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

Einleitung

Bei der Performanceanalyse einer Software kann es notwendig sein, das Verhalten der Garbage Collection genauer zu betrachten. Diese Auswertung kann mit der Software JRockit Mission Control im laufenden Betrieb oder danach auf der Basis von resultierenden Logdateien gemacht werden. Im zweiten Fall gibt es allerdings - zumindest für die JRockit 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 JRockit Virtual Machine. Die Software soll so erweiterbar sein, dass sie auch für andere Garbage Collection Algorithmen erweitert werden kann.

Die Bachelorthesis ist in sechs Teile gegliedert: im Teil eins befindet sich die Projektbeschreibung mit der Aufgabenstellung und der Analyse der Aufgabenstellung, Teil zwei enthält die Grundlagen zur Performanceanalyse und Garbage Collection, Teil drei die Anforderungsanalyse, Teil vier das Konzept und Teil fünf die Umsetzung. Der sechste und letzte Teil besteht aus dem 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 Collection 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

Kapitel 2

Analyse der Aufgabenstellung

Das Ziel dieser Arbeit ist die Konzeption und Implementation einer Software für die Analyse von Garbage Collection Logdateien der JRockit Virtual Machine. Die grosse Datenmenge soll schnell eingelesen und übersichtlich, informativ dargestellt werden.

Um die Anforderungen an diese Software zu ermitteln, werden im Bereich der Performanceanalyse, der Garbage Collection und über die JRockit Virtual Machine die nötigen Grundlagen erarbeitet. Anschliessend wird eine Stärken-/Schwächen-Analyse der bestehenden Richt Client Frameworks gemacht, diese dient zusammen mit der Anforderungsanalyse als Grundlage für die Wahl der richtigen Technologie. Danach folgt die Konzeption und Implementation der Software. Die genaueren Beschreibung der einzelnen Teilaufgaben befindet sich in den folgenden Abschnitten.

2.1 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 der JRockit Virtual Machine.

- **Konzept:** Die Konzeption des Domänen-Modells und des Parseprozesses der Logdateien bedingt eine genaue Kenntnis der verschiedenen Strategien und Formaten der Logdateien.

2.2 Stärken-Schwächen-Analyse Rich Client Frameworks

Die Applikation wird als Rich Client implementiert. Als Basis kommen die Frameworks Netbeans und Eclipse in Frage. Auf der Basis der Stärken-Schwächen-Analyse wird der Entscheid für eine dieser Plattformen herbeigeführt.

2.3 Anforderungsanalyse Software-Prototyp

Die Anforderungen werden zusammen mit einem Performance-Analysten ermittelt und nach [11, 4.3.2 Angepasste Standardinhalte] dokumentiert. Laut dem IEEE[1] und [11, 4.5 Qualitätskriterien für das Anforderungsdokument] müssen Anforderungen folgende Kriterien erfüllen:

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

2.4 Evaluation Frameworks

Basierend auf der Stärken-Schwächen-Analyse wird ein Entscheid für das jeweilige Framework (Eclipse RCP oder Netbeans RCP) ausgewählt. Nebst der Auswahl der Rich Client Plattform muss auch eine Charting-Bibliothek ausgewählt werden.

2.5 Konzeption und Implementation

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

2.6 Bewertung

Die Bewertung der Software wird auf Basis der Anforderungen gemacht.

[1] Institute of Electrical and Electronic Engineers

Teil II

Grundlagen

Kapitel 3

Einführung Performanceanalyse

3.1 Motivation

Die Performance respektive Leistungsfähigkeit einer Applikation stellt eine der bedeutendsten Qualitätsanforderungen dar. Probleme in diesem Bereich führen zu verärgerten Benutzern und verlangsamten Business-Prozessen. 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 würden.

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[4]:

1. Identifikation der neuralgischen Punkte des Systems
2. **Suche nach dem Dominating Consumer**[1] **(siehe Abschnitt 3.3)**
3. **Sammeln von Detaildaten (siehe Abschnitt 3.4)**
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 beschrieben.

[1] Kirk 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:

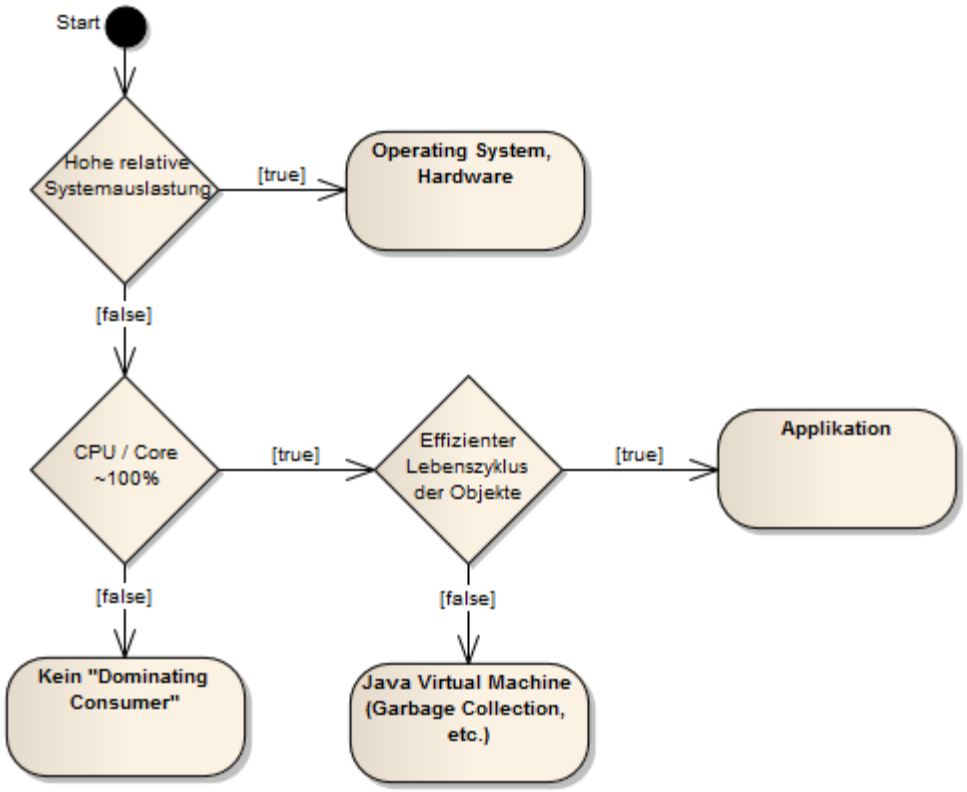

Abbildung 3.1: Suche nach dem Dominating Consumer

3.3.1 Hohe relative Systemlast

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

- Unter **Windows** kann dafür der *Task Manager* verwendet werden. Auf der Lasche Performance müssen dafür allerdings auch die Kernelzeiten eingeschaltet werden.
- Unter **Linux respektive Unix** kann dafür das Programm *vmstat* verwendet werden.

Hohe Systemauslastung kann unterschiedliche Gründe haben:

- **Exzessive Kontextwechsel:** Mehrere Prozesse können sich die gleiche CPU (CPU-Kern) nur deshalb teilen, weil beim Wechsel von Prozess A auf B ein Kontextwechsel gemacht wird, so dass Thread B am selben Ort weiterarbeitet wo er beim letzten Mal war. Gründe für exzessives Kontextwechsel können beispielsweise nicht adäquat gewählte Locks sein. Wenn ein Thread aufgrund der Synchronisation angehalten werden muss.[2]
- **Viel Interaktion (Input/Output) mit der Festplatte**
- **Viel Interaktion mit dem 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 keinen Dominating Consumer - es muss dann herausgefunden werden, was die Threads daran hindert, CPU Ressourcen zu verwenden. Es gibt dafür laut [10] unterschiedliche Gründe:

- **Dead Locks:** Threads schliessen sich gegenseitig aus. Das würde man in einem Thread Dump (Stacktrace) anhand der vielen wartenden Threads erkennen.
- **Applikation skaliert nicht:** Applikation hat schlechte multi-core Eigenschaften. Das ist beispielsweise der Fall, wenn durch Synchronisation oder Memory Barrieren Engpässe entstehen - auch zu erkennen an wartenden Threads im Thread Dump.
- **Langsame Disks / Netzwerke:** In einem Sampling oder Profiling wäre beispielsweise erkennbar, dass Threads oft in *read*-Operationen stecken.
- **Zu kleine Connection- und Thread-Pools**
- **Aurufe auf langsame externe Systeme**

3.3.3 Effizienter Objekt-Lebenszyklus

Sofern die CPU-Auslastung, trotz kleiner relativer Systemlast durch die Applikation, hoch ist, muss unter anderem[3] die Garbage Collection angeschaut werden. Dafür gibt es unterschiedliche Herangehensweisen:

- **Memory Analyse (Objektpopulation):** Es wird angeschaut, wie alt die Objekte in den unterschiedlichen Bereichen (Young Generation, Old Generation) sind. Wieviele Garbage Collection Zyklen sie überlebt haben.

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

[3] auch der Optimizer könnte dafür verantwortlich sein

- **Garbage Collection:** Wann und wie oft werden Garbage Collections durchgeführt, wie lange haben sie gedauert, wieviel Speicher haben sie befreit.

Im weiteren konzentrieren wir uns auf die Ziele und Vorgehensweise beim Garbage Collection Tuning.

3.4 Garbage Collection Tuning

Mit dem Resultat der Analyse des Dominating Consumers entscheidet sich auch, in welchem Bereich weitere Detaildaten gesammelt werden müssen. Wir beschränken uns in diesem Abschnitt auf die Analyse im Bereich der Garbage Collection. Hier gilt grundsätzlich auch: "Never change a running system". Man ändert also nur etwas, wenn die Leistung den Anforderungen nicht genügt.

In der Regel will man mit dem Tuning eines der drei unten stehenden Zielen erreichen[7]:

- 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-World[4] Pausen. Tuning bedeutet nun, zwei der Ziele konstant zu halten, während das dritte durch Anpassung der Konfiguration verbessert wird. Sofern die Optimierung nicht ausreicht, muss einer der beiden anderen Parameter aufgegeben werden, um das Ziel zu erreichen.

3.4.1 Durchsatz

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

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

Die Formel ist nur eine Annäherung und berücksichtigt weder nebenläufige Garbage Collection noch Maschinen mit mehreren Prozessorkernen, Durchsatz-Tuning spielt aber in der Praxis auch oft nur eine sehr untergeordnete Rolle[7].

[4] Zeiten in welchen die Prozessoren der Anwendung keine Rechenzeiten zur Verfügung stellen.

Eine wirkliche Steigerung des Durchsatzes kann man nämlich nur mit einer aggressiven Optimierung erzielen, und die führt zu verlängerten Pausenzeiten, was nur bei Batch-Applikationen toleriert werden kann. Der Austausch der CPU oder das Zuschalten von weiteren ist normalerweise der kostengünstigere Weg.

3.4.2 Pausenzeiten

In der Regel geht es beim Garbage Collection Tuning um das Tuning der Pausenzeiten, dies weil man die Symptome wie das Stottern einer Anwendung nicht tolerieren kann.

Man startet dann mit der Analyse einzelner Garbage Collections und versucht die Gründe für zu häufige oder zu lange dauernde Collections zu finden. Zu häufige Initiierung einer Garbage Collection kann unterschiedliche Gründe haben:

- **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, Beseitung der Fragmentierung, etc.
- **Von der Applikation initiiert (durch System.gc()):** Von der Applikation ausgelöste Garbage Collection Kommandos können mittels Aktivierung eines Flags ignoriert werden.

3.4.3 Speicherverbrauch

Dieses Tuningziel verliert insofern an Relevanz, weil 64-Bit-JVMs bei grossen Applikationen schon sehr verbreitet sind und damit die Adressierung von mehr Speicher möglich ist. Mit 32-Bit Adressen kam man früher schon eher an die Grenze (ein Teil des adressierbaren Bereichs bei 32-Bit-Adressen von 4GB wird 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

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 Collection[1]. 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 References[2] (Dangling Pointers) und Space Leaks[3]. 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 Eigenheiten dazu bei der 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 [5, S. 77] gibt es zwei unterschiedliche Strategien: Referenz-Zählung (Reference Counting) und Tracing-Techniken (Tracing Techniques).

[1] vom englischen Wort "garbage collector" für Müll-, Abfallsammler

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

[3] Man spricht von Space Leaks, wenn Memory alloziert und nicht mehr freigegeben wurde, obwohl es nicht mehr gebraucht wird.[8]

4.1.1 Reference counting

Beim Reference counting[5, 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 zum Aufräumen freigegeben. Obwohl 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 null sein - auch wenn sie nicht mehr verwendet werden.
- 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[5, S. 77] werden vor jeder Garbage Collection die Objekte gesucht, auf welche aktuell noch eine Referenz zeigt. Die anderen werden zum Aufräumen freigegeben. Diese Art von Garbage Collection Algorithmen verwenden eine Menge von Objekten[4] als Startpunkt für die Traversierung aller Objekte.

4.2 Ziele der Garbage Collection

Für Garbage Collectors gibt es folgende Bedingungen[8, S. 4]:

- **Sicherheit:** Garbage Collectors dürfen nur Speicher von Objekten freigeben, der effektiv nicht mehr gebraucht wird.
- **Umfassend:** Garbage Collectors müssen den Speicher von nicht mehr gebrauchten Objekten nach wenigen Garbage Collection Zyklen freigegeben haben.

Wünschenswert sind zudem folgende Punkte[8, S. 4]:

- **Effizienz:** Die Anwendung soll vom laufenden Garbage Collector möglichst nicht beeinträchtigt sein:
  - keine langen Pausen[5]
  - einen durch den Garbage Collector möglichst geringen Ressourcenverbrauch

[4] Das Root Set besteht aus den von globalen Variablen und Stacks aller Threads referenzierten Objekten.

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

- **Geringe bis keine 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

Für die Selektion eines Algorithmus gibt es unterschiedliche Kriterien[8, S. 5], die in der Folge genauer definiert sind:

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 ab gut zwei Prozessoren 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 (Stottern der Applikation), 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 aller Objekte in einen anderen Bereich minimiert werden.

4.4 Grundlage der Algorithmen

Der Prozess der Garbage Collection beginnt bei allen Algorithmen mit der Markings-Phase. Für jedes Objekt des Root Sets[6] werden rekursiv die transitiv abhängigen Objekte auf dem Heap bestimmt. Alle nicht im Resultat enthaltenen Objekte sind tot und können bereinigt werden.

[6] Objekte 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 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. Der grosse Nachteil dieses Algorithmus ist, dass der Speicher nach vielen Garbage Collections stark fragmentiert ist. Aus diesem Grund gibt es unterschiedliche Erweiterungen des Algorithmus.

4.4.2 Mark & Copy Algorithmus

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

Für die Old Generation 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[5, 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 Objekte[7] 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.

4.5.1 HotSpot Virtual Machine

Die HotSpot Virtual Machine tut dies folgendermassen[6]:

- **Young Generation:** Die Young Generation 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 Generation:** In der Old Generation befinden sich die Objekte, die eine gewisse Anzahl an Young 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-Informationen[8]. gespeichert.

