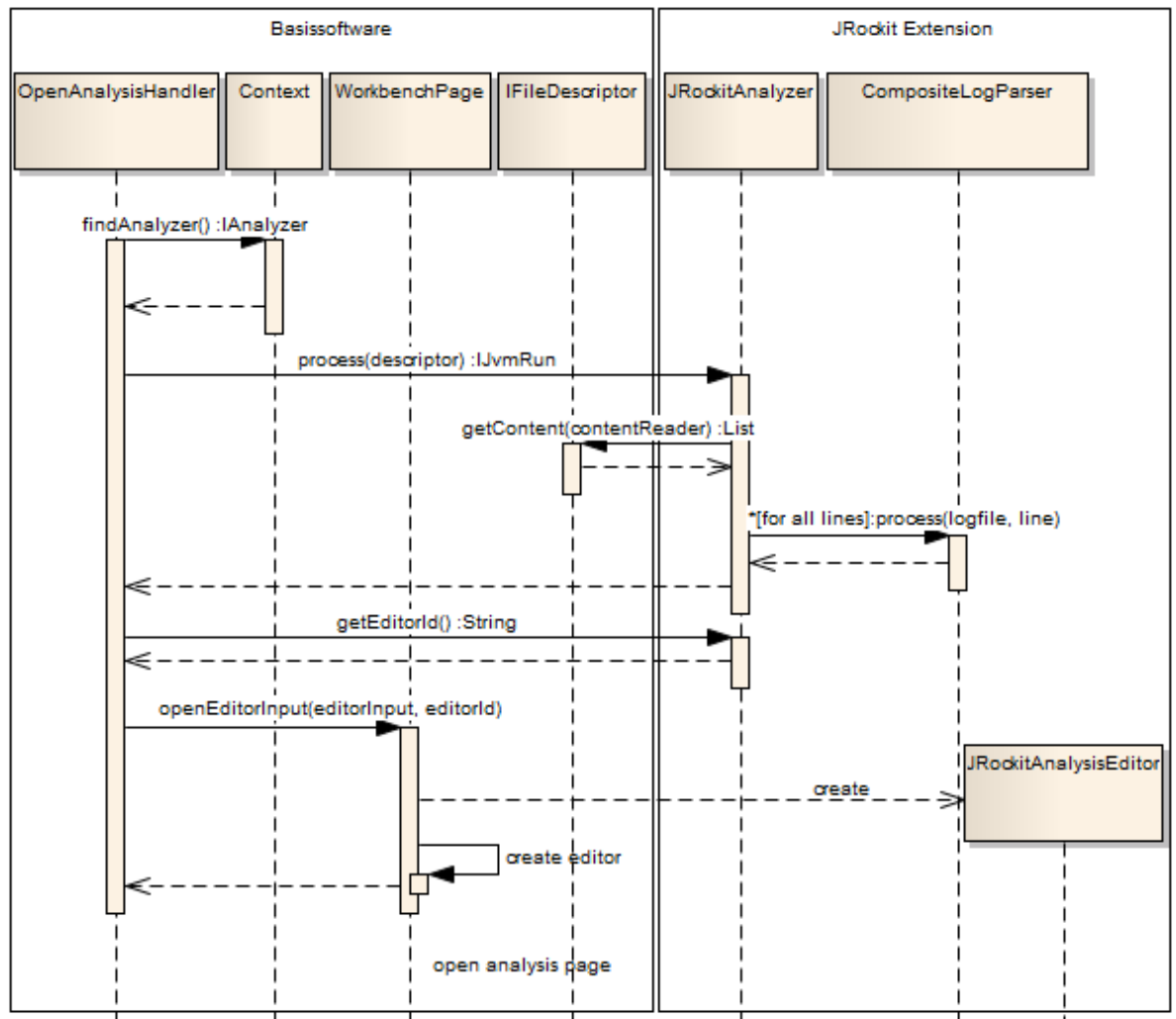


Abbildung 8.2: Sequenz-Diagramm Öffnen der Analyse

Der Ablauf der Garbage Collection Analyse ist folgendermassen:

1. Via Context wird die Erweiterung¹² gesucht.
2. Analyzer der Erweiterung parst inhalt und gibt ihn als Objekt zurück.
3. Editor wird geöffnet, das Objekt wird als Input übergeben.

¹²Erweiterungen registrieren sich als Extensions an Extension-Points. Diese Konfiguration befindet sich im plugin.xml.

8.3 Installation (UC-01)

Zur Installation der Software benötigt man die Eclipse-Entwicklungsumgebung in der Version 3.7. Darin integriert befindet sich ein Update-Manager, der Software-Komponenten von Lokal oder dem Netzwerk installieren kann. Auch Updates werden über diesen Mechanismus installiert. Die Analyse-Software wird via eine Update-Seite bereitgestellt. Der Software-Build durch das Continuous Integration System publiziert die Artefakte (Features, Plugins) auf einen via Internet zugänglichen Server, von welchem der Update-Manager die Software herunterlädt um anschliessend zu importieren. Update-Seiten im Eclipse-Umfeld bestehen aus Features (Eclipse Feature-Projekt). Features bestehen aus unterschiedlichen Plugins (Eclipse Plugin-Projekt). Eclipse¹³ ist in der Lage, Plugins zur Laufzeit zu installieren, starten, deinstallieren.

Die Update-Seite für diese Software wird folgendermassen aufgebaut:

- **Basissoftware:** Umfasst alle Plugins, die für die Basissoftware notwendig sind. Siehe Abschnitt Projektstruktur.
- **JRockit Erweiterung:** Umfasst alle Plugins zur JRockit Erweiterung und hat zugleich die **Abhängigkeit auf das Basissoftware-Feature**.

8.4 Update (UC-02)

Der Update eines Features auf der Update-Seite ist erkennbar durch eine Änderung der Major respektive Minor Version oder aber durch Änderung des an die Feature-Datei angehängten Zeitstempels¹⁴. Beim Starten der Software wird überprüft, ob ein sich auf dem Server befindendes Feature aktualisiert wurde. Der Benutzer kann diese im laufenden Betrieb der Applikation installieren. Danach ist allerdings ein Neustart der Applikation nötig.

8.5 Datei importieren (UC-03)

Der Import einer Logdatei findet über einen Import-Wizard statt. Der Ablauf zum Import einer oder mehrerer Dateien ist folgendermassen:

1. Import-Wizard öffnen
2. Auswahl des Ordners
3. Selektion einer oder mehrerer Logdateien
4. Bestätigung der Eingaben
5. Anschliessend wird die Logdatei als Instanz von IFileDescriptor in der Ansicht Logdateien angezeigt.

¹³in der Basis ist es Equinox, die Implementation des OSGi Standards

¹⁴Mittels der Versionsnummer x.x.qualifier erreicht man, dass der Zeitstempel ans Dateieende gehängt wird.

8.6 Importierte Dateien speichern (UC-03.1)

Der im Abschnitt 8.12.1 beschriebene Mechanismus wird verwendet, damit nach einem Neustart der Entwicklungsumgebung die importierten Logdateien nicht verloren gehen.

8.7 Datei einlesen (UC-04)

Durch den Import einer Garbage Collection Datei erscheinen diese im Logdateien-Fenster. Das Öffnen einer dieser Dateien lädt den Inhalt in den Arbeitsspeicher. Die Daten sind noch unstrukturiert und werden innerhalb des im nächsten Abschnitt definierten Domänenmodells gespeichert.

8.7.1 Domänenmodell

IFileDescriptor wird für die Abstraktion der Garbage Collection Logdatei verwendet. Darin enthalten sind Metadaten wie Dateiname und Pfad sowie - wenn bereits geladen - der Inhalt der Datei. Die Abstraktion einer Garbage Collection Logdatei heisst *AbstractJvmRun* und wird erst von der jeweiligen Erweiterung realisiert.

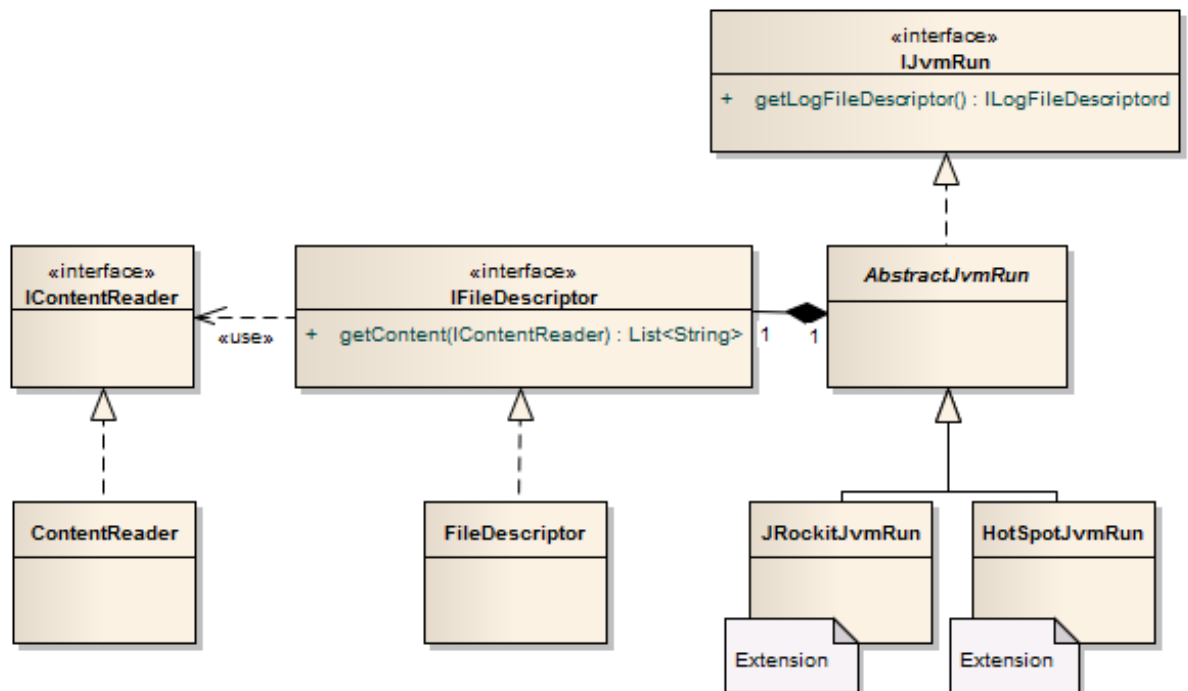


Abbildung 8.3: Domänenmodell: Eingeladene Datei

8.8 Profil erstellen (UC-07)

Die Ansicht Profile zeigt die vom Benutzer erstellten Profile¹⁵. Die Beschreibung eines Analysefensters (Name, Beschreibung, Charts mit Serien, etc.) wird anhand von Profilen definiert. Es gibt folgende zwei Arten:

- **Unveränderlich Profil:** Das Standard-Profil ist aktuell das einzige unveränderliche Profil.
- **Veränderlich Profil:** Ein veränderliches Profil wird zur Speicherung des vom Benutzer definierten Analysefensters verwendet. Alle Änderungen die der Benutzer am Analysefenster macht (Chart hinzufügen, Chart konfigurieren), werden via ein Data-Binding an das Profil propagiert. Durch das Speichern des Profils hat der Benutzer die Möglichkeit, die selbe Analyse auch zu einem späteren Zeitpunkt an der gleichen oder einer anderen Logdatei durchzuführen.

8.8.1 Domänenmodell

IConfiguration dient zur Gruppierung von profilen. Pro Extension wird eine Konfiguration mit verschiedenen Profilen abgelegt. Innerhalb eines Profils können unterschiedliche Diagramme (*IChart*) angelegt werden, welche wiederum durch Achsen (*IAxis*) und deren Datenquellen *IValueProvider* definiert sind. *IValueProvider* definieren den Weg, wie die Daten aus dem Domänenmodell gelesen werden.

