

# Bachelorthesis

## Konzeption und Entwicklung eines Auswertungstools für die Log Files 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



# Zusammenfassung



# Inhaltsverzeichnis

<b>1</b>	<b>Glossar</b>	<b>9</b>
<b>I</b>	<b>Projektdetails</b>	<b>11</b>
<b>2</b>	<b>Aufgabenstellung</b>	<b>13</b>
2.1	Ausgangslage . . . . .	13
2.2	Ziele der Arbeit . . . . .	13
2.3	Aufgabenstellung . . . . .	14
2.4	Erwartete Resultate . . . . .	14
2.5	Geplante Termine . . . . .	15
<b>II</b>	<b>Umsetzung</b>	<b>17</b>
<b>3</b>	<b>Analyse der Aufgabenstellung</b>	<b>19</b>
3.1	Übersicht . . . . .	19
3.1.1	Erarbeitung der Grundlagen Bereich der Garbage Col- lection . . . . .	19
3.1.2	Stärken- / Schwächen-Analyse der bestehenden Rich Client Frameworks . . . . .	20
3.1.3	Durchführung einer Anforderungsanalyse für den Software- Prototypen . . . . .	20
3.1.4	Auswahl der zu verwendenden Frameworks . . . . .	20
3.1.5	Konzeption und Implementation . . . . .	20
3.1.6	Bewertung . . . . .	21
<b>4</b>	<b>Grundlagen Garbage Collection</b>	<b>23</b>
4.1	Funktionsweise . . . . .	23
4.1.1	Reference counting[1, S. 77] . . . . .	24

4.1.2	Tracing techniques[1, S. 77]	24
4.2	Algorithmen	24
4.2.1	Mark & Sweep Algorithmus[4]	24
4.2.2	Mark & Copy Algorithmus[4]	25
4.2.3	Mark & Compact Algorithmus[3]	25
4.3	Generational Garbage Collection[2]	26
<b>5</b>	<b>Garbage Collection in der Oracle JRockit Virtual Machine</b>	<b>27</b>
<b>6</b>	<b>Stärken- / Schwächen-Analyse Rich Client Frameworks</b>	<b>29</b>
<b>7</b>	<b>Anforderungsanalyse</b>	<b>31</b>
<b>8</b>	<b>Konzept</b>	<b>33</b>
<b>9</b>	<b>Implementation</b>	<b>35</b>
<b>10</b>	<b>Review</b>	<b>37</b>
<b>III</b>	<b>Anhang</b>	<b>47</b>

# Einleitung

TODO





# Kapitel 1

## Glossar

Tabelle 1.1: Glossar

Wort	Beschreibung	Herkunft



Teil I

**Projektdetails**



# Kapitel 2

## Aufgabenstellung

### 2.1 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 Log-Datei (offline) möglich. Die unterschiedlichen Charakteristiken der Garbage Collectors bedingen auch unterschiedliche Auswertungs- und Einstellungsparameter. JRocket 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.

### 2.2 Ziele der Arbeit

Ziel der Bachelorthesis ist die Konzeption und Entwicklung eines Prototypen für die Analyse von Garbage Collection Log Dateien der JRocket 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 JRocket Virtual Machine spezifisch erarbeitet und zusammengestellt.

## 2.3 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 aus- gewählten Rich Client Frameworks), der die ermittelten Anforderungen erfüllt.
6. Implementation der Software
7. Bewertung der Software auf Basis der Anforderungen

## 2.4 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

## **2.5 Geplante Termine**

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





# Teil II

## Umsetzung



## Kapitel 3

# Analyse der Aufgabenstellung

### 3.1 Übersicht

Wie in der Aufgabenstellung definiert, ist das Ziel dieser Arbeit die Konzeption und Implementation einer Analysesoftware für die Garbage Collection Log Dateien der JRokit Virtual Machine. Grundlegendes Ziel einer solchen Software ist es, die anfallende grosse Menge an Daten übersichtlich darzustellen und verschiedene Sichten auf diese Daten zu ermöglichen. Um die Anforderungen an diese Software zu ermitteln, ist ein die Studie der Grundlagen der Garbage Collection und im spezifischen die der JRokit Virtual Machine notwendig. 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 genauere Untersuchung der einzelnen Teilaufgaben wird in den folgenden Abschnitten gemacht:

#### 3.1.1 Erarbeitung der Grundlagen Bereich der Garbage Collection

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 im Generellen und spezifisch die der JRockit Virtual Machine.