Intergenerationelle Referenzen

Die Idee der generationellen Garbage Collection ist, dass während einer Young Collection nur ein Teilbereich des Heaps aufgeräumt wird. Referenzen in die Old Generation werden dabei nicht verfolgt. Den Referenzen von der Old in die Young Generation muss trotzdem nachgegangen werden, da sonst der Speicher von noch gebrauchten Objekten befreit wird. Intergenerationelle Referenzen können auf folgende zwei Arten 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.

[7] Objekte mit kurzer, mittel und langer Lebensdauer.

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

- Es findet eine **Zuweisung der Referenz durch die Applikation** statt. Diese Problematik wird durch Write Barriers und der Aufteilung des Heaps in sogenannte Cards[9] gemacht. Bei der Änderung einer Referenz (die dann von der Old in die Young Generation zeigt) wird auf der entsprechenden Card das "Dirty Bit" gesetzt. Objekte innerhalb von dirty Cards werden nach der Markierungsphase nochmals überprüft.

4.5.2 JRockit Virtual Machine

Bei der JRockit Virtual Machine ist es so:

- **Nursery[10]:** Die Nursery entspricht der Young Generation 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 erneuten Garbage Collection in die Old Generation.
- **Old Generation:** Die Old Generation beinhaltet wie bei der HotSpot Virtual Machine die Objekte mit langer Lebensdauer.

Für weitere Details zu den Eigenheiten der JRockit Garbage Collection siehe Kapitel 5.

[9] Eine Card ist typischerweise ein ungefähr 512 Byte grosser Bereich auf dem Heap.
[10] Nursery bedeutet im übertragenen Sinn Kindergarten

Kapitel 5

Garbage Collection JRockit

5.1 Algorithmen

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 Optimierungen

Verwendung von threadlokalem Speicher

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 threadlokalen Queues jedes Garbage Collection Threads gespeichert werden.

Die Verwendung von threadlokalem Speicher hat hinsichtlich den folgenden Punkten einen Vorteil:[5, S. 79]:

- Threadlokaler Speicher führt zu einer besseren Parallelisierbarkeit, da Zugriffe darauf nicht synchronisiert werden müssen.

- Threadlokaler Speicher kann voraus geladen (Prefetching) 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.

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 umher kopiert, so dass danach wieder grössere freie Speicherbereiche existieren.

5.1.2 Übersicht und Auswahl Algorithmen

Algorithmen können direkt selektiert werden. Manchmal passiert es aber trotzdem, dass der Garbage Collection Algorithus im laufenden Betrieb geändert wird.

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 5.1: Auswahl des Garbage Collection Algorithmus

5.2 Nebenläufige Algorithmen

Bei den Concurrent Mark & Sweep Algorithmen handelt es sich eigentlich um "mostly concurrent" Algorithmen. Das heisst, sie finden nicht in allen Phasen konkurrierend zur Applikation statt. Damit ein Teil der **Markierungsphase** konkurrierend statt finden kann, ist sie 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 die Objekte als lebendig.
- 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.

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.

5.3 Parallele Algorithmen

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

5.4 Garbage Collection Modi

Anstelle einer direkten Auswahl eines Algorithmus (siehe Abschnitt 5.1.2), kann auch eine grundlegende Strategie angegeben werden. Die Virtual Machine entscheided dann heuristisch, welcher spezifische Algorithmus gewählt wird - in der Regel entscheided er insbesondere, ob er den Heap in Generationen aufteilen soll. Die verfügbaren Modi entsprechen den Tuningzielen die es für die Garbage Collection gibt (siehe Abschnitt 3.4):

- **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
troughput	-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 5.2: Übersicht der Garbage Collection Modi

5.5 Garbage Collection Logdateien

Die Auswertung der Garbage Collection kann auf Basis von Logdateien gemacht werden. Das Format dieser Logdateien hängt mitunter auch von den aktivierten Log-Modulen ab. Der Aufbau der Logdateien und die in der Analyse verwendeten Daten werden in diesem Abschnitt genauer beleuchtet.

5.5.1 Aktivierung Log Ausgaben

Standardmässig macht die JRockit keine Angaben darüber, wie sie die Garbage Collection durchführt. Es besteht aber durch die Aktivierung des entsprechenden Log-Modules die Möglichkeit, an diese Informationen zu gelangen. Die Ausgaben werden dann auf die Standard Ausgabe oder optional (Argument -Xverboselog:logdatei.log) in eine Datei geschrieben. Das Format für die Kommandozeile ist wie folgt:

```
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 muss man folgendes Argument übergeben:

```
1 -Xverbose:memory
```

Listing 5.2: Garbage Collection Log (Info)

Die Umleitung der Ausgaben in eine separate Logdatei kann folgendermassen angegeben 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 angepasst werden:

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

Listing 5.4: Einstellung des Log-Levels

Als Log-Levels stehen folgende Werte zur Verfügung: quiet, error, warn, info, debug, trace. Für weiter Informationen siehe [9].

5.5.2 Beispiel einer Garbage Collection Logdatei

Die ersten Zeilen einer Log-Datei sind auf der folgenden Seite ersichtlich:

```
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 Logdatei

5.5.3 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 der 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

Teil III

Anforderungsanalyse

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.

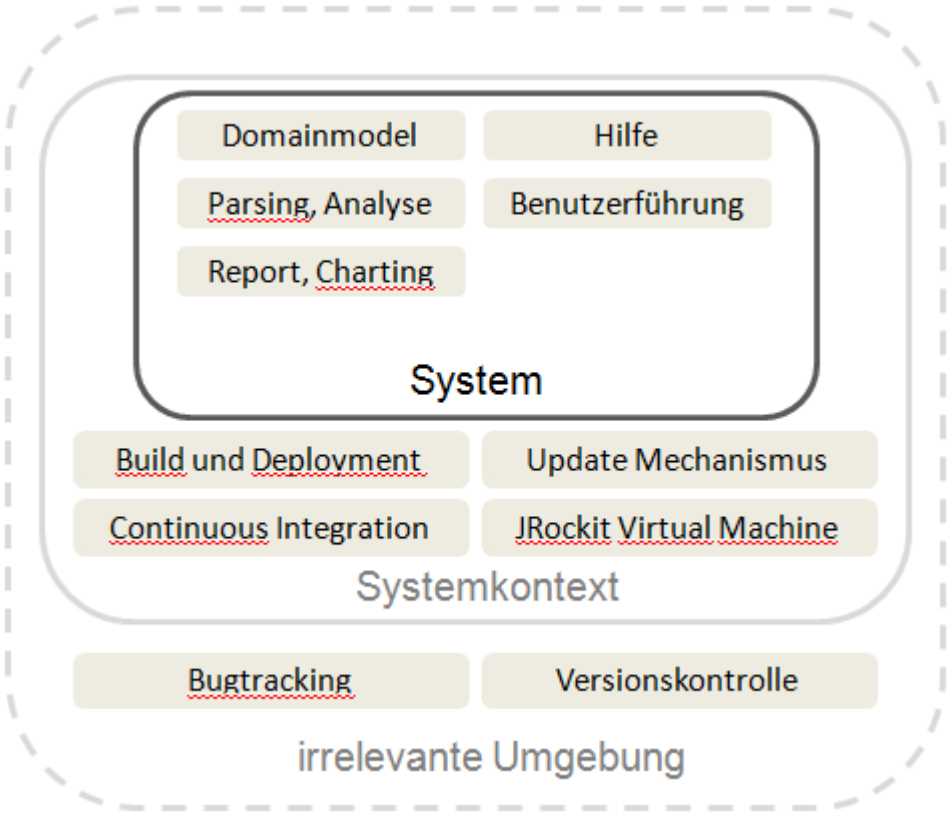

Abbildung 6.1: System und Systemkontext

System

Das System besteht aus den Teilen Datenmodell, Domäne, Log Parsing, Garbage Collection Analyse, Report, Charting, Hilfesystem[1], 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 Packete werden gebildet, es wird ein für den jeweiligen Update-Mechanismus notwendiges Packet erstellt und auf den Update-Server gestellt.

[1] Dem 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 genaurere Beschreibung befindet sich unter Garbage Collection Logdateien.

Irrelevante Umgebung

- **Issuetracker:** Sobald die Software stabil läuft und an Tester herausgegeben wird, wird für die Verwaltung der Fehler (Bugs) und Features ein Issuetracker 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 (Issuetracker, Versionskontrolle) arbeiten normalerweise eng zusammen, so dass Bug Fixes mit der eingecheckten 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.

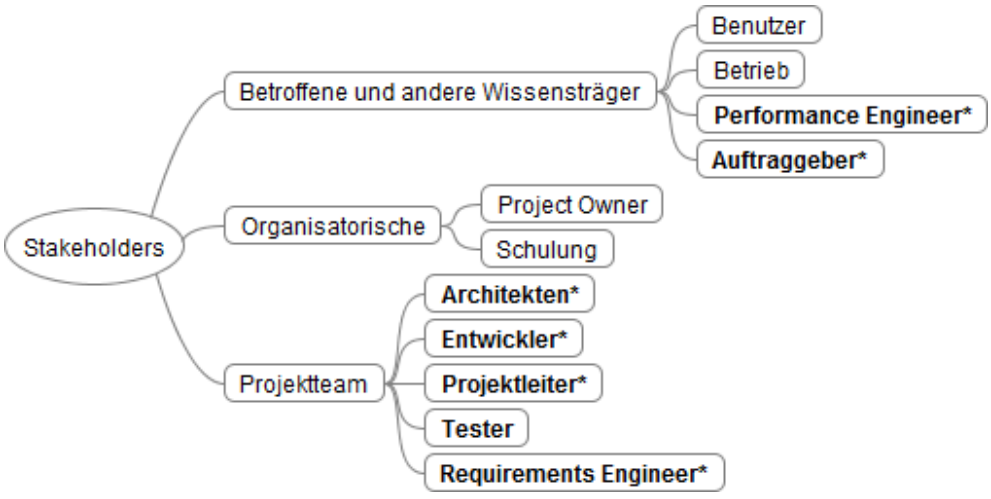

Abbildung 6.2: Übersicht der Stakeholder

6.1.4 Glossar

Das Glossar befindet sich im Abschnitt 13.2.3.

6.1.5 Referenzen

Die Referenzen innerhalb des Abschnitts 6 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 Anforderungen an die Software definiert, aus welchen sich anschliessend die unterschiedlichen Use Cases ableiten.

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 vorallem 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 First auf den Grund zu gehen.

6.2.3 Randbedingungen

Das Format der Logdatei der JRockit 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 Use Cases

Dieser Abschnitt zeigt die Use Cases für die Analysesoftware. Daraus werden dann im nächsten Abschnitt die Anforderungen respektive die finale Anforderungsliste abgeleitet. Als Einstieg dient das UML Use Case Diagramm, in welchem die Systemgrenzen, der Anwender (Performance Analyst) und die verschiedenen Use Cases dargestellt sind. Die einzelnen Use Cases sind dann nach der in [11, S. 78-79] beschriebenen Schablone definiert.

6.3.1 Übersicht

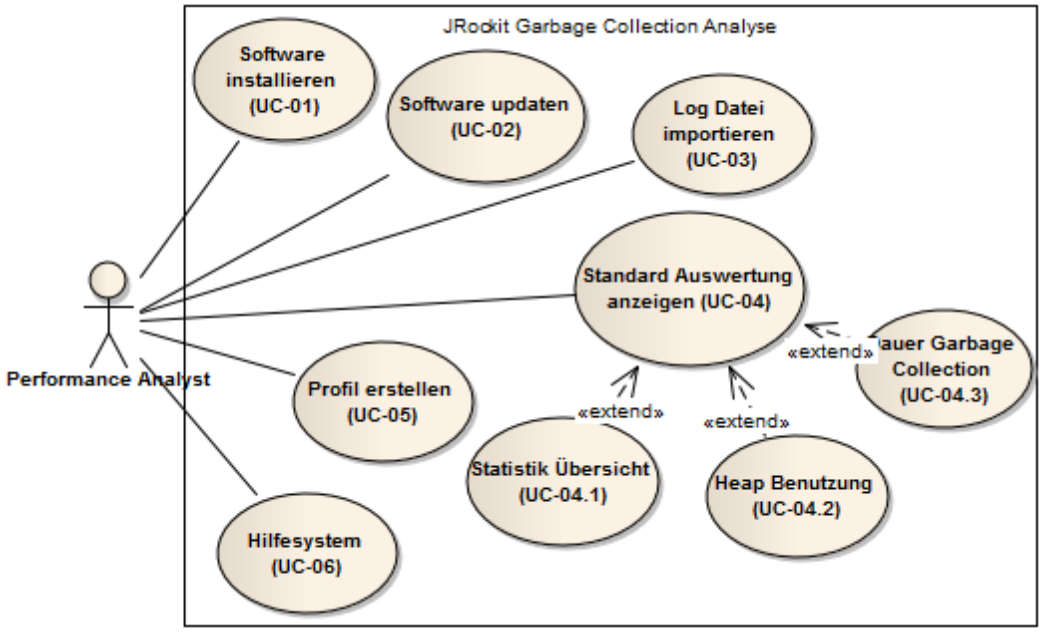

Abbildung 6.3: Systemfunktionalität als Use-Case-Diagramm

6.3.2 Beschreibung

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

Abschnitt	**Inhalt / Erläuterung**
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	1. Anwender Startet Entwicklungsumgebung 2. Anwender gibt Update-Seite an 3. Anwender selektiert zu installierendes Softwarepaket 4. Softwarepaket wird installiert
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Usability

Tabelle 6.1: Use-Case: Software installieren

Abschnitt	**Inhalt / Erläuterung**
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	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	Usability

Tabelle 6.2: Use-Case: Software updaten

Abschnitt	**Inhalt / Erläuterung**
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	1. Anwender öffnet Import-Wizard 2. Anwender navigiert zum Ordner 3. Anwender selektiert Datei(en) und importiert diese Die importierten Dateien werden gespeichert. Bei einem Neustart der Entwicklungsumgebung bleiben die zuvor importierten Dateien erhalten.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Usability

Tabelle 6.3: Use-Case: Garbage Collection Logdatei importieren

Abschnitt	**Inhalt / Erläuterung**
Bezeichner	UC-04
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	Der Benutzer hat eine Garbage Collection Logdatei importiert und möchte nun die Analyse starten.
Vorbedingung	Bevor das Analysefenster für die Garbage Collection Logdatei geöffnet werden kann, wird die Datei eingelesen und geparst. Das heisst, dass die rohen Daten semantisch und syntaktisch analysiert und in den Arbeitsspeicher geladen werden.
Nachbedingung	-
Ergebnis	Dem Benutzer wird ein Analyse-Screen angezeigt.
Hauptszenario	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, Performance, Usability
Erweiterungen	UC-04.1, UC-04.2, UC-04.3, UC-04.4

Tabelle 6.4: Use-Case: Standardausertung anzeigen

Abschnitt	**Inhalt / Erläuterung**
Bezeichner	UC-04.1
Name	Anzeige Statistik Übersicht
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: gross
Kritikalität	gross
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Der Analyse-Screen wurde geöffnet, dem Benutzer zeigen sich unterschiedliche Tabs. Auf dem ersten befinden sich verschiedene statistische Auswertungen der Logdatei: • 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, Performance, Usability

Tabelle 6.5: Use-Case: Anzeige Statistik Übersicht

Abschnitt	**Inhalt / Erläuterung**
Bezeichner	UC-04.2
Name	Anzeige Heap Benutzung
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: gross Technologisches Risiko: gross

Kritikalität	gross
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	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.
Qualitäten	Korrektheit, Performance, Usability

Tabelle 6.6: Use-Case: Anzeige Heap Benutzung

Abschnitt	**Inhalt / Erläuterung**
Bezeichner	UC-04.3
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, Performance, Usability

Tabelle 6.7: Use-Case: Anzeige Dauer Garbage Collection

Abschnitt	**Inhalt / Erläuterung**
Bezeichner	UC-05
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. Auf jedem Chart können unterschiedliche Serien[2] hinzugefügt werden. Die Profile sind persistent und können exportiert wie auch importiert werden.
Akteure	Anwender, Logdatei Analyzer, Report Engine
Auslösendes Ereignis	Die Applikation hat die Datei fertig eingelesen und geparst.