¹⁵Initial befindet sich darin allerdings nur das Standard-Profil.

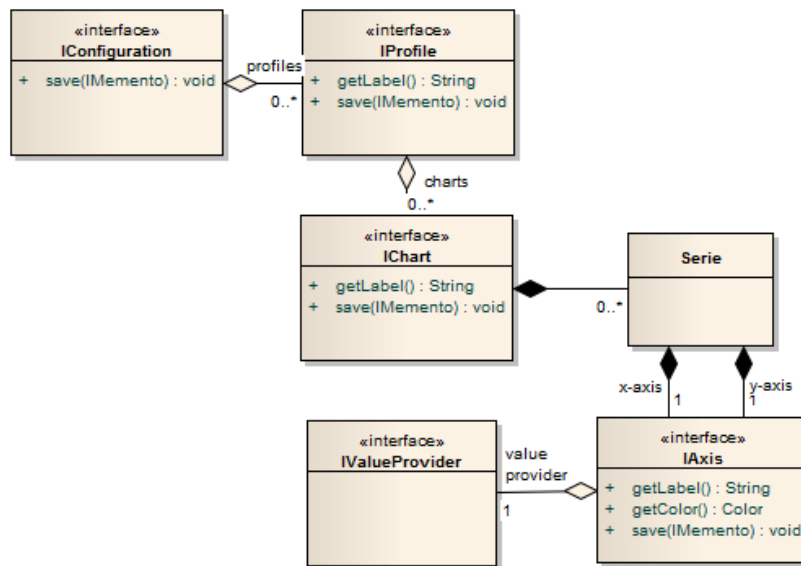


Abbildung 8.4: Domänenmodell: Profile

8.8.2 Charts definieren (UC-07.1)

Bei der Benutzung eines veränderlichen Profils hat der Benutzer die Möglichkeit, dem Analysefenster weitere Charts respektive Diagramme hinzuzufügen oder aber bereits existierende Diagramme zu manipulieren (weitere Serien hinzufügen, Serien entfernen, etc.). Die Manipulationen des Benutzers finden auf den Chart-Objekten statt und werden durch das Data-Binding propagiert. Die Administrationsoberfläche für einen Chart umfasst initial zwei Abschnitte:

- Serie erstellen, definieren und hinzufügen
- Serie löschen

8.8.3 Profil speichern (UC-07.2)

Mittels des Memento-(siehe Abschnitt 8.12.1) und Visitor-Patterns[2, S. 331] werden die Profile gespeichert.

8.8.4 Profil exportieren, importieren (UC-07.3/4)

Die Analysesoftware stellt zum Sichern und Verteilen von Profilen einen Import-Export-Mechanismus bereit. Beide sind in den Eclipse Standard-Wizards ersichtlich oder können via rechter Mouseklick (Ansicht Profile) gestartet werden. Der Profile-Export und -Import basiert wie das Speichern auf dem im Abschnitt 8.12.1 beschriebenen Memento-Mechanismus. Zur Serialisierung in eine Datei wird das XMLMemento verwendet.

8.8.5 Hilfesystem (UC-08)

Das Hilfesystem der Eclipse Entwicklungsumgebung ist als Client-Server-Lösung implementiert. Beim Start der Entwicklungsumgebung wird zusätzlich ein Jetty-Server gestartet, der die Hilfsdienste wie Suche und Indexierung bereitstellt. Hilfeseiten sind auf unterschiedliche Weise verfügbar:

- **Indexbasierte Hilfen:** Für die generellen Informationen und Hilfen werden verschiedene Hilfeseiten basierend auf einem Index bereitgestellt. Die Inhalte sind nicht an ein Fenster oder eine Aktion des Benutzers gebunden.
- **Kontextsensitive Hilfen:** Tipps die im Zusammenhang mit einer Aktion oder eines Fensters stehen, werden mit den Kontextsensitiven Hilfen implementiert.

8.9 Garbage Collection Logdatei parsen (FRQ-05)

Die Garbage Collection Logs der JRockit Virtual Machine bestehen aus Einträgen unterschiedlicher Log Module. Diese Ausgaben können selektiv per Kommandozeile aktiviert werden (siehe Abschnit 5.3.2). Für die Garbage Collection Analyse sind nicht alle Einträge interessant, es werden nur die wichtigsten verwendet. Die einzelnen Parser der für die Garbage Collection Logdatei werden auf der Basis von Regular Expressions selber implementiert. Auf die Verwendung eines Parser-Generators wird aus folgenden Gründen verzichtet:

- **Lightweight:** Um Parser-Generatoren zu verwenden, wird in der Regel auch zur Laufzeit eine Bibliothek benötigt. Diese muss mit der Software ausgeliefert werden. Die Implementation von Regulären Ausdrücken beispielsweise ist der Java Laufzeitumgebung sowieso verfügbar.
- **Proprietär:** Parser-Generatoren sind proprietär und die Verwendung dessen bedingt gute Kenntnisse. In der Regel beschreibt man das Format in einer Grammatik.

8.9.1 Parser

Der Parser für die Logdateien der JRockit Virtual Machine ist nach dem Chain-of-Responsibility Pattern[14] aufgebaut. Pro Logeintrag (respektive pro Typ) kann ein Parser in die Parser-Kette geschaltet werden. Jeder Parser extrahiert die für ihn wichtigen Informationen und aktualisiert damit das Domänenmodell. Die wichtigsten Einträge des Garbage Collection Logs sind die des Memory Modules und werden vom *MemoryModuleParser* verarbeitet. In zukünftigen Versionen ist die Verarbeitung von Logeinträgen von anderen Log-Modulen denkbar - Einträge des Nursery-, Allocation-Modules, etc. Ablauf und Aufbau des Parsers sind in den folgenden beiden Grafiken ersichtlich:

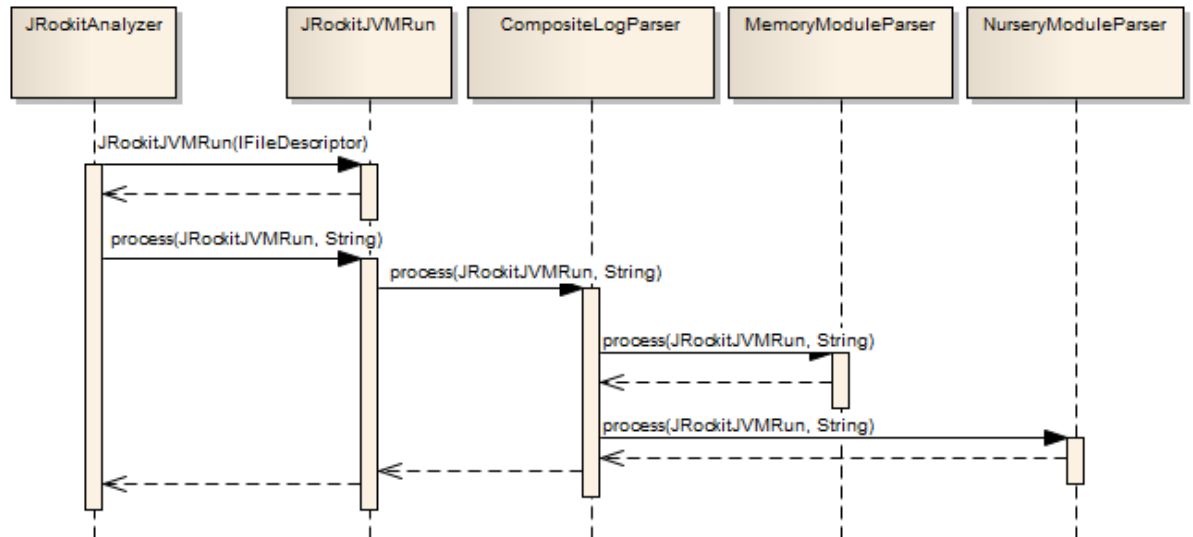


Abbildung 8.5: Sequenzdiagramm Parsen Logdatei

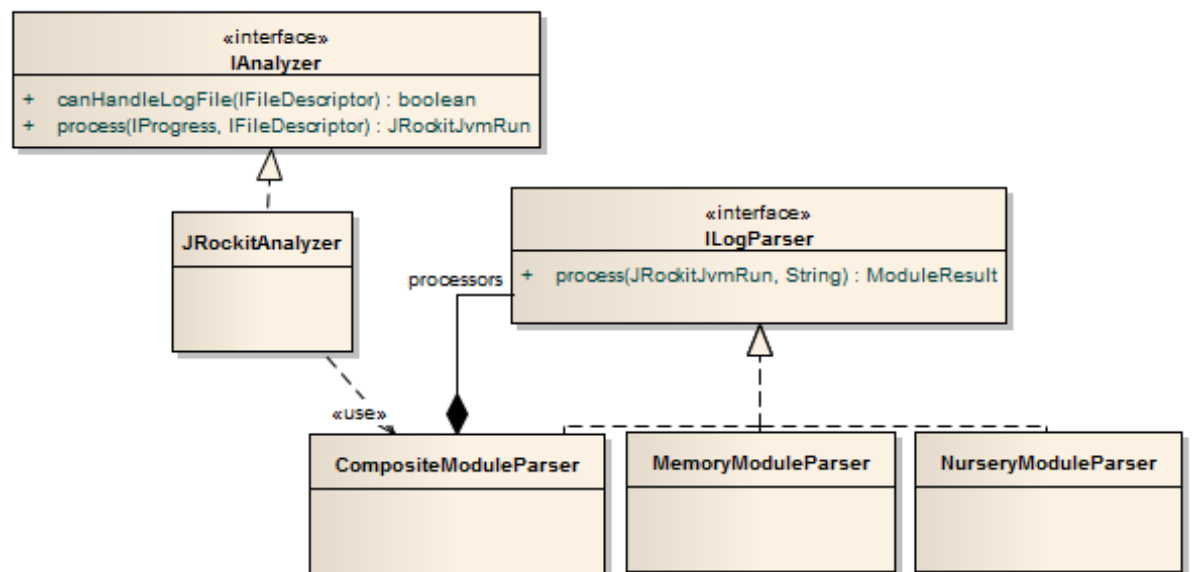


Abbildung 8.6: Klassendiagramm Parsen Logdatei

8.9.2 MemoryModuleParser

Das Parsen innerhalb des *MemoryModuleParser* wird in zwei Schritte aufgeteilt:

- **Lexer:** Der einzelne Logeintrag wird durch den Lexer (Tokenizer) in eine Map aus Tokens umgewandelt. Schlüssel für die einzelnen Tokens ist der *TokenType*. Der Lexer wird mittels Regular Expressions implementiert. Die Werte werden mittels Gruppierungs-Funktion¹⁶ extrahiert.
- **Syntactic Analyzer:** Der syntaktische Analyser verarbeitet die vom Lexer extrahierten Tokens. Er führt zu einer semantischen Validierung durch und speichert die Werte in strukturierter Form (Domänenmodell).

8.9.3 Auswertung Logdatei

Bereits in Abschnitt 5.3.1 ist ein Teil einer Garbage Collection Logdatei aufgelistet. Dieser Abschnitt zeigt die für die Garbage Collection Analyse wichtigsten Informationen, ohne dabei aber auf die Ausgaben des Debug-Log-Levels¹⁷ einzugehen.