- **Konzept:** Die Konzeption des Domänen-Modells und des Einlese-Prozesses der Log Dateien bedingt eine genaue Kenntnis der verschiedenen Strategien und Formaten der Ausgaben.

### 3.1.2 Stärken- / Schwächen-Analyse der bestehenden Rich Client Frameworks

Aus unterschiedlichen Gründen ist es sinnvoll, die Applikation als einen Rich Client zu implementieren. Als Basis kommen entweder die Netbeans- oder die Eclipse-Plattform in Frage. Die Stärken- und Schwächen-Analyse soll den Entscheid für die eine dieser Plattform begründen.

### 3.1.3 Durchführung einer Anforderungsanalyse für den Software-Prototypen

Die Anforderungen werden zusammen mit einem Performance-Analysten ermittelt und nach [5, 4.3.2 Angepasste Standardinhalte] dokumentiert. Laut dem IEEE<sup>1</sup> und [5, 4.5 Qualitätskriterien für das Anforderungsdokument] müssen Anforderungen folgende Kriterien erfüllen:

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

### 3.1.4 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.

### 3.1.5 Konzeption und Implementation

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

---

<sup>1</sup>Institute of Electrical and Electronic Engineers

### **3.1.6 Bewertung**

Die Bewertung der Software wird auf Basis der Anforderungen gemacht.



## Kapitel 4

# Grundlagen Garbage Collection

Schon seit den ersten Programmiersprachen ist das Aufräumen von verwendeten Ressourcen / Speicher ein wichtiges Thema. Im Unterschied zu den ersten Sprachen 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 Collection<sup>1</sup>. In den meisten neueren Laufzeitumgebungen spricht man zusätzlich von adaptivem Memory Management was bedeutet, dass Feedback der Laufzeitumgebung zur Anpassung der Garbage Collection Strategie verwendet wird.

Der folgende Abschnitt beschreibt die Grundlagen der Java Garbage Collection und geht im zweiten Teil auf die spezifischen Eigenheiten der JRockit Virtual Machine ein.

### 4.1 Funktionsweise

Alle Techniken der Garbage Collection zielen darauf ab, die "lebenden" von den "toten" Objekten im Speicher zu unterscheiden. Sprich, es müssen die Objekte gefunden werden, welche innerhalb der Software oder des Systems nicht mehr referenziert werden. Die aktuellen Strategien lassen sich laut [1, S. 77] in zwei unterschiedliche Familien aufteilen: "Reference Counting" und "Tracing techniques".

---

<sup>1</sup>vom englischen Wort "garbage collector" für Müll-, Abfallsammler

### 4.1.1 Reference counting[1, S. 77]

Beim Reference counting 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 Systemen.

### 4.1.2 Tracing techniques[1, S. 77]

Bei den Tracing Techniken werden von 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 von Objekten, bestehend aus den Stacks und Registern der aktuellen Threads und globalen Daten wie `static final` variablen, als Startpunkt für die zu markierenden Objektgraphen.

## 4.2 Algorithmen

### 4.2.1 Mark & Sweep Algorithmus[4]

Unabhängig von Nebenläufigkeit oder Parallelität<sup>2</sup> ist der Mark & Sweep Algorithmus heutzutage die Basis aller Garbage Collection Strategien der kommerziellen JVMs. Der Algorithmus funktioniert nach dem folgenden Prinzip:

1. Als erstes wird die Markierungs-Phase (Mark) durchlaufen. Während dieser Zeit wird ausgehend vom Root-Set für jedes referenzierte Objekt rekursiv die transitive Hülle des Objekts bestimmt. Alle die damit besuchten Objekte werden als "erreicht" markiert. Das Root-Set besteht aus den folgenden Objekten:

- Alle vom Call-Stack der Threads referenzierte Objekte

---

<sup>2</sup>Man spricht von "concurrent" Garbage Collection wenn sie parallel zur Applikation ausgeführt wird, von parallel Garbage Collection wenn der Algorithmus parallelisierbar ist und entsprechend gut skalierbar ist.