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

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. Sollte die Entwicklungsumgebung mit der Analysesoftware geschlossen werden, bleiben die erstellten Profile erhalten.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Korrektheit

Tabelle 6.8: Use-Case: Profil (benutzerdefinierte Auswertung) erstellen

Abschnitt	**Inhalt / Erläuterung**
Bezeichner	UC-6
Name	Hilfesystem
Autoren	Raffael Schmid
Priorität	Wichtigkeit für Systemerfolg: niedrig Technologisches Risiko: mittel
Kritikalität	Mittel
Verantwortlicher	Raffael Schmid
Kurzbeschreibung	Dem Benutzer steht eine eine indexbasierte[3] und eine kontextsensitive[4] Hilfe zur Verfügung.
Akteure	Anwender
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

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

[4] Hilfe zur aktuellen View oder Aktion des Benutzers

Hauptszenario	• **Indexbasierte Hilfe:** Der Benutzer kennt sich im Thema Garbage Collection und auf der Analysesoftware 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 notwendingen Informationen.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Internationalisierung, Usability

Tabelle 6.9: Use-Case: Hilfesystem

6.4 Funktionale Anforderungen

Die folgende abschliessende Liste der Funktionalen Anforderungen an die Analysesoftware (stand 25. November 2011) wurde hinsichtlich Korrektheit, Machbarkeit, Notwendigkeit geprüft und entsprechend Priorisiert:

Identifik.	**Vers.**	**Titel**	**Beschreibung**	**U.C.**[5]	**Quelle**	**Abnahmekriterium**	**Prio.**
FRQ-01	1.0	Installation	Software muss als Erweiterung in der Entwicklungsumgebung[6] installiert werden können.	UC-01	Raffael Schmid (Project Owner)	Entwickler mit durchschnittlichen Kenntnissen benötigen für die Installation in eine bestehende Entwicklungsumgebung weniger als 5 Minuten.	gross
FRQ-02	1.0	Updaten	Die Software kann mit geringem Aufwand aktualisiert werden.	UC-02	Raffael Schmid (Project Owner)	Entwickler mit durchschnittlichen Kenntnissen benötigen für den Update weniger als 3 Minuten.	mittel

[5] Use Case

[6] Die Wahl der Entwicklungsumgebung respektive des Frameworks befindet sich im Abschnitt Auswahl Frameworks und Komponenten.

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.	UC-03	Raffael Schmid (Project Owner)	-	gross
FRQ-04	1.0	Zustand Ansicht Logdateien speichern	Die sich in der Ansicht befindenden Logdateien Einträge werden gespeichert.	UC-03.1	Raffael Schmid (Project Owner)	-	gross
FRQ-05	1.0	Garbage Collection Logdatei einlesen	Geöffnete Garbage Collection Logdateien werden ins Memory gelesen.	UC-04	Raffael Schmid (Project Owner)	Der Einleseprozess bei einer Datei mit 100000 Zeilen dauert weniger als 2 Sekunden.	gross
FRQ-06	1.0	Garbage Collection Logdatei parsen	Die eingelesenen JRockit Garbage Collection Logdatei wird geparst. Aus den Daten wird ein Domänenmodell aufgebaut.	UC-04	Raffael Schmid (Project Owner)	Das Parsen einer Logdatei mit 100000 Zeilen dauert nicht länger als 8 Sekunden.	gross
FRQ-07	1.0	Standardauswertung anzeigen	Der Benutzer kann eine vordefinierte Anzeige öffnen.	UC-04	Raffael Schmid (Project Owner)	-	gross

FRQ-08	1.0	Anzeige Statistik Übersicht	Der erste Screen (Tab) auf dem Analysefenster zeigt verschiedene aggregierte Messwerte.	UC-04.1	Raffael Schmid (Project Owner)	-	gross
FRQ-09	1.0	Anzeige Heap Benutzung	Grafische Darstellung des gebrauchten Speichers im Heap über die Zeit.	UC-04.2	Raffael Schmid (Project Owner)	-	gross
FRQ-10	1.0	Anzeige Dauer Garbage Collection	Grafische Darstellung der dauer der einzelnen Garbage Collection Zyklen über die Zeit.	UC-04.3	Raffael Schmid (Project Owner)	-	gross
FRQ-11	1.0	Profil (Benutzerdefinierte Auswertung) erstellen	Benutzer kann Profil erstellen und darauf entsprechende Charts konfigurieren. Das Profil kann gespeichert sowie exportiert und importiert werden.	UC-05	Adrian Hummel (Performance Analyst)	-	klein
FRQ-12	1.0	Charts definieren	Der Benutzer kann benutzerdefinierte Charts definieren. Auf das Chart können beliebige Datenserien konfiguriert werden.	UC-05	Adrian Hummel (Performance Analyst)	-	klein

FRQ-13	1.0	Profil speichern	Die Profile werden gespeichert und überleben einen Neustart der Entwicklungsumgebung.	UC-05	Adrian Hummel (Performance Analyst)	-	klein
FRQ-14	1.0	Charts exportieren	Die definierten Profile können in eine Datei exportiert werden.	UC-05	Adrian Hummel (Performance Analyst)	-	klein
FRQ-15	1.0	Charts importieren	Die in eine Datei exportierten Profile können importiert werden.	UC-05	Adrian Hummel (Performance Analyst)	-	klein
FRQ-16	1.0	Hilfesystem	Dem Benutzer werden eine indexbasierte und eine kontextsensitive Hilfe zur Verfügung gestellt.	UC-06	Raffael Schmid (Project Owner)	Die Hilfe ist in Deutsch und Englisch verfügbar.	klein

Tabelle 6.10: Funktionale Anforderungen

6.5 Qualitätsanforderungen

6.5.1 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[7] nicht überschreiten.	mittel
QRQ-S-02	1.0	Testabdeckung	Zur Qualitätskontrolle muss es eine angemessene Testabdeckung geben.	Raffael Schmid (Project Owner)	Angestrebte Test-Coverage: 80%	klein
QRQ-S-03	1.0	Internationalisierung	Die Sprachelemente der Software (Labels, Titel, Texte) werden als Ressourcen definiert, was die spätere Erweiterung um neue Sprachen ermöglicht.	Raffael Schmid (Project Owner)	-	klein
QRQ-S-04	1.0	Usability	Schnelles Einlesen von grossen Dateien, Benutzerfeedback über eine 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

[7] Personentage

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 Softwarepacket		Raffael Schmid (Project Owner)	Die grösse der gesamten Software soll 10 Megabyte nicht überschreiten.	mittel

6.5.2 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 ist wichtig.	Raffael Schmid (Project Owner)	-	gross
QRQ-F-02	1.0	Plattformunabhängig	Software soll auf den gängigsten Betriebssystemen Windows, Linux und Apple OSX laufen.	Raffael Schmid (Project Owner)	Framework läuft auf den Plattformen Windows, Linux 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
QRQ-F-05	1.0	Offline Betriebsmodus	Der Anwender soll die Software auch im Offline-Modus[8] benutzen können.	Raffael Schmid (Project Owner)	Eigenständige Software, keine Web Applikation	gross
QRQ-F-06	1.0	Installation als Erweiterung	Software wird als Erweiterung in einer Entwicklungsumgebung installiert.	Raffael Schmid (Project Owner)	-	gross

Tabelle 6.12: Qualitätsanforderungen Basisframework

[8] Auf einem Computer der sich nicht am Netz befindet.

Teil IV

Konzept

Kapitel 7

Auswahl Frameworks und Komponenten

7.1 Vorgehen Evaluation

Für die Bewertung der verschiedenen zu evaluierenden Komponenten wird ein Berechnungsmodell benötigt. Aus den relevanten Anforderungen ergeben sich die Kriterien an die Software. Für jedes Kriterium wird eine entsprechende Gewichtung definiert, Quelle dafür ist die Priorität der Anforderung aus der Anforderungsanalyse. Die Umwandlung findet nach folgendem Muster statt:

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

Tabelle 7.1: Schema Umwandlung Priorität in Gewicht

Die Bewertung der Komponenten hinsichtlich den Kriterien basiert auf dem aus der Evaluation erhaltenen Erfahrungswert und wird folgendermassen in eine Zahl umgewandelt:

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

Tabelle 7.2: Schema Vergabe der Punkte

Die Punktezahl berechnet sich aus dem Produkt aus Gewichtung und Bewertung:

Punktezahl = Gewichtung x Bewertung

7.2 Evaluation Rich Client Framework

Grundsätzlich kommen für die Entwicklung der Analysesoftware auch die Java GUI-Bibliothek Swing in Frage. Viele der Anforderung (Deployment, Installation, Update, Modularisierung, Internationalisierung, etc.) werden von Rich Client Frameworks bereitgestellt und stehen mit teilweise geringem Aufwand zur Verfügung. Bei der Evaluation der Bibliothek werden darum nur Rich Client Frameworks berücksichtigt. Von denen kommen aufgrund der funktionalen Anforderung QRQ-F-05 (Offline Betriebsmodus) und QRQ-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 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. Seit dem 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[14].

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

7.2.2 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.

7.2.3 Auswertung

Die erste und zweite Spalte zeigen die Anforderungen aus Abschnitt 6.5.2. Die dritte Spalte enthält das aus der Priorität abgeleitete Gewicht (siehe Abschnitt Gewichtung). Die weiteren Spalten zeigen pro Kandidat die Bewertung zusammen mit dem errechneten Produkt aus Gewichtung und Bewertung.

Anforderung	Nummer	Gewicht.[1]	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

7.2.4 Entscheid

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

7.3 Evaluation Bibliothek Charting

Für die grafische Darstellung der Daten (Charts) muss eine Bibliothek verwendet werden. Im Bereich der nicht-kommerziellen Produkte wird dafür sehr oft Eclipse BIRT und JFreeChart eingesetzt.

7.3.1 Business Intelligence and Reporting Tools (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.

[1] Gemäss Priorität der Anforderung

7.3.2 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.

7.3.3 Auswertung

Anforderung	**Nummer**	**Gewicht.**[2]	**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

7.3.4 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, kann in der Analysesoftware nicht angewendet werden. Deshalb fiel der Entscheid für die Bibliothek JFreeChart aus.

[2] Gemäss Priorität der Anforderung

Kapitel 8

Architektur

8.1 Allgemein

8.1.1 Ausgangslage

Als Rich Client Framework wird Eclipse 3.x[1] 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 License[2] 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 Entwickler beteiligen. Der Entwicklungsaufwand kann skaliert werden.
- Jedermann kann Erweiterungen entwickeln oder Fehler beheben.

8.2 Architektur

Aufgrund der Anforderung QRQ-S-01 (Erweiterbarkeit) muss die Analysesoftware 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 Logs der HotSpot Virtual Machine auswertet. Dies hat hinsichtlich Architektur einige Konsequenzen:

[1] die aktuelle Version ist 3.7 (stand: 31.8.2011)
[2] http://www.eclipse.org/legal/epl-v10.html

- Die Applikation wird in zwei Komponenten aufgeteilt:
  - 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 Machinen analysiert werden können.
- Die Basissoftware stellt Extension-Points[3] bereit, über welche sich die Erweiterungen registrieren.

8.2.1 Struktur

Die generischen Aspekte der Software befinden sich in der Basissoftware. Nur die Spezialisierung im Bereich jedes einzelnen Log-Formats findet in den Erweiterungen statt. Grundsätzlich werden die Aufgaben folgendermassen verteilt:

- Basissoftware (Core Feature)
  - Garbage Collection Log importieren
  - Garbage Collection Log einlesen
  - Profil erstellen, speichern, exportieren, importieren
  - Hilfesystem (wobei allerdings jedes einzelne Feature spezifische Hilfethemen beisteuern kann)
- JRockit Extension (JRockit Extension Feature)
  - Garbage Collection Log parsen
  - Jede Erweiterung hat ein eigenes Domänemodell, da der Aufbau des Garbage Collectors Herstellerabhängig ist.

Das Core-Feature ist die Basis und verantwortlich für den gesamten Import-Prozess (Import-Wizard, Leseprozess der Log-Datei, Anzeige der Menus, Profil-Verwaltung, etc.). Die JRockit Extension ist eine für die Garbage Collection Logs der JRockit geschriebene Erweiterung. Sie ist für das Parsing der Logdateien, die Aufbereitung und Bereitstellung der Daten zuständig. Beinhaltet aber keine Basisfunktionalität.

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

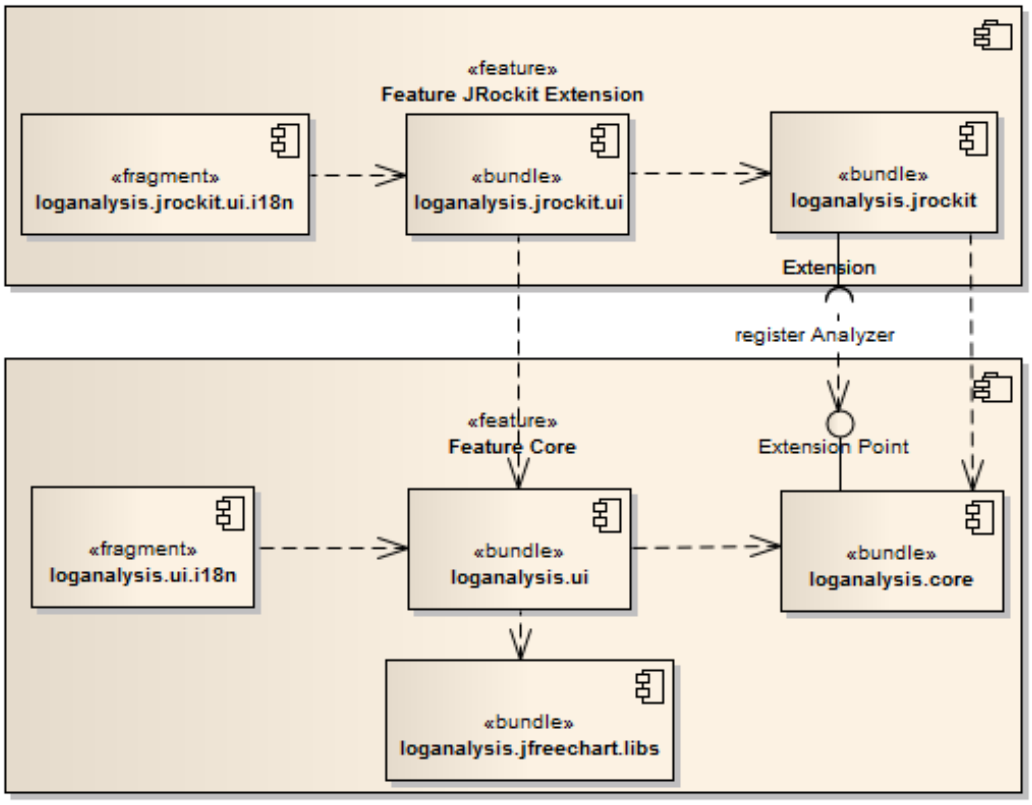

Abbildung 8.1: Architektur: Komponentendiagramm

Das Core Feature besteht aus dem Modul User Interface ("loganalysis.core.ui") und einem von JFace und SWT[4] unabhängigen Teil ("loganalysis.core"). Öffnet der Benutzer eine Garbage Collection Logdatei, wird diese durch das Core Feature eingelesen und der Inhalt wird an alle verfügbaren Extensions weitergeleitet. Die erste Extension welche den Inhalt der Datei verarbeiten kann, öffnet seine dafür vorgesehenen Reports und Charts. Jede Extension hat ein basierend auf der Logdatei eigenes Domänen-Modell.