Garbage Collection Algorithmus

```
[INFO ][memory ] GC mode: Garbage collection optimized for
      throughput, strategy: Generational Parallel Mark & Sweep.
```

Listing 8.1: Logdatei: Ausgabe initialer Garbage Collection Algorithmus

Die eigentlich wichtigste Information respektive Entscheidung zum Garbage Collection Tuning ist die Wahl der Strategie. Sie wird im Header der Logdatei angezeigt und ist entspricht entweder der Standardeinstellung oder wird konfiguriert. Ausgewertet aus dem Eintrag werden zwei Dinge: für was die Garbage Collection optimiert ist (Durchsatz oder Pausenzeiten) und die Strategie (Generational Parallel Mark & Sweep, Parallel Mark & Sweep, Generational Concurrent Mark & Sweep, Concurrent Mark & Sweep)

Initiale und maximale Heap-Kapazität, Grösse Old- und Young-Space

```
[INFO ][memory ] Heap size: 65536KB, maximal heap size: 1048576KB,
      nursery size: 32768KB.
```

Listing 8.2: Logdatei: Initiale und maximale Heap-Kapazität und Grösse des Old- und Young-Spaces

Ebenfalls im Header der Logdatei befindet sich die Angabe über die initiale und maximale Heap-Kapazität und die Grösse des Young-Spaces (Nursery). Die Grösse des Old-Spaces (Tenured Space) errechnet sich aus der Differenz zwischen Young-Space und maximaler Kapazität.

¹⁶Mittels Gruppen innerhalb von Regulären Ausdrücken lassen sich einzelne Werte aus einem String extrahieren.

¹⁷Der Log-Level kann für jeden einzelnen Logger eingestellt werden.

Young-Collection Information

```
[INFO ][memory ] [YC#660] 2.172-2.172: YC 200108KB->200147KB  
(233624KB), 0.001 s, sum of pauses 0.536 ms, longest pause  
0.536 ms.
```

Listing 8.3: Logdatei: Information Young-Collection

Der Abschluss einer Young-Collection wird mit dem oben aufgelisteten Log-Eintrag dokumentiert. Er beschreibt Start- und Endzeitpunkt sowie die Grösse des Heaps vor und nach der Garbage Collection. Des weiteren ist die Dauer, die Summe aller einzelnen Pausen und die längste Pause aufgelistet.

Old-Collection Information

```
[INFO ][memory ] [OC#3] 2.544-2.733: OC 233624KB->187955KB (280628  
KB), 0.189 s, sum of pauses 187.019 ms, longest pause 187.019  
ms.
```

Listing 8.4: Logdatei: Information Old-Collection

Die Ausgabe einer Old-Collection unterscheidet sich, ausgenommen vom Typ (OC), nicht von der einer Young-Collection.

Wechsel der Garbage Collection Strategie

```
[INFO ][memory ] [OC#6] Changing GC strategy from: singleconpar to:  
singleconcon, reason: Return to basic strategy.
```

Listing 8.5: Logdatei: Wechsel Garbage Collection Strategie

Aus unterschiedlichen Gründen kann es zu einem Wechsel der Garbage Collection Strategie kommen. Dieser wird, zusammen mit dem Grund für den Wechsel, in der Logdatei dokumentiert und kann ausgewertet werden.

8.9.4 Domänenmodell JRockit Garbage Collection

Der Parseprozess wandelt die unstrukturierten Daten in ein strukturiertes Domänenmodell um. Dieses wurde wie folgt erarbeitet:

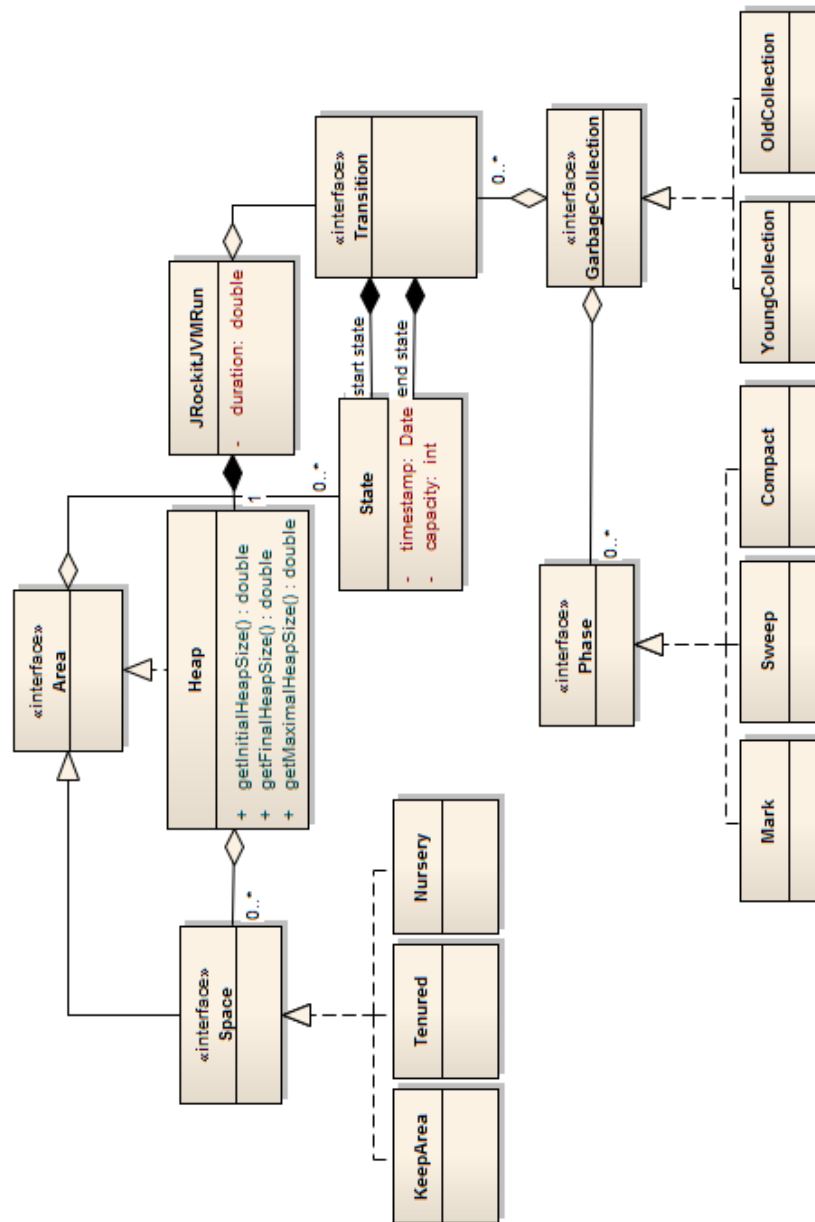


Abbildung 8.7: Domänenmodell: Garbage Collection (JRockit Implementation)

Die Daten der Log-Datei repräsentieren einen Lauf einer JVM (*JRockitJVM-Run*) bestehend aus einem Heap und den darin enthaltenen Bereichen Keep-Area, Nursery und Tenured Space. Jeder dieser Bereiche hat unterschiedliche Zustände in welchen die verschiedenen Messgrößen gemessen und gespeichert werden. Zustandsübergänge finden durch Transitionen respektive durch eine Garbage Collection statt, es kann sich dabei um Young oder Old Collections handeln. Das starten einer Transition wird durch einen Event ausgelöst (hier im Diagramm nicht ersichtlich), Events werden aufgrund von heuristischen Daten der Laufzeitumgebung geworfen - zum Beispiel wenn die Nursery oder die Old Collection an ihre Speichergrenze gelangen. Der Vollständigkeit halber sind im Diagramm zusätzlich noch die einzelnen Phasen der Garbage Collection definiert, die bei einer Garbage Collection durchlaufen werden.

8.10 Standardauswertung anzeigen (FRQ-06)

8.10.1 Anzeige Übersicht Garbage Collection (FRQ-06.1)

Der initiale Tab der Analyseseite zeigt eine Zusammenfassung der geöffneten Garbage Collection. Die Daten werden in den Tabellen Heap Kapazität, Garbage Collection Aktivität und Gesamtstatistik angezeigt:

Heap Kapazität

- **Initiale und maximale Heap Kapazität**
- **Grösse Young- und Old-Space**
- **Speicherbedarf Peak, Durchschnitt:** Aus den Informationen des Speicherverbrauchs vor und nach jeder Garbage Collection wird der durchschnittliche und der maximale Bedarf berechnet.
- **Kapazität Peak, Durchschnitt:** Aus den Informationen der Kapazität vor und nach jeder Garbage Collection wird die durchschnittliche und maximale Kapazität des Heaps berechnet.

Garbage Collection Aktivität (Young und Old Collection)

- **Letzte Garbage Collection:** Zu welchem Zeitpunkt hat die letzte Garbage Collection stattgefunden.
- **Anzahl Garbage Collections:** Wieviele Garbage Collections hat es insgesamt gegeben.
- **Anzahl Old Collections:** Wieviele Old Collections hat es gegeben.
- **Anzahl Young Collections:** Wieviele Young Collections hat es gegeben.
- **Total Zeit der Garbage Collection:** Totale Zeit in welcher sich die Virtual Machine in der Garbage Collection befunden hat.

- **Durchsatz der Applikation** (siehe Abschnitt 3.4.1)
- **Durchschnittlicher Interval in Sekunden:** Durchschnittliche Pausenzeit zwischen den einzelnen Garbage Collections.
- **Durchschnittliche Dauer in Sekunden**
- **Totale Zeit der Old Garbage Collection Zyklen**
- **Totale Zeit der Young Garbage Collection Zyklen**
- **Prozentuale Zeit der Old Garbage Collection Zyklen**
- **Prozentuale Zeit der Young Garbage Collection Zyklen**

Gesamtstatistik

- **Dauer der Messung in Sekunden**

8.10.2 Chart Anzeige Heap Benutzung (FRQ-06.2)

Die Heap-Analyse zeigt den Verlauf des benutzten Speichers im Heap über die Zeit auf. Die einzelnen Garbage Collection Zyklen (Young, Old) werden verschiedenfarbig dargestellt.

8.10.3 Chart Anzeige Dauer Garbage Collection (FRQ-06.3)

Die Dauer der einzelnen Garbage Collection Zyklen wird über die Zeit aufgezeigt. Die einzelnen Garbage Collection Zyklen (Young, Old) werden verschiedenfarbig dargestellt.

8.11 Qualitätsanforderungen

8.11.1 Testabdeckung (QRQ-S-02)

Bei Plugin-Projekten wird der Test-Code normalerweise in eigenen Fragment-Projekten abgelegt. Diese Test-Projekte werden allerdings nicht mit dem Feature ausgeliefert, sondern nur während dem Testen verwendet. Der Zugriff auf den Code der Implementation wird gewährleistet, indem aus dem Test-Projekt ein Fragment¹⁸ gemacht wird.

¹⁸Fragmente haben vollen Zugriff auf ihre Host-Bundles.

8.12 Eclipse Rich Client Framework

8.12.1 Speicherung des Zustands einer View

Die Speicherung des Zustands einer View wird von Eclipse unterstützt, indem die Methode `saveState(memento:IMemento)` von `ViewPart` implementiert wird. `IMemento` ist eine Eclipse-Klasse und gleichzeitig die Abstraktion eines Mementos. Memento ist ein Design Pattern und wurde durch die Gang of Four¹⁹ in [2, S. 283] zum ersten Mal definiert. Mementos dienen zur Serialisierung von Objekten und haben den Vorteil, dass auch Klassen aus einem Memento einer anderen Version deserialisiert werden können. Das Eclipse-Framework persistiert die Zustände von Views mittels eines `XMLMemento` entsprechend im XML-Format ab. Die Deserialisierung erreicht man mit dem Überschreiben der Methode `init(site:IViewSite, memento:IMemento)`.