- Alle globalen Referenzen von Variablen die mit `static final` deklariert sind
2. In der Sweep-Phase wird der Speicher aufgeräumt - im einfachsten Fall heisst das, dass die Bereiche einfach in eine Free-Liste eingetragen werden und die Bereiche vom Allokator somit wieder verwendet werden.

### Nachteil dieses Algorithmus

Der zwar simple Mark & Sweep Algorithmus hat allerdings einen entscheidenden Nachteil: eine alleinige Freigabe der Speicherbereiche führt zu einer starken Fragmentierung. Ein fragmentierter Speicher ist deshalb nicht wünschenswert, weil es für den Allokator immer schwieriger wird, die Speicherrlöcher zu füllen. In einem solchen Fall kann es bis zu einer `OutOfMemoryException` führen, wenn zwar insgesamt genügend Memory frei ist, allerdings nicht genügend am Stück.

#### 4.2.2 Mark & Copy Algorithmus[4]

Da der Mark & Sweep Algorithmus diesen oben beschriebenen Nachteil mit sich bringt, dass der Speicher nach einer gewissen Zeit stark fragmentiert ist, wird teilweise (insbesondere wenn grosse Memory Allokation notwendig ist) der Mark & Copy Algorithmus eingesetzt. In der Theorie wird der gemanagte Speicher in zwei Teile aufgeteilt, einem "from space" und einen "to space". Der Unterschied zum Mark & Sweep Algorithmus manifestiert sich dadurch, dass Objekte immer nur auf einem dieser Bereiche angelegt werden, der andere dieser Bereiche bleibt leer. Anschliessend an die Markierungs-Phase (Mark) wird die Copy-Phase eingeleitet, in der Folge werden alle überlebenden Objekte des "from space" in den "to space" kopiert. Nun werden die Rollen der beiden Bereiche getauscht und der aktive Bereich ist der ehemalige "to space". Dieser Algorithmus bringt den Vorteil mit sich, dass nun der aktive Memory-Bereich wieder aneinanderhängend belegt ist und keine Fragmentierung aufweist.

#### 4.2.3 Mark & Compact Algorithmus[3]

In gewissen Situationen respektive auch in unterschiedlichen Bereichen des Heaps (siehe Generational Garbage Collection[2]) macht es Sinn, unterschiedliche Algorithmen einzusetzen: Mark & Compact ist ein Algorithmus, bei welchem nach der Markierungs-Phase und der Quasi-Elimination der

”totenÖbjekte eine Kompaktierung des Speichers durchführt. Das hat einerseits den Vorteil, dass im Gegensatz zu Mark & Copy nicht der doppelte Speicher benötigt wird, es bringt aber auch einen relativ komplexen Kompaktierungsprozess mit sich. Dieser ist aufgeteilt in die folgenden Schritte:

1. Markierung
2. Berechnung der neuen Lokationen
3. Anpassung der Referenzen
4. Verschiebung der Objekte

### 4.3 Generational Garbage Collection[2]

TODO: Young Generation, Eden, Survivor 1/2, Old Generation, Cards, etc.

## Kapitel 5

# Garbage Collection in der Oracle JRockit Virtual Machine



## Kapitel 6

# Stärken- / Schwächen-Analyse Rich Client Frameworks



## Kapitel 7

# Anforderungsanalyse





## Kapitel 8

## Konzept



## Kapitel 9

# Implementation



## Kapitel 10

## Review



# Literaturverzeichnis

- [1] M. Lagergren and M. Hirt. Oracle Jrookit: The Definitive Guide. Packt Publishing, Limited, 2010.
- [2] A. Langer and K. Kreft. Generational garbage collection. Java Magazin, 3:26–30, 2010.
- [3] A. Langer and K. Kreft. Mark-and-compact. Java Magazin, 7:20–24, 2010.
- [4] A. Langer and K. Kreft. Young generation garbage collection. Java Magazin, 5:24–30, 2010.
- [5] K. Pohl and C. Rupp. Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering– Foundation Level. Dpunkt.Verlag GmbH, 2010.





# Listings



# Abbildungsverzeichnis



# Tabellenverzeichnis

1.1 Glossar . . . . . 9



**Teil III**

**Anhang**