Nicht-funktionale Module

Die folgenden Projekte (Plugins) sind im Zusammenhang mit dem Deployment und Testing notwendig, beinhalten aber keine eigentliche Funktionalität. In der Regel beinhalten sie Konfigurationsdateien oder Test-Klassen:

- Test-Projekte[5]

[4] JFace und SWT wird in Eclipse als Library für die Präsentationsschicht verwendet.
[5] Der Test-Code befindet sich in eigenen Projekten (siehe Abschnit 10.2).

  - core.test
  - core.ui.test
  - jrockit.test
  - jrockit.ui.test
- Features[6]
  - loganalysis.feature
  - loganalysis.jrockit.feature
- Thirdparty Bibliotheken[7]
  - loganalysis.jfreechart.libs (JFreeChart Library)
- Targetplattform[8]
  - loganalysis.targetplatform (beinhaltet die Target-Plattform)
- Update-Seite[9]
  - loganalysis.updatesite (definiert und generiert die Update-Seite)

[6] Features sind in sich lauffähige Softwarekomponenten mit definiertem Umfang. Zur Definition wird ein eigenes Eclipse-Projekt erstellt.
[7] Thirdparty Bibliotheken werden bei Eclipse-Anwendungen als Plugins gepackt.
[8] Definiert gegen welche Plattform die Anwendung entwickelt wird.
[9] Definiert bereitgestellte Features

Kapitel 9

Konzept funktionale Anforderungen

9.1 Installation (FRQ-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 Analysesoftware 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). Die Eclipse Runtime kann solche Softwarepakete zur Laufzeit installieren und updaten.

Die Update-Seite widerspiegelt die Struktur der Software:

- **Basissoftware:** Umfasst alle Plugins , die für die Basissoftware notwendig sind (Siehe Abschnitt 8.2.1).
- **JRockit Erweiterung:** Umfasst alle Plugins zur JRockit Erweiterung und hat zugleich die Abhängigkeit auf das Basissoftware-Feature.

9.2 Update (FRQ-02)

Der Update eines Features auf der Update-Seite durch die Erstellung eines neuen Releases ist erkennbar durch eine Änderung der Versionsnummer. 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, ein Neustart ist allerdings danach Pflicht.

9.3 Datei importieren (FRQ-03)

Der Import[1] einer Logdatei findet über einen Import-Wizard statt. Der Ablauf 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, der Inhalt wurde allerdings noch nicht analysiert.

9.4 Importierte Dateien speichern (FRQ-04)

Der im Abschnitt A.1 beschriebene Mechanismus wird verwendet, damit nach einem Neustart der Entwicklungsumgebung die importierten Logdateien nicht verloren gehen.

9.5 Datei einlesen (FRQ-05)

Durch den Import einer Garbage Collection Logdatei erscheinen diese im Fenster *Logdateien*. 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.

9.5.1 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.

[1] Der Begriff Import ist irreführend, da diese Aktion einzig dazu dient, die Datei in die Ansicht Logdateien zu kriegen. Erst mit dem Öffnen der Datei werden die Daten wirklich gelesen.

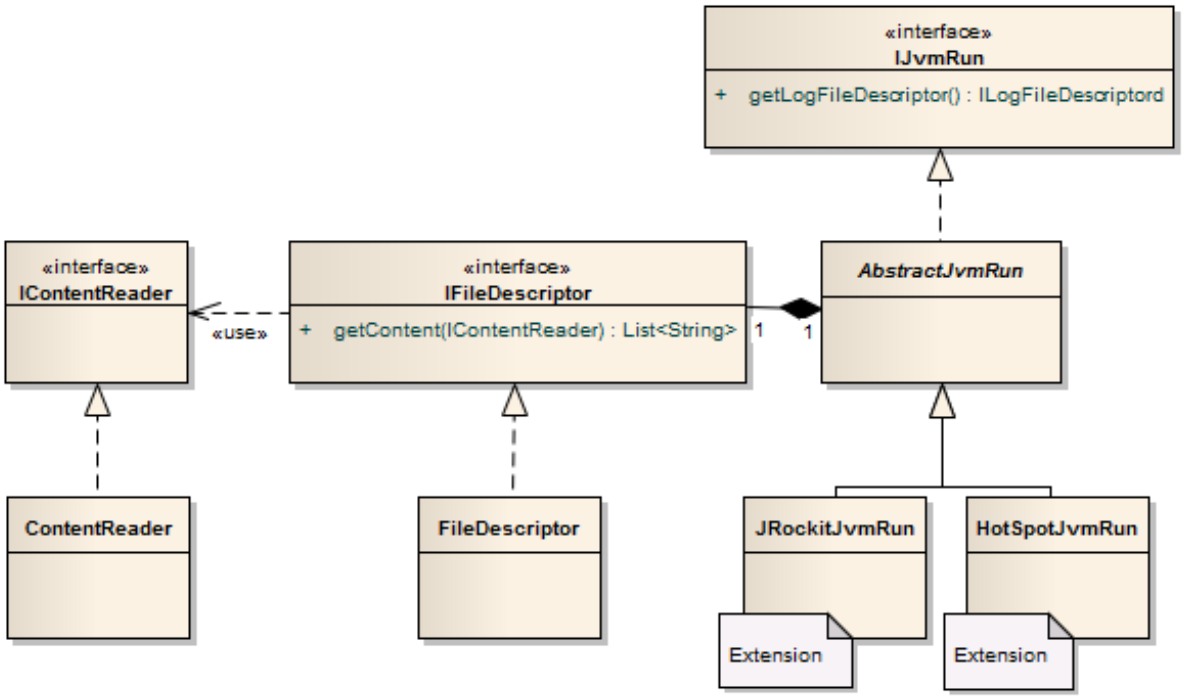

Abbildung 9.1: Domänenmodell: Eingelesene Datei

9.6 Logdatei parsen (FRQ-06)

Die Garbage Collection Logs der JRockit Virtual Machine bestehen aus Einträgen unterschiedlicher Log Module. Diese Ausgaben können selektiv per Kommandozeile aktiviert werden (siehe Abschnit 5.5.3). Für die Garbage Collection Analyse sind nicht alle Einträge relevant, es werden nur die wichtigsten verwendet. Die einzelnen Parser 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 eines Parser auf der Basis von Regulären Ausdrücken kann mit Java Standardmitteln gemacht werden.
- Proprietär: Parser-Generatoren sind proprietär und die Verwendung dessen bedingt gute Kenntnisse. In der Regel beschreibt man das Format in einer Grammatik.

9.6.1 Parser

Der Parser für die Logdateien der JRockit Virtual Machine ist nach dem Chain-of-Responsibility Pattern[13] aufgebaut. Pro Log-Eintrag (respektive pro Typ eines Log-Eintrags) kann ein Parser in die Parser-Kette geschaltet werde. Jeder Parser extrahiert die für ihn wichtigen Informationen und aktualisiert damit das Domänenmodell. Die wichtigsten Einträge der Garbage Collection Logdatei sind die des Memory Modules und werden vom *MemoryModuleParser* verarbeitet. In zukünftigen Versionen ist die Verarbeitung von Log-Einträgen von anderen Log-Modulen denkbar - Einträge des Nursery- oder Allocation-Modules, etc. Ablauf und Aufbau des Parsers sind in den folgenden beiden Grafiken ersichtlich:

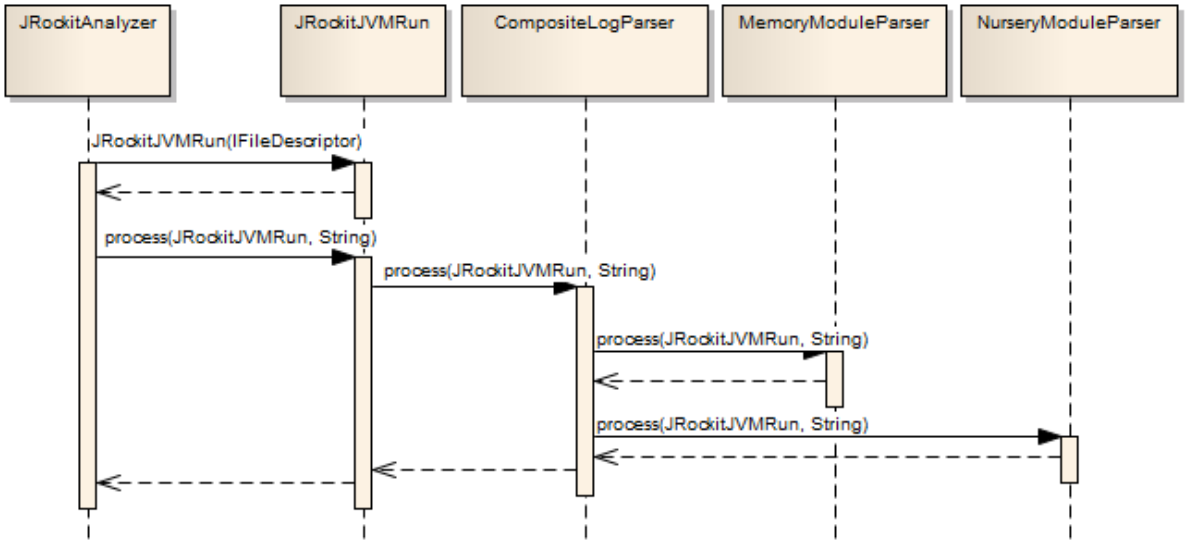

Abbildung 9.2: Sequenzdiagramm Parser

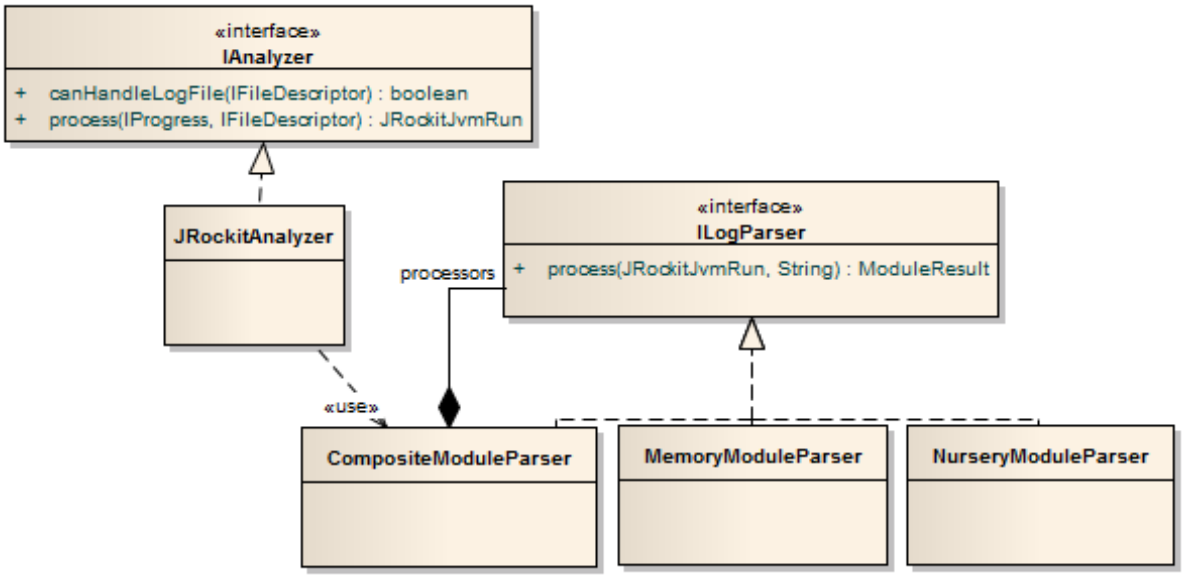

Abbildung 9.3: Klassendiagramm Parser

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-Funktion[2] extrahiert.
- **Syntactic Analyzer:** Der syntaktische Analyser verarbeitet die vom Lexer extrahierten Tokens. Er führt zum einen eine semantische Validierung durch und speichert die Werte in strukturierter Form (Domänenmodell).

9.6.2 Auswertung Logdatei

Bereits in Abschnitt 5.5.2 ist ein Teil einer Garbage Collection Logdatei aufgelistet. Dieser Abschnitt zeigt die für die Garbage Collection Analyse wichtigsten Informationen und Auswertungen die mit der Analysesoftware dafür gemacht werden können, ohne dabei aber auf die Ausgaben des Debug-Log-Levels[3] einzugehen.

[2]Mittels Gruppen innerhalb von Regulären Ausdrücken lassen sich einzelne Werte aus einem String extrahieren.

[3]Der Log-Level kann für jeden einzelnen Logger eingstellt werden.

Garbage Collection Algorithmus

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

Listing 9.1: 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 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, Genertional 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 9.2: 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 grösser des Young-Spaces (Nursery). Die grösse des Old-Spaces (Tenured Space) errechnet sich aus der Differenz zwischen Young-Space und maximaler Kapazität.

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 9.3: 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 9.4: 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 9.5: 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.