8.12.2 Internationalisierung (QRQ-S-03))

Die Analysesoftware kann in den Sprachen Deutsch und Englisch gestartet werden. Die gewählte Sprache wird von der Entwicklungsumgebung übernommen²⁰ und kann nicht via ein Menu geändert werden. Basierend auf der `Locale`-Klasse der Java Laufzeitumgebung können sprachabhängige Ressourcen geladen werden. Sprachabhängig sind folgende Bereiche:

- **Texte, Labels im Code:** Eclipse stellt zur Externalisierung von Strings einen Wizard zur Verfügung. Die Texte werden in Properties-Dateien extrahiert und beim Starten der Applikation geladen.
- **Texte, Labels in Deskriptoren:** Die sich in den Eclipse-Deskriptoren (`plugin.xml` und `Manifest.MF`) befindenden sprachabhängigen Texte wie Organisation, Plugin-Name und -Beschreibung werden ebenfalls in Properties-Dateien extrahiert. Der Ort und Name dieser Dateien ist per Konvention `\${plugin}\OSGI-INF\I10n\bundle_${lang}.properties`. Der Inhalt wird vom Eclipse-Framework geladen.
- **Hilfesystem:** Eclipse startet zur Anzeige des auf HTML basierenden Hilfesystems einen Webserver. Die Hilfeseiten können ebenfalls in unterschiedlichen Sprachen definiert werden und werden auf der Basis der gewählten Locale angezeigt.

8.12.3 Usability (QRQ-S-04))

Einige der Funktionalitäten der Analysesoftware wie Beispielsweise das Einlesen und Parsen der Logdateien dauern lange. Der Benutzer benötigt eine Statusinformation. Operationen dieser Art werden mittels des Eclipse `IProgressService` gestartet. Dies hat für die Applikation und den Benutzer folgende Vorteile:

¹⁹Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

²⁰Die Entwicklungsumgebung übernimmt die Sprache der Java Laufzeitumgebung: Voreinstellung oder Auswahl über Kommandozeile (-nl de).

- **Nebenläufigkeit:** Die Applikation startet die Arbeit in einem eigenen nicht-UI Thread²¹, sodass es nicht zu Nebeneffekten wie einem eingefrorenen Bildschirm kommt. Der Benutzer kann die Fortschrittsanzeige minimieren und mit der Applikation weiterarbeiten.
- **Fortschrittsanzeige:** Die Applikation teilt dem Benutzer über eine Anzeige mit, bei welcher Position sich der Prozess befindet und wie viel Arbeit prozentual bereits gemacht wurde.
- **Unterbrechbarkeit:** Der Prozess erkundigt sich periodisch bei der Monitoring-Komponente ob er durch den Benutzer abgebrochen wurde. Sobald dies der Fall wäre, würde er die Arbeit beenden und das bereits Erledigte aufräumen.

8.13 Infrastruktur

8.13.1 Build-Automatisierung

Die Automatisierung des Software-Builds ist hinsichtlich der Integration in ein Continuous Integration System wichtig. Zusätzlich entfallen so zeitaufwändige Tasks wie die Paketierung und das Deployment der Applikation. Als Werkzeug zum automatisierten Build der Software wird Maven Tycho²² verwendet.

Tycho ist relativ neu und bringt im Vergleich mit dem PDE Build einige Vorteile mit sich:

- Maven folgt dem Prinzip “Convention over Configuration”²³ - die Konfiguration des Builds wird dadurch wesentlich einfacher.
- Maven ist de facto Standard bei den Build-Werkzeugen.

8.13.2 Versionskontrolle

Zur Versionsverwaltungssysteme kommen mehrere Werkzeuge in Frage. Git²⁴ ist ein verteiltes Sourcecode Management System und ist konzeptionell und hinsichtlich Benutzerfreundlichkeit besser als Subversion und CVS²⁵. Auf der Plattform Github²⁶ kann man öffentliche Projekte gratis “hosten”.

²¹Einem Thread der nicht für das Zeichnen des Benutzerinterfaces verwendet wird.

²²Im Bereich der Eclipse Rich Client Entwicklung kann entweder PDE Build, ein auf Apache Ant basiertes Build-System für Eclipse RCP Applikationen[13] oder die Maven-Integration Tycho (<http://tycho.sonatype.org>) verwendet werden.

²³Das Prinzip “Convention over Configuration” hat zur Folge, dass im Wesentlichen nur von den Standardeinstellungen abweichende Werte konfiguriert werden müssen.

²⁴<http://git-scm.com>

²⁵Git kann offline verwendet werden, das Verschieben von Verzeichnissen führt nicht zu Problemen, etc.

²⁶<http://github.com>

8.13.3 Continuous Integration

Continuous Integration Systeme dienen zur Steigerung der Softwarequalität. Sie machen dies in der Regel, indem sie alle Tests und den Gesamtbuild der Software periodisch - üblich ist jede Stunde - durchführen. Für die Analysesoftware gibt es bei der Evaluation einige Grundvoraussetzungen:

- **Buildwerkzeug Maven:** Das System muss Maven als Build- und Automatisierungswerkzeug unterstützen.
- **Git Versionskontrolle:** Der Quelltext der Applikation muss via Git vom Sourcecode Repository ausgecheckt werden können.
- **Freie Lizenz:** Die Software muss mindestens frei verfügbar sein oder open-source.

In den letzten Jahren hat sich Hudson²⁷ in vielen Projekten durchgesetzt. Hudson ist eine open-source Continuous Integration Software die gegenüber anderen Systemen einige Vorteile mit sich bringt:

- Hudson ist open-source.
- Hudson ist sehr einfach zu installieren und administrieren.
- Hudson basiert auf einem Plugin-System und ist aufgrund dessen erweiterbar. Es gibt Plugins für die Integration von Maven-Projekten und den Zugriff auf Git Repositories.

8.13.4 Issue Tracker

Als Issue Tracker wird Jira verwendet. Es handelt sich dabei um eine kostenpflichtige aber relativ günstige Software für das Issue-Tracking.

²⁷Vor kurzem haben sich einige Entwickler von Hudson aufgrund von Streitigkeiten mit Oracle dazu entschieden, die Software weiter unter dem Namen Jenkins zu entwickeln.

Kapitel 9

Implementation

Im Proof of Concept wurden die Anforderungen folgendermassen umgesetzt.¹:

9.1 Funktionale Anforderungen

Die funktionalen Anforderungen konnten alle umgesetzt werden. Dieser Abschnitt beschreibt die Punkte im Detail:

9.1.1 Installation (FRQ-01) und Update (FRQ-02)

Installation und Update der beiden Features Core und JRockit Extension können über die Update-Seite gemacht werden. Beide Funktionalitäten sind Bestandteil des Eclipse-Frameworks und werden via ein Update-Site-Plugin und die beiden Feature-Plugins bereitgestellt.

9.1.2 Datei importieren (FRQ-03)

Die Dateien werden in eine eigens dafür registrierte View importiert. Das Speichern und Wiederherstellen des Zustands dieser View findet über das von Eclipse implementierte Memento-Pattern (siehe Abschnitt Speicherung des Zustands einer View) statt.

9.1.3 Datei einlesen (FRQ-04)

Die eingelesene Datei wird in Form des im Konzept beschriebenen Domänenmodells ins Memory geladen.

¹Kann auch im Abschnitt Bedienungsanleitung oder an der laufenden Software überprüft werden.

9.1.4 Garbage Collection Logdatei parsen (FRQ-05)

Die Log-Ausgaben des Memory-Moduls (Debug-Level: info) werden mittels Regulären Ausdrücken geparkt und in strukturierter Form gespeichert. Strukturiert meint eine objektorientierte Form der Daten (siehe Abschnitt Domänenmodell JRockit Garbage Collection).

9.1.5 Standardauswertung anzeigen (FRQ-06)

Für die importierten Dateien besteht die Möglichkeit, sie in der Standardauswertung darzustellen. Die Standardauswertung zeigt die im Konzept definierten Ansichten, Tabs (siehe Abschnitt Anzeige Übersicht Garbage Collection (FRQ-06.1) auf Seite 71).

9.1.6 Profil erstellen (FRQ-07)

Profile können durch den Benutzer erstellt und an seine Bedürfnisse angepasst werden. Das Anpassen beinhaltet die Definition von eigenen Charts und deren Datenserien. Das Speichern, Exportieren und Importieren wird über das von Eclipse implementierte Memento-Pattern (siehe Abschnitt Speicherung des Zustands einer View) gemacht.

9.1.7 Hilfesystem (FRQ-08)

Das Hilfesystem wurde sowohl für die contextsensitive wie auch die indexbasierte Hilfe angelegt. Aktuell sind alle existierenden Hilfeseiten in den Sprachen Deutsch und Englisch vorhanden, diese können aber ohne Entwicklungsaufwand auch um weitere Sprachen ergänzt werden.

9.2 Qualitätsanforderungen Software

9.2.1 Erweiterbarkeit (QRQ-S-01)

Die Analyse der JRockit Dateien wurde als Erweiterung implementiert. Es besteht die Möglichkeit, dass sich Erweiterungen für andere Log-Dateien bei der Basissoftware registrieren, die dann das Parsen und Aufbereiten der Datei durchführt und die Analysen durchführt.

9.2.2 Testabdeckung (QRQ-S-02)

Die Testabdeckung entspricht nicht den Anforderungen von 80 Prozent.

9.2.3 Internationalisierung (QRQ-S-03)

Viele der eingebauten Namen und Labels sind bereits zweisprachig (Englisch, Deutsch) definiert. Die restlichen müssen noch übersetzt und in die Sprachressourcen extrahiert werden.

9.2.4 Usability (QRQ-S-04)

Lange dauernde Operationen (Einlesen, Parsen der Daten) werden vom Eclipse-Framework asynchron gestartet. Dem Benutzer wird ein Progress-Monitor angezeigt, in welchem er die Operation auch abbrechen kann.

9.2.5 Korrektheit (angezeigte Werte) (QRQ-S-05)

Um mit Java-Applikationen genaue Werte zu berechnen wird von [1] empfohlen, die Klasse *BigDecimal* anstelle von *Double* und *Long* zu verwenden. Die in der Analysesoftware verwendeten Werte werden mit einer Genauigkeit von 0.1 gerechnet.

Kapitel 10

Review und Ausblick

10.1 Was leistet das Tool?

Während einer Performanceanalyse kommt es zur Auswertung der Garbage Collection, wenn die CPU-Last hoch ist, aber nicht durch den Kernel (Systemprozess) verursacht wird. Der Dominating Consumer ist dann die Java Virtual Machine (siehe Abschnitt Suche nach dem Dominating Consumer auf Seite 15). Die folgenden Abschnitte zeigen wann und wie die Analysesoftware verwendet werden kann.

10.1.1 Offline-Analyse

Zur Auswertung einer laufenden Virtuellen Machine, kann auch die Software JRockit Mission Control von Oracle verwendet werden. Oft ist allerdings der Zugriff via Mission Control auf den Server nicht möglich - es gibt Firewall-Regeln oder man befindet sich ausserhalb des Netzwerkes. In diesem Fall müssen zur Analyse die erstellten Garbage Collection Logdateien verwendet werden. **Mit der Analysesoftware kann diese Aufgabe vereinfacht werden indem sie die gesammelten Daten visualisiert.**

10.1.2 Log Module

Aktuell werden die Log-Einträge des **Memory-Moduls (Log-Level: INFO)** ausgewertet. Das sind die wichtigsten Ausgaben die im Zusammenhang mit der Garbage Collection geschrieben werden - aber nicht alle. Um eine Auswertung einer Logdatei zu machen, muss das Logging für das Memory-Modul über das Argument `-Xverbose:memory` aktiviert werden.

10.1.3 JRockit Version

Version R28 ist die neuste Version der JRockit. Die Analysesoftware wurde auf das Format dieser Virtuellen Machine ausgerichtet. Die Analyse von Logdateien

älterer Versionen ist momentan noch nicht möglich.