9.6.3 Domänenmodell JRockit Garbage Collection

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

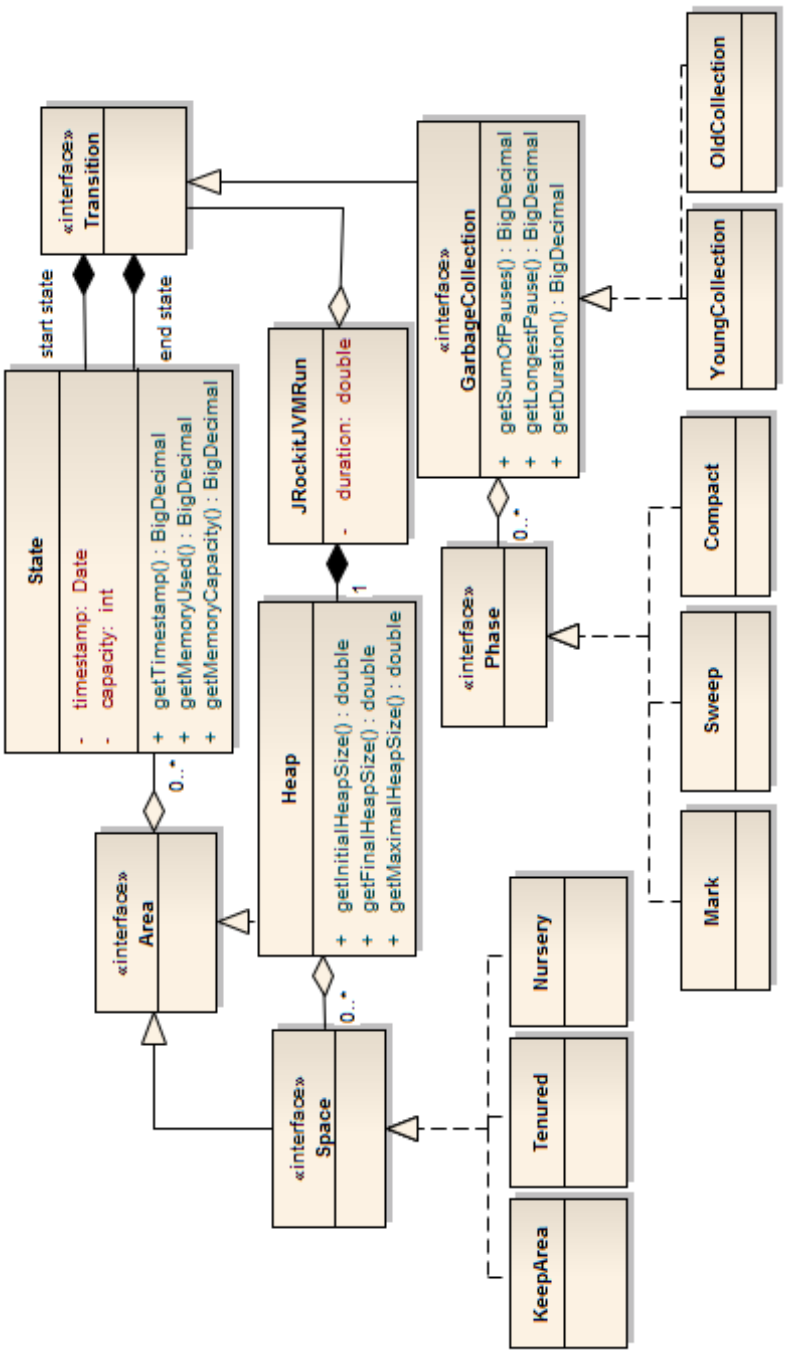

Abbildung 9.4: Domänenmodell: Garbage Collection (JRockit Implementation)

Die Daten der Logdatei 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 welche durch die Logdatei repräsentiert werden. Die Daten sind also nicht zu jedem Zeitpunkt, sondern nur für diese Zustände bekannt. Übergänge 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 Lauzeitumgebung ausgelöst - zum Beispiel wenn die Nursery oder die Old Generation 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.

9.7 Standardauswertung anzeigen (FRQ-07)

Von der Analysesoftware werden Standardprofile zur Verfügung gestellt, welche vom Benutzer nicht an seine eigenen Anforderungen angepasst werden können. Ein Doppelklick auf die Logdatei startet die Standardanalyse und öffnet entsprechend das Fenster mit den Tabs *Übersicht*, *Heap Benutzung* und *Zeit Garbage Collection*. Die Aktion bedingt ein enges Zusammenspiel zwischen Basissoftware und JRockit Erweiterung: Beim öffnen einer Analyse wird die zuständige Extension gesucht[4], welche den Inhalt[5] der Logdatei verstehen kann. Die Erweiterung ist für das Parsen und Aufbereiten der Daten und die Anzeige in einem Analysefenster zuständig. Zur illustration dient das folgende Sequenz-Diagramm:

[4] Extensions registrieren sich via das plugins.xml an einem Extension-Point.

[5] Der Inhalt der Datei wird lazy via die Methode getContent und einem ContentReader geladen.

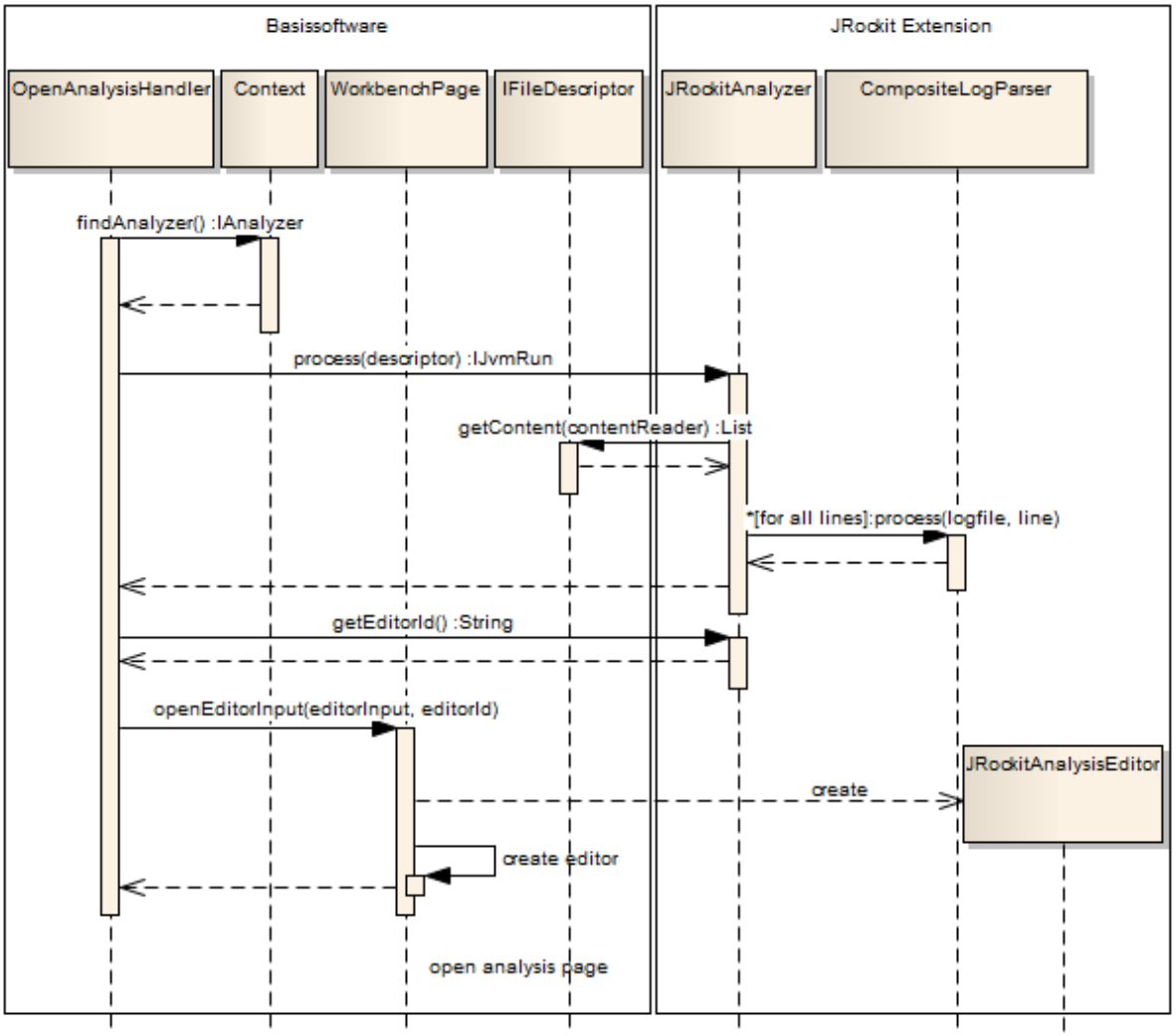

Abbildung 9.5: Sequenz-Diagramm Öffnen der Analyse

Der Ablauf der Garbage Collection Analyse ist folgendermassen:

1. Via Context wird die Erweiterung[6] gesucht.
2. Der Analyser der Erweiterung interpretiert den Inhalt und speichert die Daten im eigenen Domänenmodell.
3. Der Editor wird geöffnet.

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

9.8 Anzeige Übersicht Garbage Collection (FRQ-08)

Der initiale Tab der Analyseseite zeigt eine Zusammenfassung der geöffneten Garbage Collection Logdatei. 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 Collections)

- Letzte Garbage Collection: Zu welchem Zeitpunkt hat die letzte Garbage Collection statt gefunden.
- 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

9.9 Anzeige Heap Benutzung (FRQ-09)

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.

9.9.1 Datenquellen

Dieser Abschnitt zeigt auf, wie die Daten für das Chart zur Anzeige der Heap Benutzung aus dem Domänenmodell geladen werden.

Achse	**Beschreibung**	**Datenquelle**[7]
X-Achse	Auf der X-Achse wird die Zeit für jeden Zustand (*State*) angezeigt.	State.getTimestamp()
Y-Achse	Auf der Y-Achse wird der Verbrauch an Arbeitsspeicher angegeben.	State.getMemoryUsed()

9.10 Anzeige Dauer Garbage Collection (FRQ-10)

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

9.10.1 Datenquellen

Dieser Abschnitt zeigt auf, wie die Daten für das Chart zur Anzeige der Dauer einer Garbage Collection aus dem Domänenmodell geladen werden.

Achse	**Beschreibung**	**Datenquelle**
X-Achse	Auf der X-Achse wird die Zeit für jeden Zustand (*State*) angezeigt.	State.getTimestamp()
Y-Achse	Auf der Y-Achse wird der Verbrauch an Arbeitsspeicher angegeben.	State.getTransitionEnd(). getDuration()

[7] Zeigt den Pfad auf, wie auf die Daten zugegriffen wird. Siehe Abschnitt 9.6.3

9.11 Profil erstellen (FRQ-11)

Die Ansicht Profile zeigt die vom Benutzer erstellten Profile[8]. Die Beschreibung eines Analysefensters (Name, Beschreibung, Charts mit Serien, etc.) wird anhand von Profilen definiert. Es gibt folgende zwei Arten:

- **Unveränderliches Profil:** Das Standard-Profil ist aktuell das einzige unveränderliche Profil. Es wird durch die *JRockit Extension* definiert und kann nicht konfiguriert, verändert werden.
- **Veränderliches 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.

9.11.1 Domänenmodell zur Persistierung der Profile

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 ins Chart gelesen werden.

[8] Initial befindet sich darin allerdings nur das Standard-Profil.

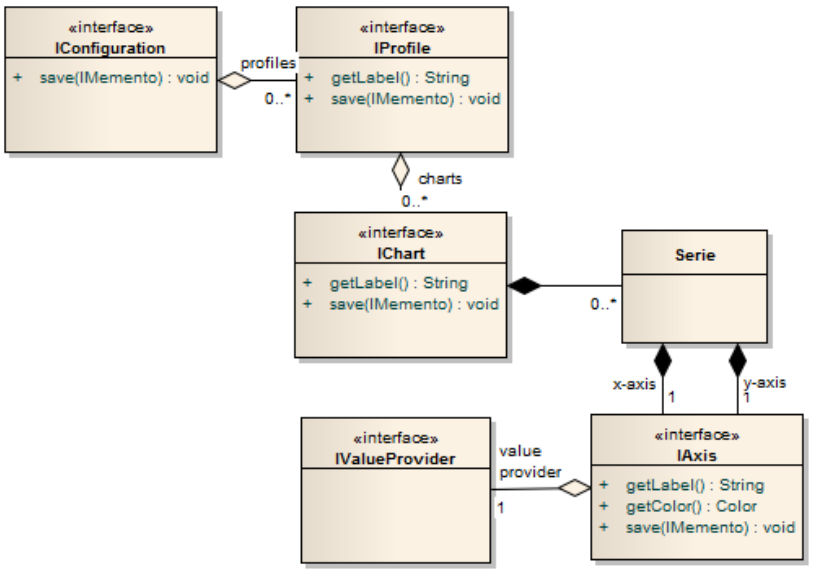

Abbildung 9.6: Domänenmodell: Profile

9.12 Charts definieren (FRQ-12)

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

9.13 Profil speichern (FRQ-13)

Mittels des Memento-(siehe Anhang A.1) und Visitor-Patterns[3, S. 331] werden die Profile gespeichert.

9.14 Profil exportieren, importieren (FRQ-14/ FRQ-15)

Die Analysesoftware stellt zum Sichern und Verteilen von Profilen einen Import-Export-Mechanismus bereit. Beide sind in den Eclipse Standard-Wizards er-

sichtlich oder können via rechtem Mouseklick (Ansicht Profile) gestartet werden. Der Profile-Export und -Import basiert wie das Speichern auf dem im Abschnitt A.1 beschriebenen Memento-Mechanismus. Zur Serialisierung in eine Datei wird das XMLMemento verwendet.

Kapitel 10

Konzept Qualitätsanforderungen

10.1 Hilfesystem (FRQ-16)

Das Hilfesystem der Eclipse Entwicklungsumgebung ist als Client-Server-Lösung implementiert. Beim Start der Entwicklungsumgebung wird zusätzlich ein Jetty-Server gestartet, der die Hilfedienste 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.

10.2 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 Fragment[1] gemacht wird. Als Testing-Framework wird der defakto Standard JUnit verwendet. JUnit ist sowohl von allen Entwicklungsumgebungen als auch von Maven und Tycho unterstützt.

[1] Fragmente haben vollen Zugriff auf ihre Host-Bundles.

10.3 Internationalisierung (QRQ-S-03))

Die Analysesoftware kann in den Sprachen Deutsch und Englisch gestartet werden. Die gewählte Sprache wird von der Entwicklungsumgebung übernommen[2] 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.

10.4 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:

- **Nebenläufigkeit:** Die Applikation startet die Arbeit in einem eigenen nicht-UI Thread[3], 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.

[2] Die Entwicklungsumgebung übernimmt die Sprache der Java Lauzeitumgebung: Voreinstellung oder Auswahl über Kommandozeile (-nl de).

[3] Einem Thread der nicht für das Zeichnen des Benutzerinterfaces verwendet wird.

10.5 Korrektheit (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 11

Infrastruktur

11.1 Verwendete Werkzeuge

11.1.1 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 Tycho[1] verwendet.

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

- Maven folgt dem Prinzip "Convention over Configuration"[2] - die Konfiguration des Builds wird dadurch wesentlich einfacher.
- Maven ist de facto Standard bei den Build-Werkzeugen.

11.1.2 Issue Tracker

Als Issue Tracker wird Jira verwendet. Es handelt sich dabei um eine kostenpflichtige aber relativ günstige Software für die Verwaltung von Bugs, Feature-Requests, etc. Sie lässt sich auch gut mit allen gängigen Systemen zur Versionsverwaltung integrieren.

11.1.3 Versionsverwaltung

Als Versionsverwaltungssysteme kommen mehrere Werkzeuge in Frage, darunter befinden sich auch CVS und Subversion. Git[3] ist ein verteiltes Versionsver-

[1] Im Bereich der Eclipse Rich Client Entwicklung kann entweder PDE Build, ein auf Apache Ant basiertes Build-System für Eclipse RCP Applikationen[12] oder die Maven-Integration Tycho (http://tycho.sonatype.org) verwendet werden.