10.2 Ausblick

10.2.1 Analyseumfangs JRockit R28

Aktuell werden nur die Log-Einträge des Memory-Moduls (Log-Level: INFO) berücksichtigt. Wie der nachfolgende Auszug einer Logdatei zeigt, würden in den Debug-Einträgen spannende und teilweise wichtige Informationen stehen:

- **Gründe, warum eine Garbage Collection gestartet wurde:** Die erste Zeile zeigt, dass es hängige Anfragen für die Allokation von Speicher gibt.
- **Dauer der einzelnen Garbage Collection Phasen (Initial Marking, Precleaning, Final Marking):** Nicht alle dieser Phasen laufen beispielsweise konkurrierend ab. Im Sinne von möglichst kurzen Pausenzeiten ist es deshalb interessant, wie lange die Final Marking Phase dauert.

```

1 [INFO ][alloc  ] [OC#1] Satisfied 0 object and 0 tla allocations.
   Pending requests went from 1 to 1.
2 [DEBUG][memory ] [OC#1] Initial marking phase promoted 3620 objects
   (206KB).
3 [DEBUG][memory ] [OC#1] Starting concurrent marking phase (OC2).
4 [DEBUG][memory ] [OC#1] Concurrent mark phase lasted 0.235 ms.
5 [DEBUG][memory ] [OC#1] Starting precleaning phase (OC3).
6 [DEBUG][memory ] [OC#1] Precleaning phase lasted 0.249 ms.
7 [DEBUG][memory ] [OC#1] Starting final marking phase (OC4).
8 [INFO ][nursery] [OC#1] Young collection started. This YC is a part
   of OC#1 final marking.
```

Listing 10.1: Garbage Collection Log (Debug Informationen)

10.2.2 Analyseumfangs JRockit R27

Wie bereits erwähnt, können die Logs der Version R27 noch nicht ausgewertet werden, obwohl diese Version noch an einigen Orten im Einsatz ist. Ziel wäre die möglichst rasche Kompatibilität zur Version R27.

10.2.3 G1 Algorithmus

Ab Version 1.6.0_14 des Java Runtime Environments ist eine Vorversion des G1 Garbage Collectors¹ verfügbar. Die Funktionsweise dieses Algorithmus unterscheidet sich stark von den bisherigen Versionen des Mark & Sweep Algorithmus, auch die Logdateien weisen Unterschiede auf. Für die Auswertung solcher Dateien gibt es aktuell noch kein Werkzeug, die Implementation dieses Formats wäre deshalb spannend.

¹G1 ist auch unter dem Namen Garbage First Garbage Collector bekannt.

Glossar

Wort	Beschreibung	Herkunft
Continuous Integration	Kontinuierliches Kompilieren, Bilden und Testen einer Applikation. Hilft einem oder mehreren Entwicklern eine bessere Softwarequalität zu erreichen.	-
Bundle (Eclipse)	siehe Plugin	siehe Plugin
Category (Eclipse)	Auf einer Eclipse Update-Seite werden Features zur Verfügung gestellt. Diese können logisch noch einmal in Kategorien unterteilt werden.	Eclipse
Feature (Eclipse)	Als Feature verpackt man im Eclipse-Umfeld eine logische Einheit an Funktionalität.	Eclipse
Fragment	Manchmal macht es im Eclipse-Umfeld Sinn, gewisse Teile der Applikation optional zu definieren. In diesem Fall verwendet man Fragmente. Sie erlauben die Erweiterung eines bestehenden Bundles (Host-Bundle) und haben Zugriff auf alle auch nicht exportierten Pakete. Test-Projekte werden oft auch als Fragmente definiert, da ihnen der volle Zugriff auf das Host-Bundle gewährleistet wird.	Eclipse
Data Binding	Data Binding ist ein Mechanismus in Client Applikationen, um die hinterlegten Werte eines Bedienelementes mit dem hinterlegten View-Model zu verbinden.	-
Garbage Collection	Der Begriff Garbage Collection bezeichnet das Aufräumen von nicht mehr benutzten Objekten im Speicher.	Speichermanagement

Garbage Collection Algorithmus	Das Aufräumen von Speicher wird je nach Technologie mittels unterschiedlicher Algorithmen gemacht.	Speichermanagement
Graphical User Interface (GUI)	Im Unterschied zu einer textbasierten Benutzeroberfläche ist ein GUI grafisch. Die Bedienung findet nebst der Tastatur mit der Maus statt.	-
Old Collection	Der Begriff hat zwei Bedeutungen und wird je nach Kontext unterschiedlich verwendet: <ul style="list-style-type: none"> • Der Bereich im Heap in welchem sich die alten Objekte befinden (siehe Old Generation). • Das Aufräumen von alten Objekten. 	Speichermanagement
Old Generation	Bei einigen Garbage Collection Algorithmen wird der Heap in Generationen unterteilt. Der Bereich mit den jungen Objekten wird Old Generation genannt.	Speichermanagement
Parseprozess	Bezeichnet die Interpretation und Aufbereitung von unstrukturierten Daten in eine strukturierte Form (Domänenmodell, Liste, Key-Value Datenstruktur, etc.).	Compilerbau
Plugin (Eclipse)	Ein Plugin ist eine technische Trennung von gewissen logischen Softwareteilen. Dabei definiert man die Schnittstelle zu anderen Plugins mittels einer Manifest.mf respektive einer plugin.xml Datei. Diese definiert die Abhängigkeiten (Import, Required Bundles inklusive den jeweiligen Versionen), die Exportierten Klassen und die Extension Points	Eclipse

Rich Client Plattform	Der Begriff Rich Client Plattform wird in diesem Dokument als Synonym für Desktop- respektive Client-Applikation verwendet.	Vor gut 10 Jahren verlagerte sich die Logik vom Client auf den Server. Jede Interaktion mit dem System fand über eine Verbindung zum Server statt, der Client stellte die Inhalte nur dar. Die Anwendungen waren wenig benutzerfreundlich. Es gab deshalb eine Gegenbewegung zu den Rich Client Applikationen, dabei wird mindestens ein Teil der Logik wieder in den Client verlagert.
Update Site	Eine Update Site ist eine per Eclipse-Konvention aufgebaute Webseite die via das Http-Protokoll zugänglich ist und Software-Features enthält.	Eclipse
Virtual Machine (JRockit, HotSpot)	Laufzeitumgebung die beispielsweise eine Java Applikation ausführt.	Softwareentwicklung
Wizard	Dialog innerhalb einer Anwendung, der den Benutzer durch einen Prozess führt.	-
Young Collection	Der Begriff hat zwei Bedeutungen und wird je nach Kontext unterschiedlich verwendet: <ul style="list-style-type: none"> • Der Bereich im Heap in welchem sich die jungen Objekte befinden (siehe Young Generation). • Das Aufräumen von jungen Objekten. 	Speichermanagement
Young Generation	Bei einigen Garbage Collection Algorithmen wird der Heap in Generationen unterteilt. Der Bereich mit den jungen Objekten wird Young Generation genannt.	Speichermanagement
User Interface	Als User Interface (Benutzeroberfläche) werden alle Schnittstellen zwischen Mensch und Maschine bezeichnet. Diese können textbasiert oder grafisch sein (GUI).	

Tabelle 10.1: Glossar

Abkürzungen

Abkürzung	Begriff	Beschreibung
BIRT	Business Intelligence and Reporting Tool	Open-Source-Framework für Business Intelligence und Reporting Anwendungen
GC	Garbage Collection	siehe Glossar
GUI	Graphical User Interface	siehe Glossar
I/O	Input / Output	Eingabe und Ausgabe - der Begriff wird meist im Zusammenhang mit Daten verwendet.
JVM	Java Virtual Machine	siehe Glossar (Java Virtual Machine)
Lap	Log Analysis Profile	Die exportierten Profile werden in Lap-Dateien (Dateien mit der Endung .lap) gespeichert.
RCP	Rich Client Plattform	siehe Glossar
UUT	Unit under Test	Von Unit under Test spricht man bei der zu testenden Klasse.
UI	User Interface	siehe Glossar
VM	Virtual Machine	siehe Glossar

Tabelle 10.2: Glossar

Literaturverzeichnis

- [1] Joshua Bloch. Effective Java. Addison-Wesley Java series. Addison-Wesley, 2008.
- [2] Erich Gamma, Richard. Helm, Ralph Johnson, and John Vlissides. Design patterns: elements of reusable object-oriented software. Addison-Wesley professional computing series. Addison-Wesley, 1995.
- [3] Adrian Hummel and Michael Beer. Java performance analysis. http://blog.trivadis.com/cfs-file.ashx/_key/communityserver-blogs-components-weblogfiles/00-00-00-00-80-slides/7343.Trivadis_5F00_JavaLounge_5F00_JavaPerformanceAnalysis.pdf, 2011.
- [4] Marcus Lagergren and Marcus Hirt. Oracle Jrookit: The Definitive Guide. Packt Publishing, Limited, 2010.
- [5] Angelika Langer and Klaus Kreft. Generational garbage collection. Java Magazin, 3:26–30, 2010.
- [6] Angelika Langer and Klaus Kreft. Mark-and-compact. Java Magazin, 7:20–24, 2010.
- [7] Angelika Langer and Klaus Kreft. Young generation garbage collection. Java Magazin, 5:24–30, 2010.
- [8] Angelika Langer and Klaus Kreft. Java Core Programmierung. Entwickler Press, 2011.
- [9] Sun Microsystems. Memory management in the hotspot virtual machine. http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf, 2006.
- [10] Oracle. Oracle jrookit command-line reference. http://download.oracle.com/docs/cd/E15289_01/doc.40/e15062.pdf, 2011.
- [11] Kerk Pepperdine. Concurrent and performance reloaded. <http://www.jfokus.se/jfokus/page.jsp?id=recordings#page=page-1>, 2011.

- [12] Klaus Pohl and Chris Rupp. Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering– Foundation Level. Dpunkt.Verlag GmbH, 2010.
- [13] Lars Vogel and Dominik Zapf. Eclipse pde build - tutorial. <http://www.vogella.de/articles/EclipsePDEBuild/article.html#overview>, 2008.
- [14] Wikipedia. Chain-of-responsibility pattern — wikipedia, the free encyclopedia, 2011. [Online; accessed 11-September-2011].
- [15] Wikipedia. Eclipse (ide) — wikipedia, die freie enzyklopädie. [http://de.wikipedia.org/w/index.php?title=Eclipse_\(IDE\)&oldid=92563983](http://de.wikipedia.org/w/index.php?title=Eclipse_(IDE)&oldid=92563983), 2011. [Online; Stand 29. August 2011].

Listings

5.1	Format Aktivierung Log Modul	29
5.2	Garbage Collection Log (Info)	29
5.3	Garbage Collection Log (Info) - Umleitung in gc.log	29
5.4	Einstellung des Log-Levels	29
5.5	Garbage Collection Log	30
8.1	Logdatei: Ausgabe initialer Garbage Collection Algorithmus . . .	68
8.2	Logdatei: Initiale und maximale Heap-Kapazität und Grösse des Old- und Young-Spaces	68
8.3	Logdatei: Information Young-Collection	69
8.4	Logdatei: Information Old-Collection	69
8.5	Logdatei: Wechsel Garbage Collection Strategie	69
10.1	Garbage Collection Log (Debug Informationen)	80
B.1	Checkout Quelltext Repository	102

Abbildungsverzeichnis

3.1	Suche nach dem Dominating Consumer	15
6.1	System und Systemkontext	33
6.2	Übersicht der Stakeholder	34
6.3	Systemfunktionalität als Use-Case-Diagramm	42
8.1	Architektur: Komponentendiagramm	59
8.2	Sequenz-Diagramm Öffnen der Analyse	61
8.3	Domänenmodell: Eingeleseene Datei	63
8.4	Domänenmodell: Profile	65
8.5	Sequenzdiagramm Parsen Logdatei	67
8.6	Klassendiagramm Parsen Logdatei	67
8.7	Domänenmodell: Garbage Collection (JRockit Implementation)	70
A.1	Installation Garbage Collection Log Analyse	91
A.2	Update Garbage Collection Log Analyse	92
A.3	Dashboard	93
A.4	Profil importieren	94
A.5	Profil erstellen	95
A.6	Profil exportieren	96
A.7	Profil importieren	97
A.8	Standardauswertung: Zusammenfassung	98
A.9	Standardauswertung: Heap Analyse	99
A.10	Standardauswertung: Dauer Garbage Collection	100
A.11	Standardauswertung: Zusammenfassung	101