[2] Das Prinzip "Convention over Configuration" hat zur Folge, dass im Wesentlichen nur von den Standardeinstellungen abweichende Werte konfiguriert werden müssen.

[3] http://git-scm.com

waltungssystem und ist konzeptionell und hinsichtlich Benutzerfreundlichkeit besser als Subversion und CVS[4]. Die Charakteristik des Systems erfordert es nicht unbedingt, dass es ein zentrales Repository gibt. Ein Repository befindet sich auf jedem Client. Trotzdem setzt man in der Regel ein zentrales Repository auf, in welches die Entwickler jederzeit commiten können. Eine empfehlenswerte Einführung in Git kann unter [2] gefunden werden.

Auf der Plattform Github[5] kann man Git-Projekte frei "hosten".

11.1.4 Continous 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 - beispielsweise jede Stunde - durchführen. Für dieses Projekt sind einige Voraussetzungen gegeben:

- **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 Hudson[6] 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.

11.2 Konfigurationsmanagement

Jedes Deployment entspricht einer Version in Jira (siehe Abschnitt 11.1.2). Ab dem ersten Release entspricht die Versionsnummer folgendem Muster:

<Major>.<Minor>.<Build>

Ein Beispiel für eine Versionsnummer wäre 1.3.1. Gestartet wird mit 1.0.0.

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

[5] http://github.com

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

11.2.1 Pflege der einzelnen Versionen

- **Build:** Bei jedem Bugfix- und/oder sonstigen ausserordentlichen Zwischenrelease, wird die Build-Version um eins hochgezählt.
- **Minor:** Bei jedem Feature-Deployment während einem ordentlichen Release, wird die Minor-Version um eins hochgezählt.
- **Major:** Bei grossen Änderungen und/oder Entwicklungen von Zusatzmodulen wird die Major-Version um eins hochgezählt.

11.2.2 Versionsverwaltung

Das zentrale Repository dieses Projektes befindet sich auf Github[7]. Es ist jederzeit verfügbar und kann von allen Clients (Entwickler, Continuous Integration) jederzeit angesprochen werden. Ein neuer Entwickler klont das Repository auf seinen Rechner, um am Sourcecode arbeiten zu können.

Übersicht

Das Repository besteht aus drei Branches:

- **master:** Auf dem *master*-Branch befindet sich der aktuelle Entwicklungsstand.
- **release:** Nur vom *release*-Branch wird in die Produktion deployt. Dadurch kann jederzeit am produktiven System ein Bugfix gemacht werden. Der *release*-Branch entspricht der aktuell deployten produktiven Applikation.
- **next:** Auf dem *next*-Branch werden zukünftige Features entwickelt, die noch nicht in die Produktion deployt werden sollen.

Bugfix Entwicklung

Ein Bugfix der produktiven Applikation findet auf der Basis des *release*-Branches statt. Dazu wird daraus allerdings ein neuer Branch erzeugt. Die Minor-Version für den Bugfix wird um eins inkrementiert. Ausgehend von einem Beispiel zweier Bugfixes für die Version 1.0.0 ist der Ablauf wie folgt:

1. Für den laufenden Release (Version 1.0.0) werden die Bugs (#4321, #4322) im Jira rapportiert.
2. Der Entwickler checkt den *release*-Branche aus und erstellt daraus einen neuen Branch (*bugfix-for-1.0.0*).
3. Der Entwickler inkrementiert die Maven-Version um eins (neu 1.0.1).
4. Der Entwickler erstellt eine neue Jira Version (1.0.1).

[7] https://github.com/schmidic/bachelorthesis

5. Der Entwickler repariert die Fehler, die Applikation wird getestet.
6. Die Bugs innerhalb des Jiras werden der neuen Version zugeordnet.
7. Der Entwickler führt (merge) die Bugfixes zurück in den *release*-Branch. Die Zuordnung der Commits mit den Jira-Bugs wird über den Kommentar in der Versionsverwaltung mit dem Bug im Jira verlinkt. Die Anpassung des Kommentars der vorherigen Commits kann mittels einem Rebasing (siehe [2]) gemacht werden.
8. Erstellen eines Tags für den Release
9. Release der neuen Version (1.0.1).

Mit dem dedizierten *release*-Branch kommt ein wesentlicher Vorteil zum Tragen: Man kann ein Bugfix Release in das Produktivsystem veranlassen, der nur einen bestimmten Bugfix enthält, selbst wenn auf dem *master* eventuell schon weitere Features entwickelt wurden.

Feature Entwicklung

Die Entwicklung eines Features, welches in die Produktion einfliessen soll, wird auf dem Master-Branch durchgeführt. Um Konflikte mit dem *release*-Branch zu vermeiden, wird nach jedem Release zumindest die Minor-Version des *master*-Branches um eins erhöht. Das Vorgehen ist wie folgt:

1. Für den laufenden Release gibt es die Feature-Requests (#4351, #4352) im Jira.
2. Der Entwickler checkt den *master*-Branch (Version 1.0.0) aus und erstellt daraus einen neuen Branch (*feature-1.1.0*)
3. Der Entwickler inkrementiert die Maven-Version (1.1.0).
4. Der Entwickler erstellt eine neue Jira Version (1.1.0).
5. Der Entwickler erstellt das neue Feature.
6. Der Entwickler ordnet die Feature-Requests im Jira der neu erstellten Version zu.
7. Der Entwickler führt (merge) den *feature-1.1.0*-Branch zurück in den *master*-Branch. Die Zuordnung der Commits mit den Jira-Bugs wird über den Kommentar mit dem Jira-Bug verlinkt. Die Anpassung des Kommentars der vorherigen Commits kann mittels einem Rebasing (siehe [2]) gemacht werden.

Teil V

Umsetzung

Kapitel 12

Implementation

Dieser Abschnitt zeigt auf, wie die Anforderungen und das Konzept im Proof of Concept umgesetzt wurden. Detailliertere Informationen befinden sich auch in der Benutzeranleitung im Anhang B.

12.1 Funktionale Anforderungen

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

Installation und Update der beiden Features *Core* und *JRockit Extension* können über das Netzwerk gemacht werden[1]. Die Update-Funktionalität ist Bestandteil des Eclipse-Frameworks: beide Features werden über eine Update-Seite bereitgestellt. Die Features wiederum bestehen aus den einzelnen in Abschnitt 8.2.1 beschriebenen Projekten. Der Release der Artefakten wird durch das Continuous Integration System Hudson gemacht.

12.1.2 Datei importieren (FRQ-03)

Die Dateien werden in eine eigene View importiert. Views sind programmierbare Komponenten (Ansichten) und können durch das Plugin deklarativ registriert werden. Sie stehen anschliessend dem Benutzer zur Verfügung. Das Speichern und Wiederherstellen des Zustands dieser View findet über das von Eclipse implementierte Memento-Pattern (siehe Anhang A.1) statt. Der Import findet über einen Wizard[2] statt. Der Benutzer wählt das Verzeichnis und die zu importierenden Logdateien aus.

[1] Aktuell befindet sich der Update-Server noch in einem nicht öffentlichen Netzwerk und ist nur via ein VPN verfügbar.

[2] Eclipse bietet die Möglichkeit, mit sehr geringem Aufwand Import- und Export-Wizards zu erstellen. Im Prinzip muss nur die GUI-Funktionalität implementiert werden.

12.1.3 Datei einlesen (FRQ-05)

Nachdem der Benutzer die Logdatei erfolgreich importiert hat, kann er sie mittels einem Doppelklick öffnen. Erst dann wird die Datei durch das Feature *Core* eingelesen und im Arbeitsspeicher als Liste abgelegt. Die Speicherung als Liste ermöglicht es den Parsern, sequentiell oder durch die Angabe des Zeilenindexes auf die Daten zuzugreifen.

12.1.4 Garbage Collection Logdatei parsen (FRQ-06)

Die Log-Ausgaben des Memory-Moduls (Log-Level: info) werden mittels Regulären Ausdrücken geparst und in die logische Form (siehe Abschnitt 9.6.3) gebracht. Die Analyse wird realisiert durch die Implementation eines Analyzers (*Analyzer*$<T>$) und bedingt daraus Implementation folgender Methoden:

- *boolean canHandleLogFile(IFileDescriptor)*
- *T process(IFileDescriptor descriptor, IProgress progress)*
- *String getEditorId()*.

Die Abstraktion durch das Interface *Analyzer*$<T>$ ist Grundlage für die Erweiterbarkeit um andere Log-Formate.

12.1.5 Standardauswertung anzeigen (FRQ-07)

Für die importierten Dateien besteht die Möglichkeit, sie in der Standardauswertung darzustellen. Die Standardauswertung zeigt die im Konzept definierten Ansichten (siehe Abschnitt 9.8):

- Übersicht Garbage Collection (FRQ-08)
- Heap Kapazität (FRQ-09)
- Dauer Garbage Collection (FRQ-10)

12.1.6 Profil erstellen (FRQ-11)

Profile können durch den Benutzer erstellt und an seine Bedürfnisse angepasst werden. Das Anpassen beinhaltet die Definition von eigenen Charts und deren Datenserien (FRQ-12). Das Speichern (FRQ-13), Exportieren (FRQ-14) und Importieren (FRQ-15) wird über das Memento-Pattern gemacht. Für weiter Informationen zum Memento siehe Anhang A.1.

12.1.7 Hilfesystem (FRQ-16)

Das Hilfesystem wurde sowohl für die contextsensitive wie auch die indexbasierte Hilfe implementiert und kann stetig durch weitere Inhalte erweitert werden.

Aktuell sind alle existierenden Hilfeseiten in den Sprachen Deutsch und Englisch vorhanden, diese können aber ohne Entwicklungsaufwand auch um weitere Sprachen ergänzt werden. Die Sprache wird durch den Benutzer in der Datei *eclipse.ini* definiert oder ist die Standard-Sprache der Eclipse-Distribution.

12.2 Qualitätsanforderungen Software

12.2.1 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. Die Erweiterung um ein weiteres Dateiformat bedingt folgende Schritte:

- erstellen eines Plugins
- erstellen eines Features
- Konfiguration des neuen Features für die Update-Seite
- Realisation des *Analyzer*s und Konfiguration als Extension für den Extension-Point *com.trivadis.loganalysis.analyzer*
- erstellen des Auswertungsfensters, dafür können Super-Klassen der Basissoftware verwendet werden

12.2.2 Testabdeckung (QRQ-S-02)

Die Testabdeckung wurde zum jetzigen Zeitpunkt noch nicht ausgewertet, aber entspricht lange nicht den Anforderungen von 80 Prozent.

12.2.3 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.

12.2.4 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 abbrechen kann.

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

Alle zu berechnenden Werte befinden sich im Datentyp *BigDecimal*. Dies führt zwar zu einem erhöhten Verbrauch an Arbeitsspeicher, ist aber für die geforderte Genauigkeit unumgänglich.

Kapitel 13

Review und Ausblick

13.1 Was das Tool leistet

Wenn die Auslastung einer CPU konstant hoch ist aber nicht durch den Kernel verursacht wird, kann unter Umständen das Tuning des Garbage Collectors Sinn machen. Die folgenden Abschnitte zeigen wann und wie die Analysesoftware verwendet werden kann.

13.1.1 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.**

13.1.2 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.

13.1.3 JRockit Version

Version R28 ist die neuste Version der JRockit. Die Analysesoftware wurde auf das Format dieser Virutellen Machine ausgerichtet. Die Analyse von Logdateien älterer Versionen ist momentan noch nicht möglich.

13.2 Ausblick

13.2.1 Funktionsumfang

Daten vergleichen

Beim Performance Tuning im Bereich des Garbage Collectors vergleicht man oft die Implikation einer Anpassung der Konfiguration. Aktuell können zwar verschiedene Logdateien gleichzeitig geöffnet werden, der Vergleich von Kurven im gleichen Diagramm ist beispielsweise aber noch nicht möglich und wäre ein denkbares nächstes Feature.

Selektierbare Zeitachse

Man ist beim Performance Tuning in der Regel nicht an den Informationen über die ganze Zeit interessiert, die Möglichkeit zur Wahl von Start- und Endzeitpunkt der a Daten wäre deshalb eine wichtige Erweiterung.

13.2.2 Weitere Daten

Auswertung Debug Informationen JRockit R28

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

```
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 13.1: Garbage Collection Log (Debug Informationen)

Einige Beispiele für solche Informationen sind:

- **Gründe, warum eine Garbage Collection gestartet wurde:** Die erste Zeile im Listing oben zeigt, dass es hängige Anfragen für die Allokation von Speicher gibt. Was ein Grund ist, warum die Garbage Collection gestartet wurde.
- **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 einzelnen Phasen und im speziellen die Final Marking Phase gedauert haben.

13.2.3 Andere Log-Formate

JRockit R27

Wie bereits erwähnt, können die Logs der Version R27 noch nicht ausgewertet werden. Weil diese Version noch an vielen Orten im Einsatz ist, würde die schnelle Kompatibilität mit R27 Sinn machen.

G1 Algorithmus

Ab Version 1.6.0_14 des Java Runtime Environments ist eine Vorversion des G1 Garbage Collectors[1] verfügbar. Die Funktionsweise dieses Algorithmus unterscheidet sich, auch im Bereich des Log-Formats, stark von den bisherigen Versionen des Mark & Sweep Algorithmus. Für die Auswertung solcher Dateien gibt es aktuell noch kein Werkzeug, die Implementation dieses Formats wäre deshalb spannend.

[1] G1 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 Packete dieses Host-Bundles. 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 Werte eines Bedienelementes mit dem hinterlegten View-Model zu verbinden. Änderungen am Wert des Eingabefeldes werden beispielsweise an das Model propagiert - sofern es sich um ein bidirektionales Binding handelt, auch umgekehrt.	-

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 unterschiedlichen Algorithmen gemacht.	Speichermanagement
Graphical User Interface (GUI)	Im Unterschied zu einer textbasierten Benutzeroberfläche ist ein GUI grafisch. Die Bedienung findet nebst der Tastatur über die Maus statt.	-
Old Collection	Eine Old Collection bezeichnet die Garbage Collection auf der Old Generation.	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
Old Space	siehe Old Generation	siehe Old Generation
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 Framework	Software (-paket) mittels welchem sich Rich Client Applikationen entwickeln lassen.	

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	Eine Young Collection bezeichnet die Garbage Collection auf der Young Generation.	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
Young Space	siehe Young Generation	siehe Young Generation
User Interface	Als User Interface (Benutzeroberfläche) werden alle Schnittstellen zwischen Mensch und Maschine bezeichnet. Diese können textbasiert oder grafisch sein (GUI).	

Tabelle 13.1: Glossar

Abkürzungen

Abkürzung	Begriff	Beschreibung
BIRT	Business Intelligence and Reporting Tool	Open-Source-Framework für Business Intelligence und Reporting Anwendungen
CPU	Central Processing Unit	Zentrale Recheneinheit eines jeden Computers, Servers, etc.
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 13.2: Glossar

Literaturverzeichnis

[1] Joshua Bloch. Effective Java. Addison-Wesley Java series. Addison-Wesley, 2008.

[2] Martin Dilger. Besser gits nicht! Java Magazin, 11:100–105, 2011.

[3] 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.

[4] 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.

[5] Marcus Lagergren and Marcus Hirt. Oracle Jrockit: The Definitive Guide. Packt Publishing, Limited, 2010.

[6] Angelika Langer and Klaus Kreft. Generational garbage collection. Java Magazin, 3:26–30, 2010.

[7] Angelika Langer and Klaus Kreft. Java Core Programmierung. Entwickler Press, 2011.