Tabellenverzeichnis

5.1	Übersicht der Garbage Collection Modi	28
5.2	Übersicht der Garbage Collection Algorithmen	28
5.3	Beschreibung der verschiedenen relevanten Log Modulen	31
6.1	Funktionale Anforderungen	38
6.3	Qualitätsanforderungen Basisframework	41
6.4	Use-Case: Software installieren	43
6.5	Use-Case: Software updaten	44
6.6	Use-Case: Garbage Collection Logdatei importieren	45
6.7	Use-Case: Zustand Ansicht Logdateien speichern.	45
6.8	Use-Case: Garbage Collection Logdatei einlesen	46
6.9	Use-Case: Garbage Collection Logdatei importieren	47
6.10	Use-Case: Standardauswertung anzeigen	47
6.11	Use-Case: Anzeige Statistik Übersicht	48
6.12	Use-Case: Anzeige Heap Benutzung	49
6.13	Use-Case: Anzeige Dauer Garbage Collection	49
6.14	Use-Case: Profil (benutzerdefinierte Auswertung) erstellen	50
6.15	Use-Case: Chart definieren	50
6.16	Use-Case: Profil speichern, löschen	50
6.17	Use-Case: Auswertungsprofil exportieren	51
6.18	Use-Case: Auswertungsprofil importieren	51
6.19	Use-Case: Hilfesystem	52
7.1	Schema Vergabe der Punkte	53
7.2	Schema Umwandlung Priorität in Gewicht	53
7.3	Auswertung Rich Client Frameworks	55
7.4	Auswertung Charting-Bibliothek	56
10.1	Glossar	83
10.2	Glossar	84
B.1	Inhalt Datenträger	102

Teil III

Anhang

Anhang A

Bedienungsanleitung

A.1 Installation der Software

Eclipse Installationsdialog mit Menu *Hilfe / Neue Software installieren* öffnen.

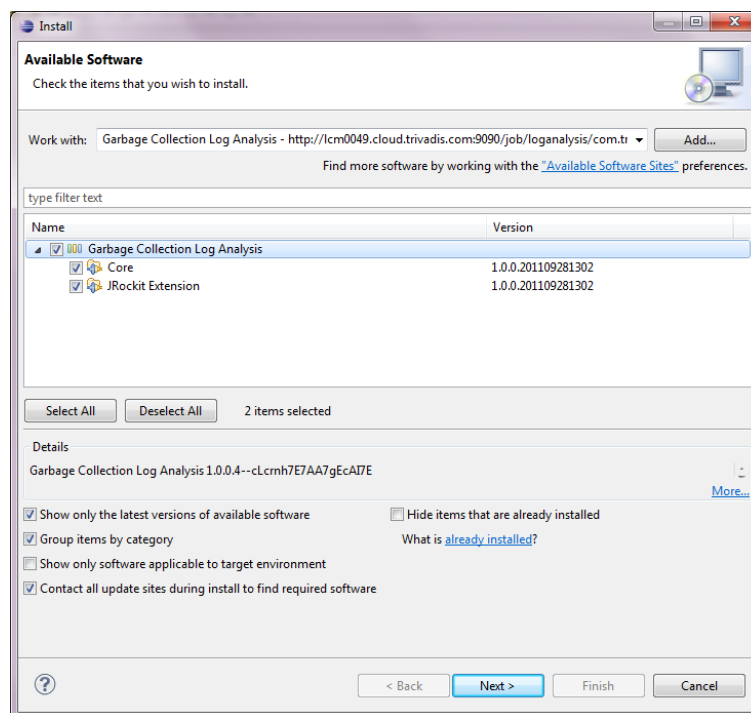


Abbildung A.1: Installation Garbage Collection Log Analyse

Durch Angabe der Update-Seite können danach beide Features ausgewählt

werden. Nach zweimaligem Klick auf *Weiter* und anschließender Bestätigung der Lizenzbestimmungen wird die Software installiert. Danach muss die Entwicklungsumgebung neu gestartet werden.

A.2 Update

Wenn ein Update der Software verfügbar ist, kann via *Hilfe / Nach Updates suchen* das Update-Fenster geöffnet werden. Die Softwarepakete werden wenn gewünscht heruntergeladen und aktualisiert.

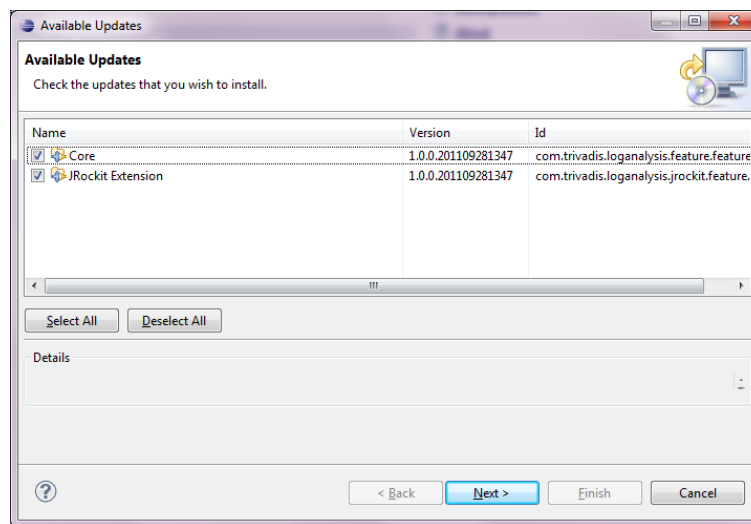


Abbildung A.2: Update Garbage Collection Log Analyse

A.3 Dashboard

Nach der Installation kann über den Knopf *GC Log Analyse Dashboard* in der Toolbar das Dashboard geöffnet werden.

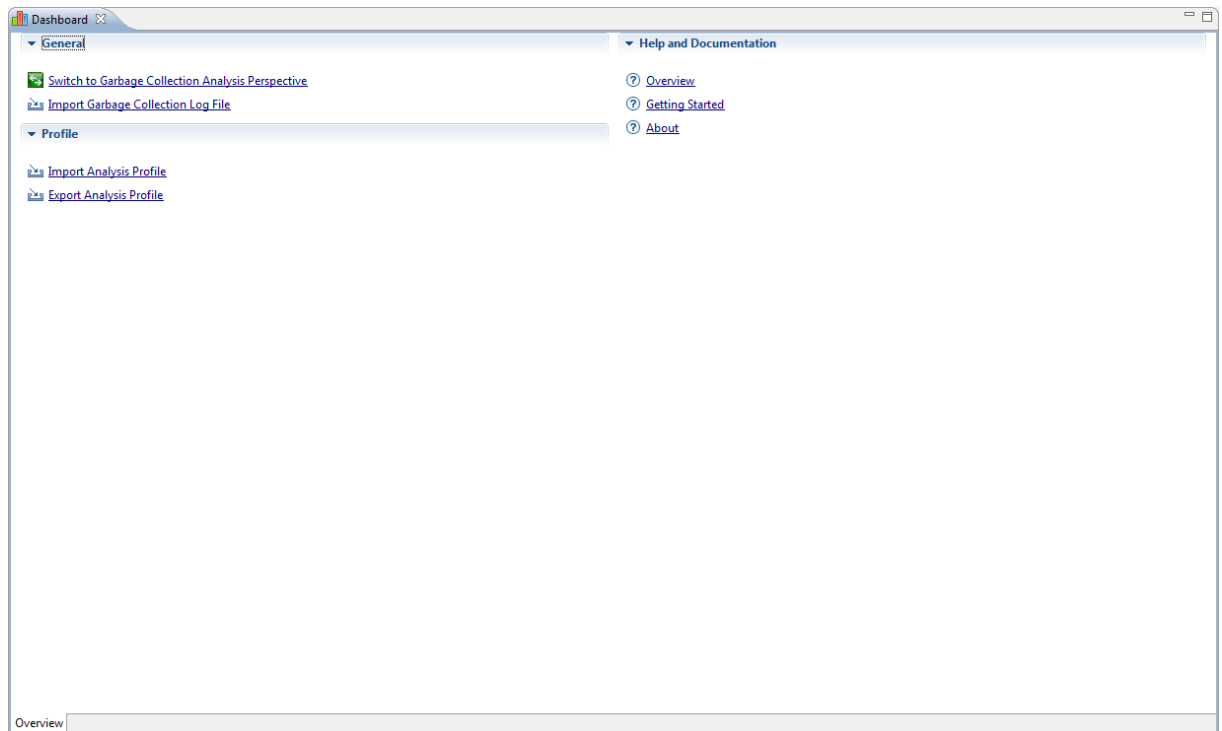


Abbildung A.3: Dashboard

Auf dem Dashboard befinden sich verschiedene Abschnitte mit unterschiedlichen Funktionen:

- Generell
 - Wechsel in die Garbage Collection Perspektive
 - Import einer Garbage Collection Logdatei
- Profile
 - Import eines Analyseprofils
 - Export eines Analyseprofils
- Hilfe und Dokumentation
 - Beinhaltet Links auf verschiedene Inhalte der Hilfe

A.4 Import einer Garbage Collection Logdatei

Der Import-Wizard kann auf verschiedene Arten geöffnet werden:

- Über Rechtsklick auf die View *Logdateien / Import Garbage Collection Logdatei*.
- Über *File / Import / Import Garbage Collection Logdatei* (Kategorie: Garbage Collection)
- Im Dashboard über *Import Garbage Collection Logdatei*

Über den *Browse* Button muss der Ordner angegeben werden, welcher die Logdateien beinhaltet. Anschliessend kann die Logdatei im darunterliegenden Fenster selektiert werden. Mit Klick auf *Finish* wird die Datei importiert.

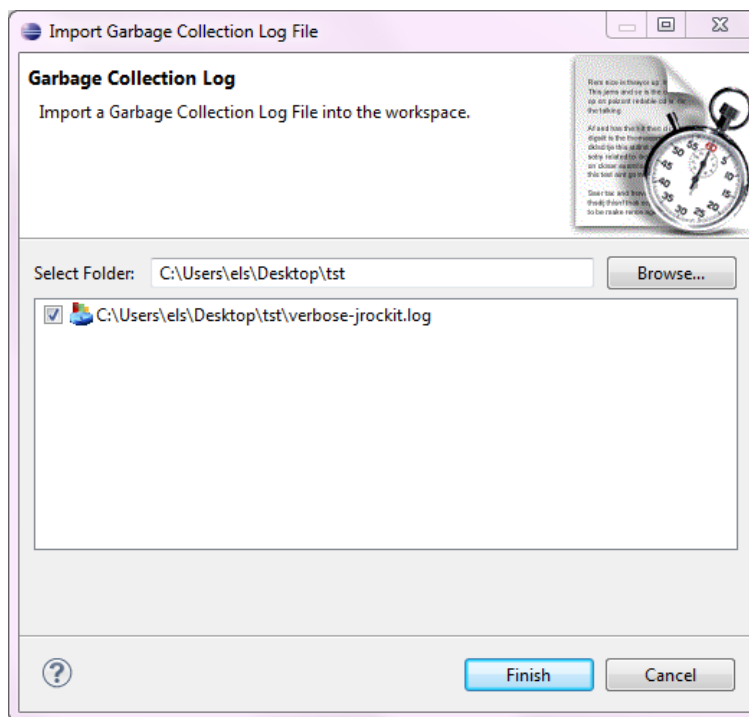


Abbildung A.4: Profil importieren

A.5 Profile

Zur Verwaltung der verschiedenen Analyseprofile gibt es die View *Profile*. In ihr können Profile erstellt, exportiert und importiert werden. Wenn eine Logdatei mit einem bestimmten Analyseprofil geöffnet werden soll, muss das entsprechende Profil darin selektiert sein.