[8] Sun Microsystems. Memory management in the hotspot virtual machine. http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf, 2006.

[9] Oracle. Oracle jrockit command-line reference. http://download.oracle.com/docs/cd/E15289_01/doc.40/e15062.pdf, 2011.

[10] Kerk Pepperdine. Concurrent and performance reloaded. http://www.jfokus.se/jfokus/page.jsp?id=recordings#page=page-1, 2011.

[11] 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.

[12] Lars Vogel and Dominik Zapf. Eclipse pde build - tutorial. http://www.vogella.de/articles/EclipsePDEBuild/article.html#overview, 2008.

[13] Wikipedia. Chain-of-responsibility pattern — wikipedia, the free encyclopedia, 2011. [Online; accessed 11-September-2011].

[14] Wikipedia. Eclipse (ide) — wikipedia, die freie enzyklopädie. http://de.wikipedia.org/w/index.php?title=Eclipse_(IDE)&oldid=92563983, 2011. [Online; Stand 29. August 2011].

Listings

Abbildungsverzeichnis

Tabellenverzeichnis

Teil VI

Anhang

Anhang A

Eclipse Rich Client Framework

A.1 Zustand View speichern

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 Four[1] in [3, 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)*.

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

Anhang B

Bedienungsanleitung

B.1 Installation der Software

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

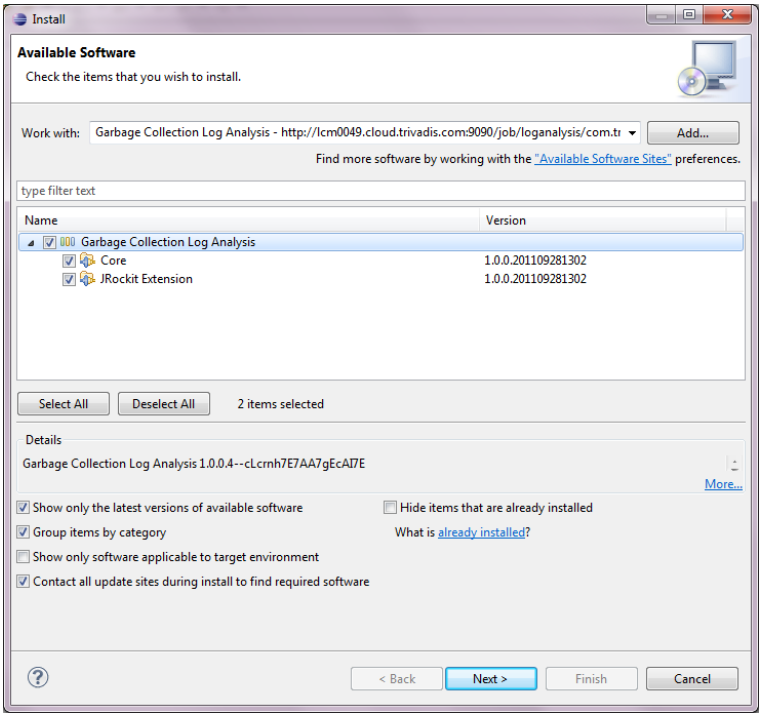

Abbildung B.1: Installation Garbage Collection Log Analyse

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.

B.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.

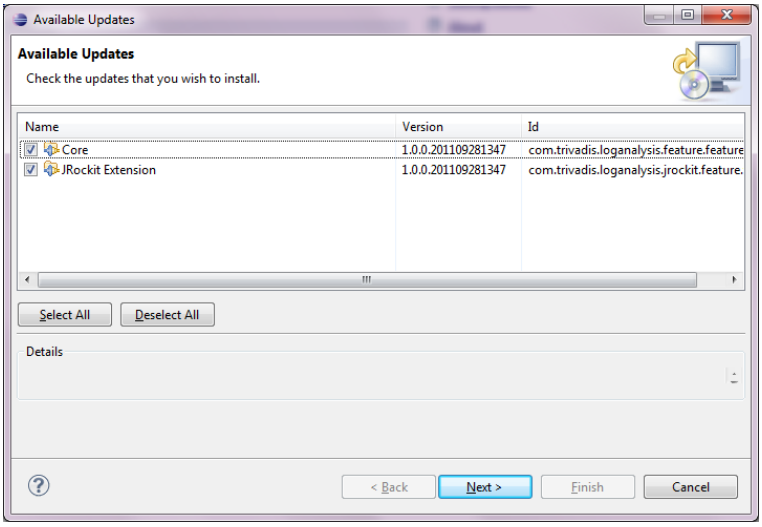

Abbildung B.2: Update Garbage Collection Log Analyse

B.3 Dashboard

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

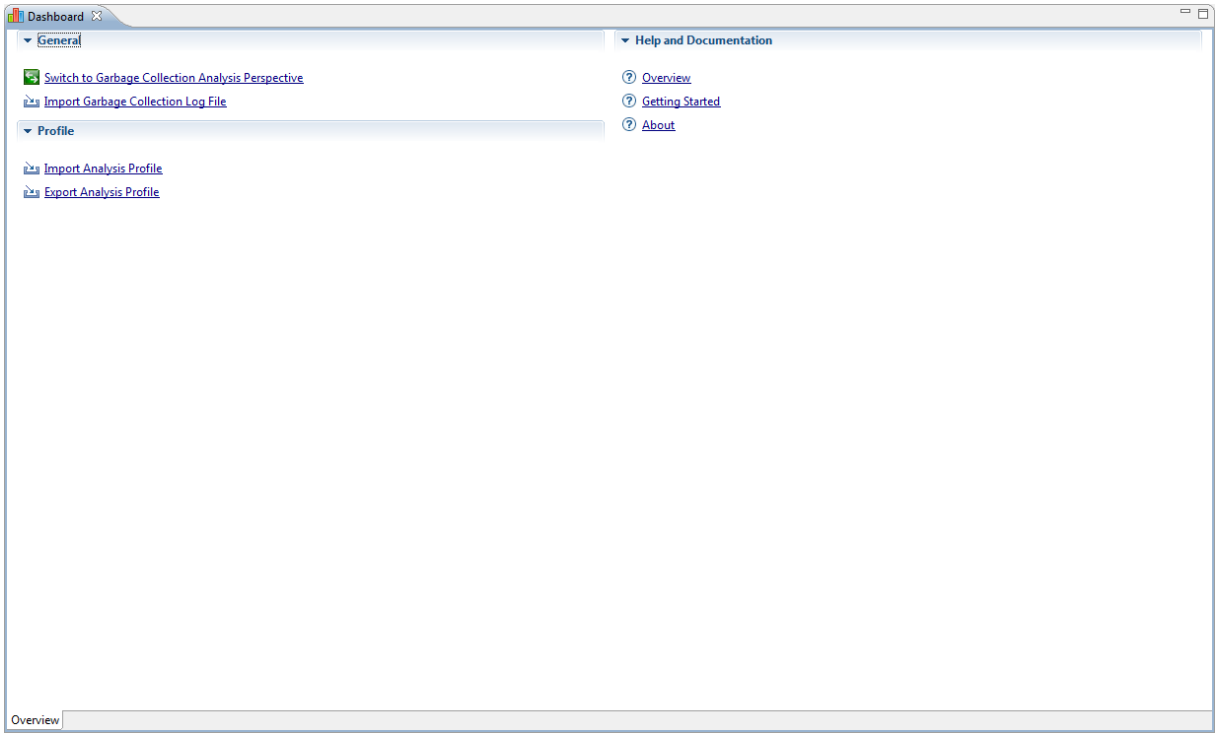

Abbildung B.3: Dashboard

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

B.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.

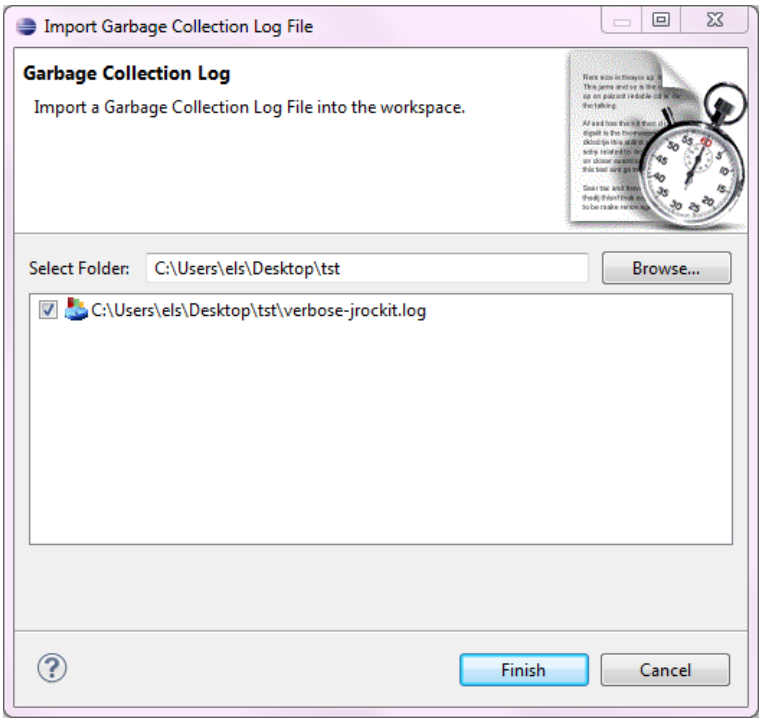

Abbildung B.4: Profil importieren

B.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.

B.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.

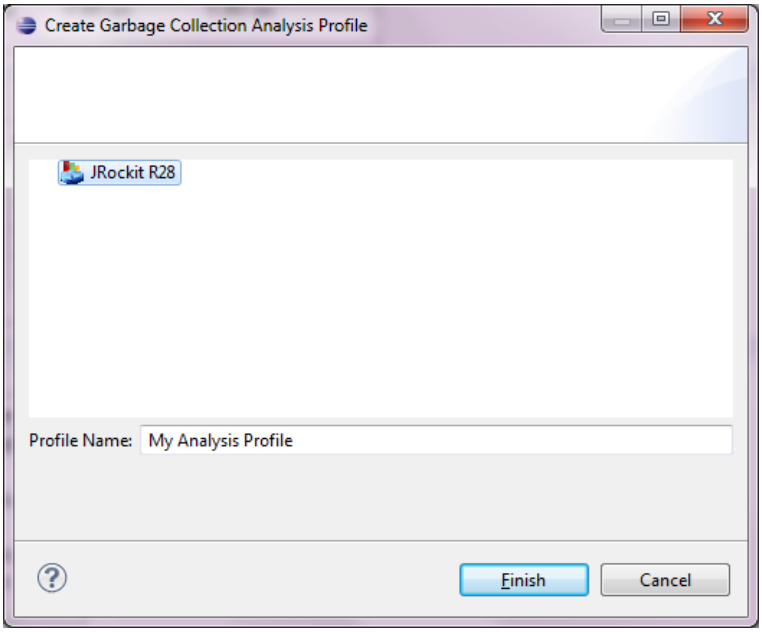

Abbildung B.5: Profil erstellen

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

B.5.2 Profil exportieren

Mit Rechtsklick auf die Ansicht *Profile* und Auswahl von *Profil exportieren*, können die Profile in eine Lap[1]-Datei exportiert werden.

[1] Lap steht für Log Analysis Profile

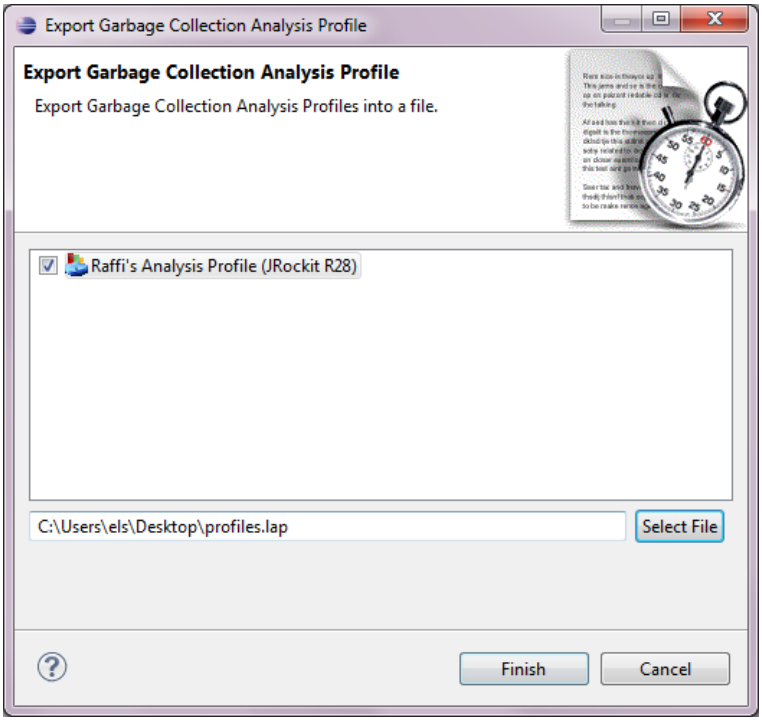

Abbildung B.6: Profil exportieren

B.5.3 Profil importieren

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

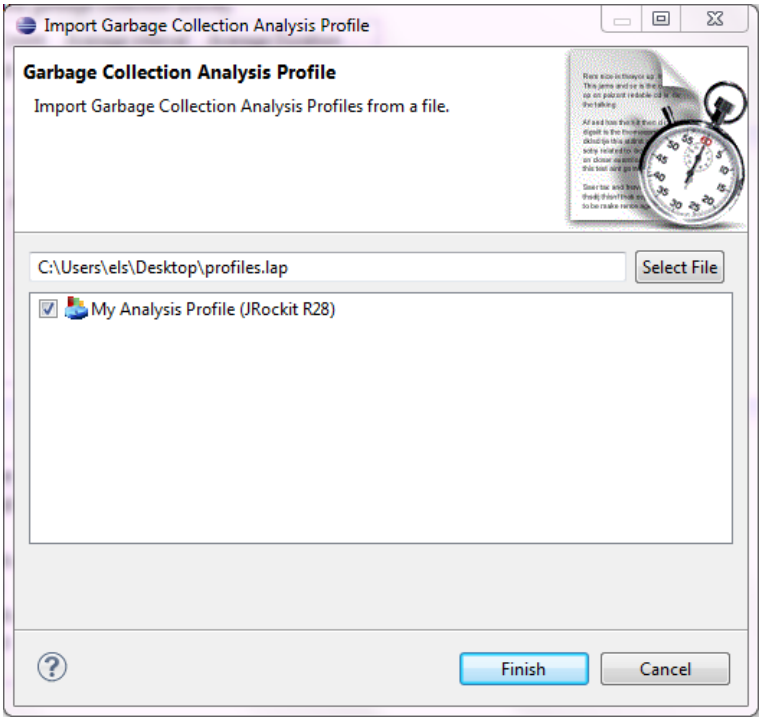

Abbildung B.7: Profil importieren

B.6 Garbage Collection Analyse

B.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

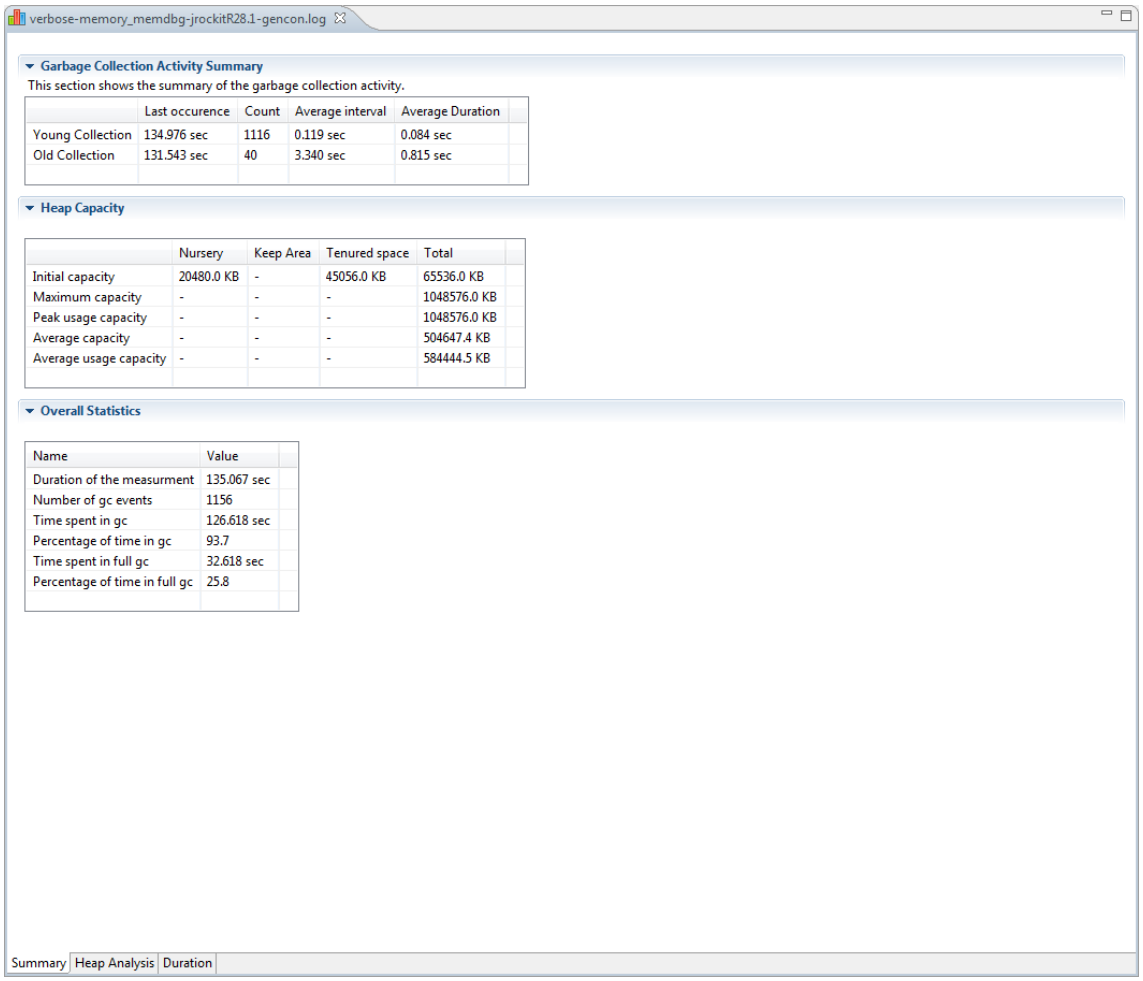

Abbildung B.8: Standardauswertung: Zusammenfassung

Heap Analyse

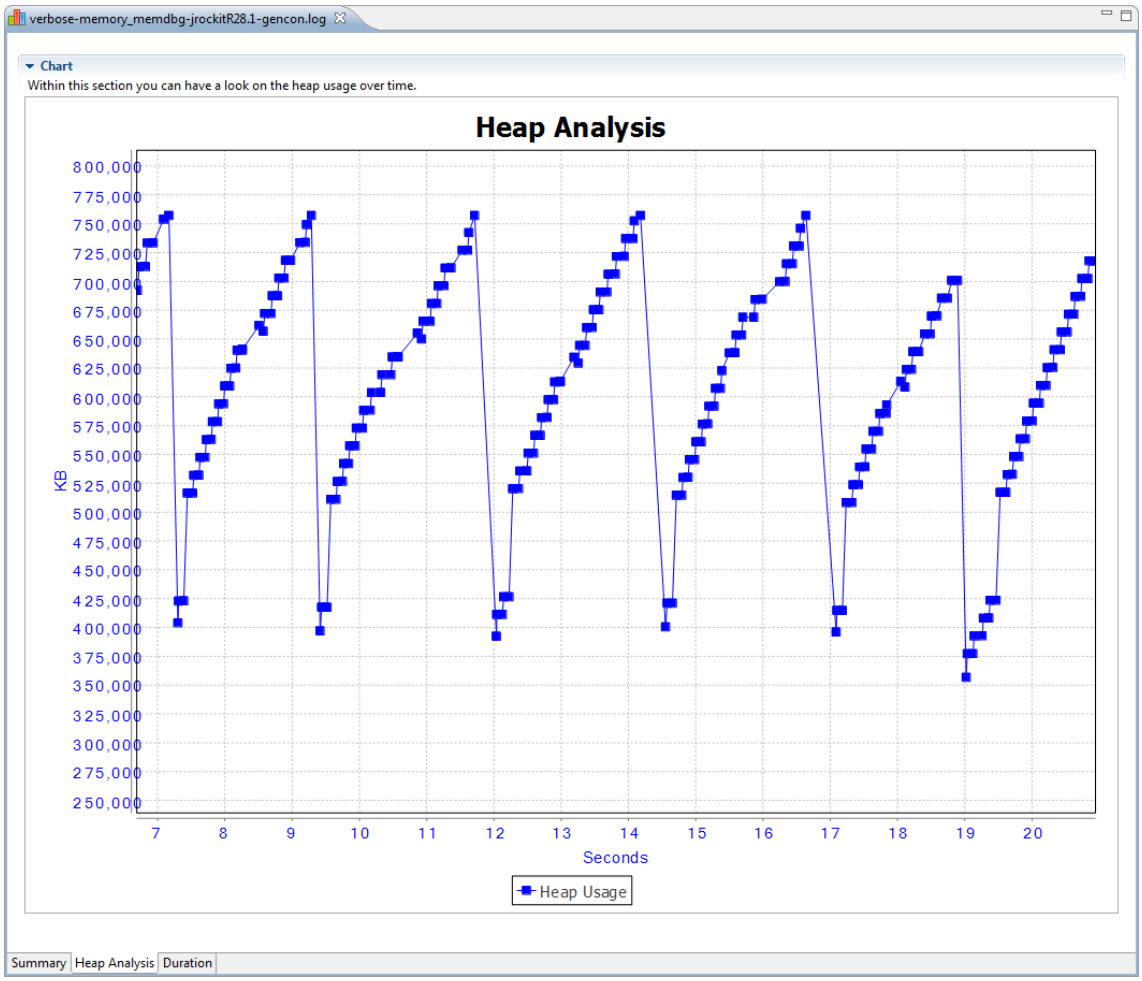

Abbildung B.9: Standardauswertung: Heap Analyse

Dauer Garbage Collection

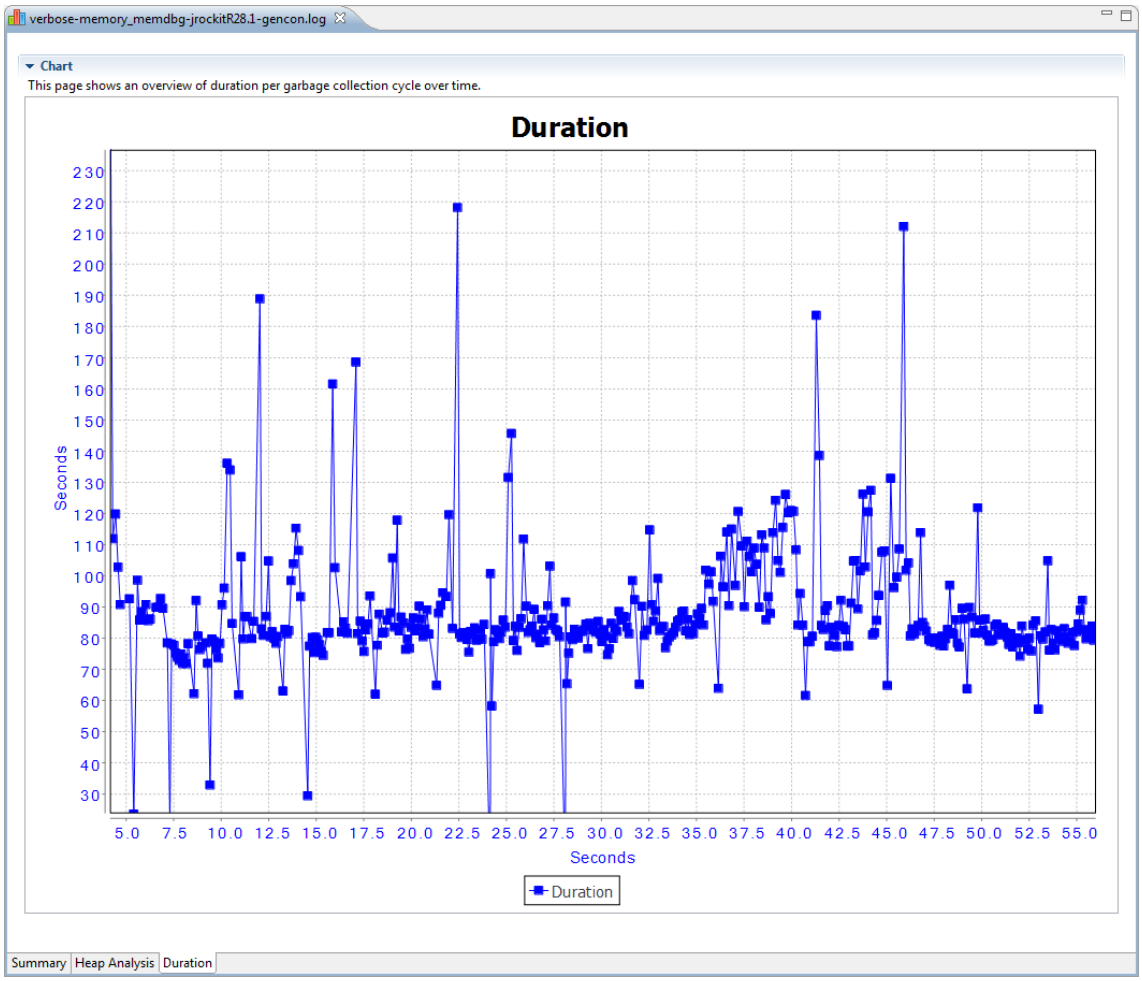

Abbildung B.10: Standardauswertung: Dauer Garbage Collection

B.6.2 Benutzerdefinierte Auswertung (Profile)

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

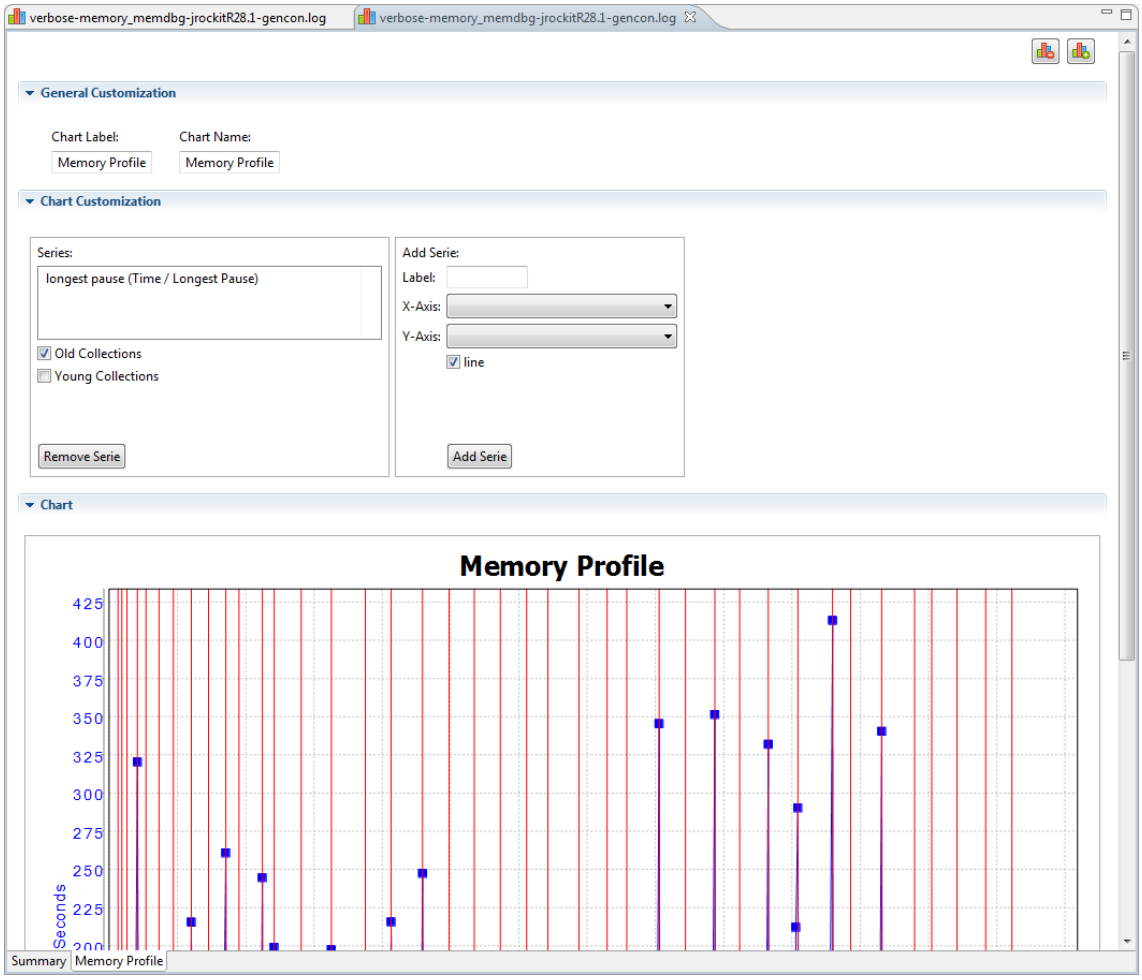

Abbildung B.11: Standardauswertung: Zusammenfassung

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 C

Informationen

C.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 C.1: Inhalt Datenträger

C.2 Repository

Der Quelltext und alle Dokumente befinden sich auf Github, einem öffentlich verfügbaren Git-Repository. Mittels folgendem Kommando[1] kann alles aus dem Repository ausgecheckt werden.

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

Listing C.1: Checkout Quelltext Repository

[1] Dafür ist die Installation der Software Git erforderlich.