A.5.1 Profil erstellen

Mit Rechtsklick auf die Ansicht *Profile* / *Profil erstellen* oder *Datei* / *Neu* / *Andere...* / *Analyse Profil* (Kategorie Garbage Collection) kann der Dialog zum Erstellen eines neuen Profils geöffnet werden.

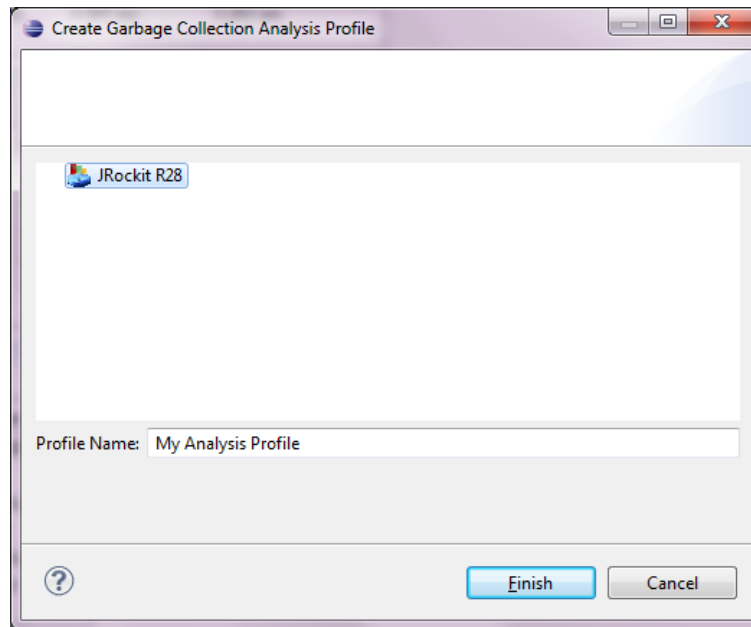


Abbildung A.5: Profil erstellen

Sofern die Erweiterung, für die das Profil erstellt wird, selektiert und ein Name vergeben ist, wird mittels Klick auf *Fertigstellen* das Profil angelegt.

A.5.2 Profil exportieren

Mit Rechtsklick auf die Ansicht *Profile* und Auswahl von *Profil exportieren*, können die Profile in eine Lap¹-Datei exportiert werden.

¹Lap steht für Log Analysis Profile

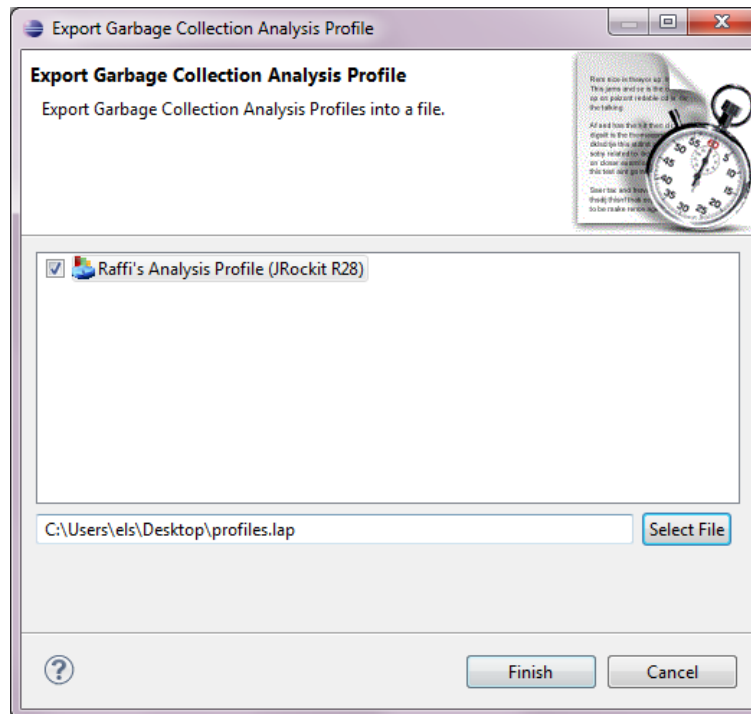


Abbildung A.6: Profil exportieren

A.5.3 Profil importieren

Mit Rechtsklick auf die Ansicht *Profile* / *Profil Importieren* können die Profile aus einer Lap-Datei importiert werden.

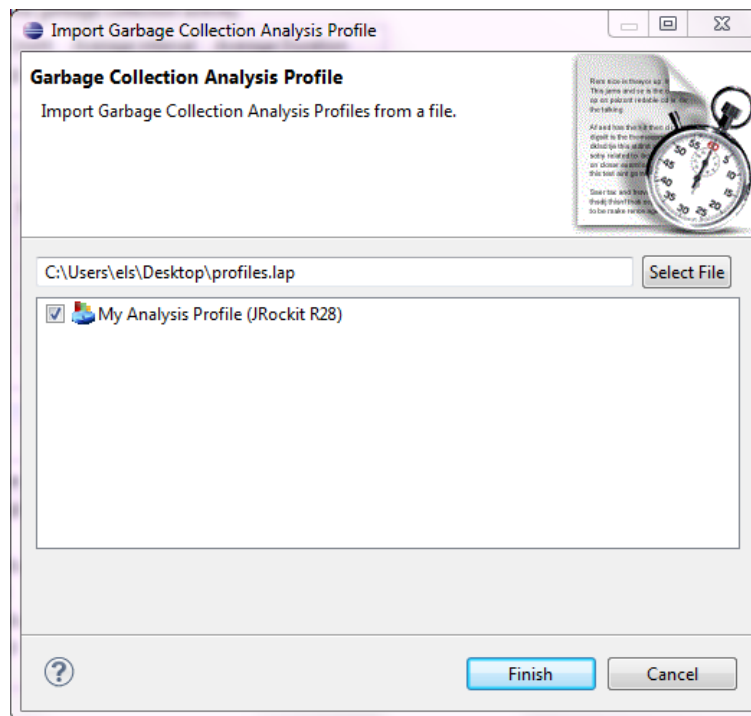


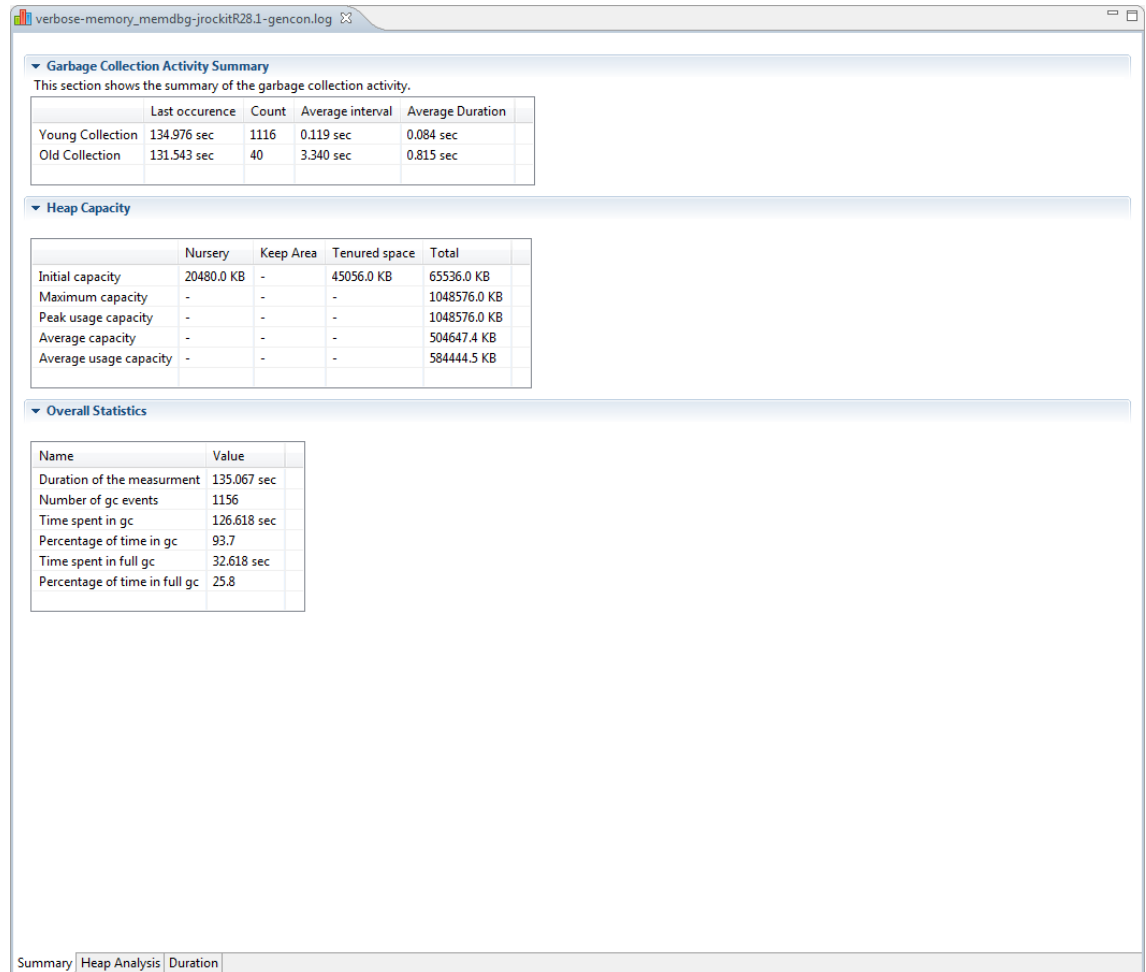
Abbildung A.7: Profil importieren

A.6 Garbage Collection Analyse

A.6.1 Standardauswertung

Sofern in der Ansicht *Profile* kein Profil selektiert ist, wird mit Doppelklick auf eine importierte Logdatei die Standardauswertung geöffnet. Die Standardauswertung besteht aus drei unterschiedlichen Tabs. Der erste Tab zeigt eine Zusammenfassung der Garbage Collection, der zweite Tab den Verlauf des benötigten Speichers auf dem Heap über die Zeit, und der dritte Tab die Dauer der einzelnen Garbage Collection Zyklen über die Zeit.

Zusammenfassung



▼ Garbage Collection Activity Summary
This section shows the summary of the garbage collection activity.

	Last occurrence	Count	Average interval	Average Duration
Young Collection	134.976 sec	1116	0.119 sec	0.084 sec
Old Collection	131.543 sec	40	3.340 sec	0.815 sec

▼ Heap Capacity

	Nursery	Keep Area	Tenured space	Total
Initial capacity	20480.0 KB	-	45056.0 KB	65536.0 KB
Maximum capacity	-	-	-	1048576.0 KB
Peak usage capacity	-	-	-	1048576.0 KB
Average capacity	-	-	-	504647.4 KB
Average usage capacity	-	-	-	584444.5 KB

▼ Overall Statistics

Name	Value
Duration of the measurment	135.067 sec
Number of gc events	1156
Time spent in gc	126.618 sec
Percentage of time in gc	93.7
Time spent in full gc	32.618 sec
Percentage of time in full gc	25.8

Summary | Heap Analysis | Duration

Abbildung A.8: Standardauswertung: Zusammenfassung

Heap Analyse

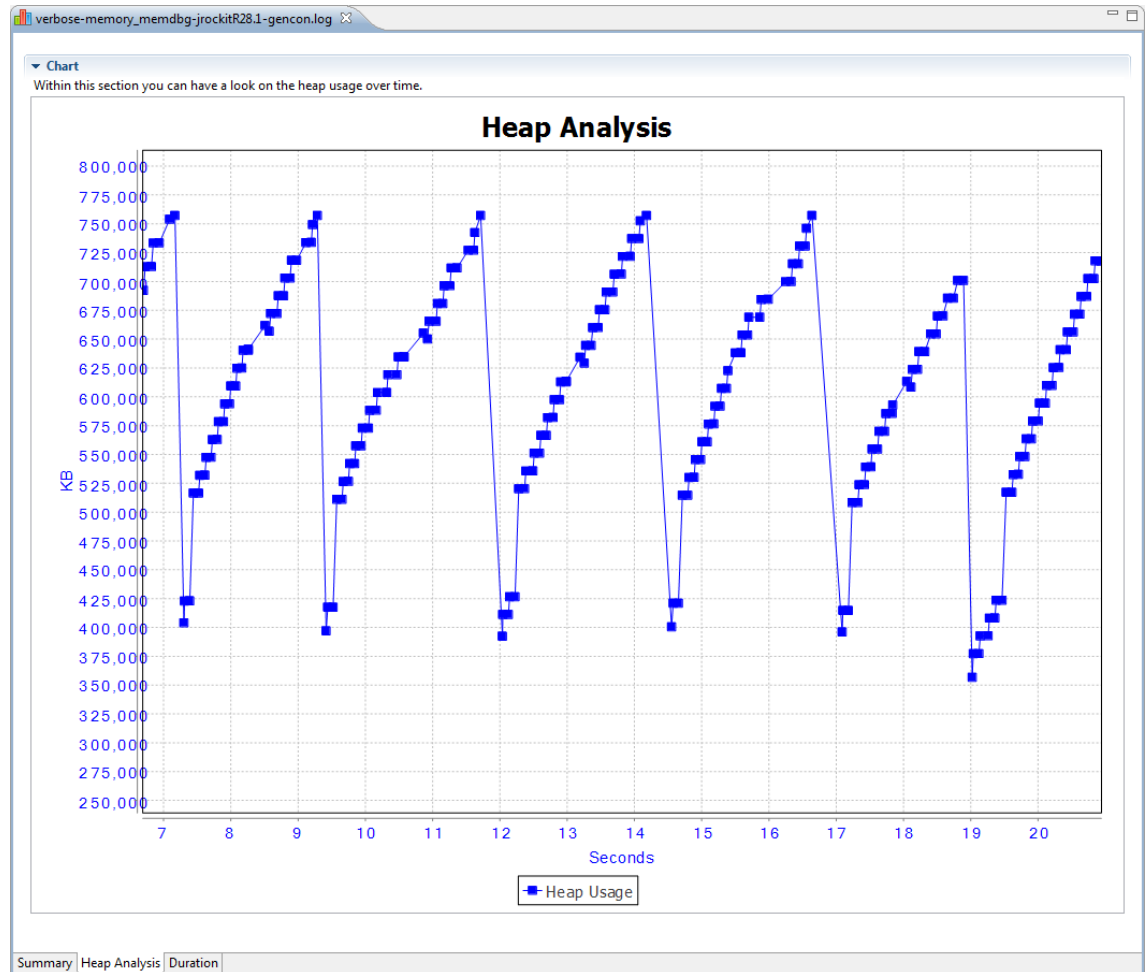


Abbildung A.9: Standardauswertung: Heap Analyse

Dauer Garbage Collection

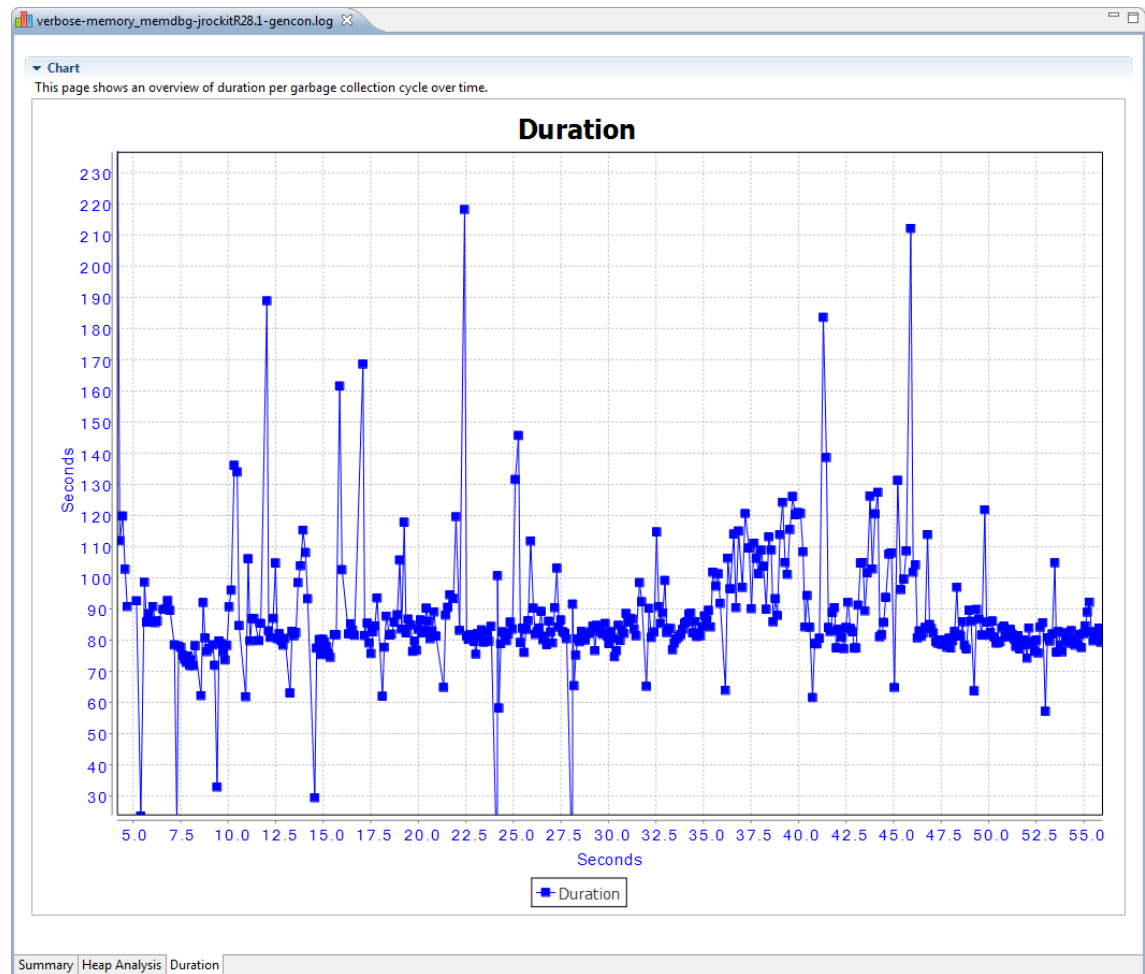


Abbildung A.10: Standardauswertung: Dauer Garbage Collection

A.6.2 Benutzerdefinierte Auswertung (Profile)

Die benutzerdefinierte Auswertung wird geöffnet, indem man vor dem Öffnen einer Datei in der Ansicht *Profile* ein Profil selektiert.

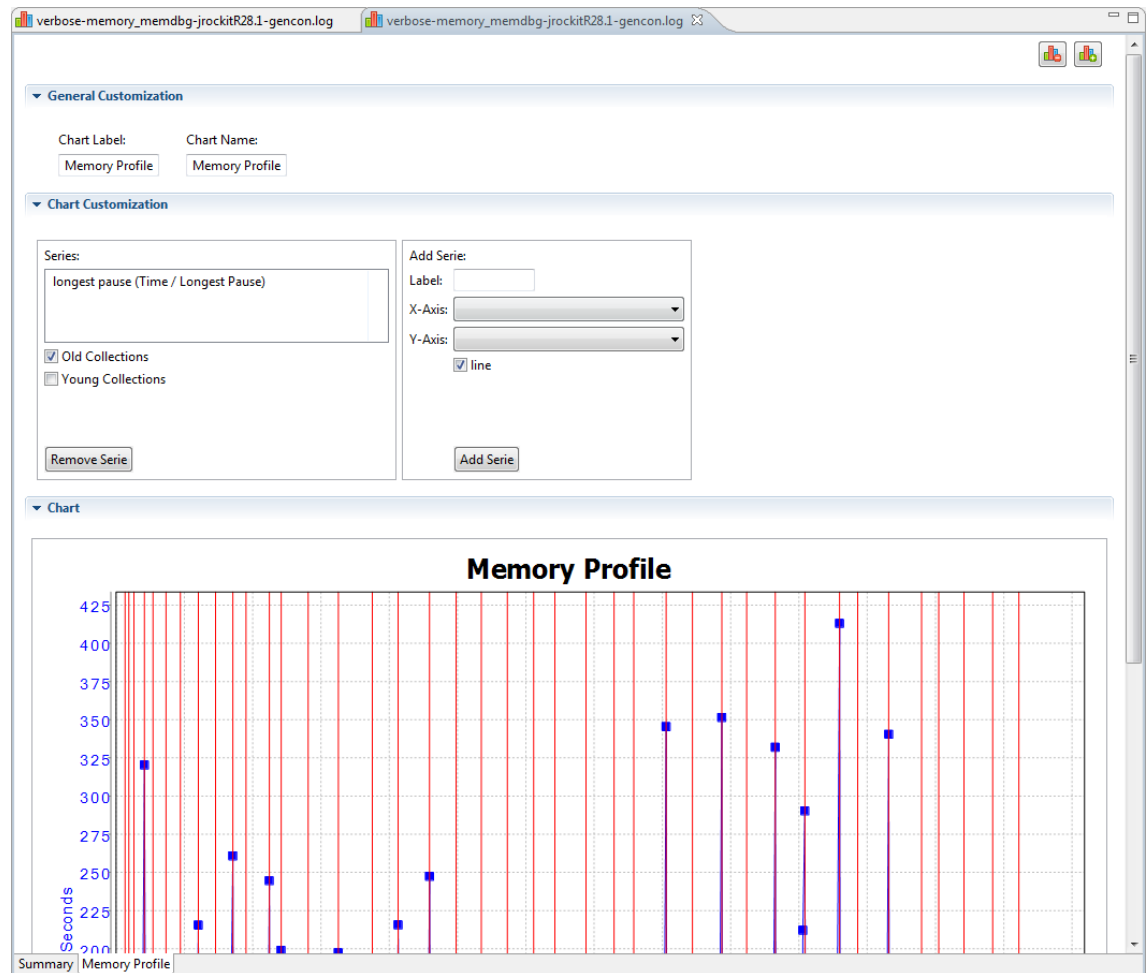


Abbildung A.11: Standardauswertung: Zusammenfassung

Im Abschnitt *Generelle Anpassungen* können die Namen von Tab und Diagramm definiert werden. Im Abschnitt *Anpassung Diagramm* können neue Serien auf das Diagramm hinzugefügt werden. Dabei können pro Serie die Werte für X-, Y-Achse und ein paar Eigenschaften angegeben werden. Das jeweilige Ende einer Garbage Collection kann als vertikale Linien in rot (Old Collections) oder blau (Young Collections) angezeigt werden.

Anhang B

Informationen

B.1 Inhalt Datenträger

Pfad	Inhalt
Dokumentation	Beinhaltet alle Dokumente, die im Zusammenhang mit der Bachelorthesis entstanden sind.
Ressourcen	Beinhaltet Dokumente die im Zusammenhang mit der Bachelorthesis hilfreich waren.
Software/development/data	Beinhaltet den Quelltext der Analysesoftware.
Software/sampling/data	Beinhaltet ein Programm zur Generierung von Garbage Collection Logdateien. Einige Beispiele solcher Dateien sind ebenfalls vorhanden.

Tabelle B.1: Inhalt Datenträger

B.2 Repository

Der Quelltext und alle Dokumente befinden sich auf Github, einem öffentlich verfügbaren Git-Repository. Mittels folgendem Kommando¹ kann alles aus dem Repository ausgecheckt werden.

```
1 git clone git://github.com/schmidic/bachelorthesis.git
```

Listing B.1: Checkout Quelltext Repository

¹Dafür ist die Installation der Software Git erforderlich.