Bachelorthesis

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

von

Raffael Schmid

Schule:	Hochschule für Technik, Zürich
Studiengang:	Informatik
Betreuer:	Matthias 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.

Hiermit bestätigt die oder der Unterzeichnende, dass die Bachelorarbeit mit dem Thema *Konzeption und Entwicklung eines Auswertungstools für die Logdateien des JRockit Garbage Collectors* gemäss freigegebener Aufgabenstellung mit Freigabe vom 07. Juni 2011 ohne jede fremde Hilfe im Rahmen der gültigen Reglements selbständig ausgeführt wurde.

Wettingen, den 6. Dezember 2011

Raffael Schmid

Inhaltsverzeichnis

Listings

Abbildungsverzeichnis

Tabellenverzeichnis

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 (Laufzeitumgebung) in der Version 28 keine Werkzeuge zur Automation dieser Aufgabe. 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 fünf 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.

Kapitel 1

Definitionen

Wichtige Begriffe

Dieser Abschnitt definiert die im Zusammenhang mit dieser Arbeit wichtigsten Begriffe. Das restliche Glossar befindet sich im Anhang A auf Seite 104.

Garbage Collection

Der Begriff *Garbage Collection* bezeichnet das Aufräumen von nicht mehr benutzten Objekten im Speicher.

Garbage Collection Algorithmus

Die verschiedenen Hersteller von virtuellen Maschinen haben für die Garbage Collection unterschiedliche Methoden, man spricht dabei auch von *Garbage Collection Algorithmen*.

Teil I

Projektbeschreibung

Kapitel 2

Aufgabenstellung

2.1 Ausgangslage

Für die Ermittlung von Java Performance-Problemen braucht es Wissen über die Funktionsweise der JVM (Java virtual Machine, Laufzeitumgebung der Java Plattform), deren Ressourcenverwaltung (Speicher, Input, Output, Prozessor) 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 (virtuellen Maschine, Laufzeitumgebung). Aktuell gibt es noch kein Tool, welches die Daten der Logs sammelt und grafisch darstellt.

2.2 Ziele der Arbeit

Ziel der Bachelorthesis ist die Konzeption und Entwicklung eines Prototypen für die Analyse von Garbage Collection 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.

2.3 Aufgabenstellung

Im Rahmen der Bachelorthesis werden vom Studenten folgende Aufgaben durchgeführt:

1. Studie der Theoretischen Grundlage im Bereich der Garbage Collection (generell und spezifisch JRockit virtual Machine)
2. Stärken- / Schwächen-Analyse der bestehenden 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

2.4 Erwartete Resultate

Die erwarteten Resultate dieser Bachelorthesis sind:

1. Detaillierte Beschreibung der Garbage Collection Algorithmen der Java virtual Machine im Generellen und spezifisch der JRockit
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

Kapitel 3

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 die Anforderungsanalyse durchgeführt. Aufbauend auf dieser kann die Stärken-/Schwächen-Analyse und die Evaluation der bestehenden Rich Client Framworks und Bibliotheken zur Generierung von Diagrammen gemacht werden. Es folgt die Konzeption, Implementation und das Review der Software. Weitere Einzelheiten zu den Teilaufgaben befinden sich in im weiteren Verlauf dieses Abschnittes.

3.1 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 wichtig, dass er die richtige Sicht auf die Daten der Garbage Collection hat und geeignet filtern kann. Voraussetzung für diese Anforderung ist die Kenntnisse der verschiedenen Garbage Collection Algorithmen, insbesondere der JRockit VM.

- **Konzept:** Die Konzeption des Domänen-Modells und des Parseprozesses der Logdateien bedingt eine gute Kenntnis des Aufbaus der VM im Bereich des Speichermanagements.

3.2 Anforderungsanalyse Software-Prototyp

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

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

3.3 Stärken-Schwächen-Analyse Rich Client Frameworks

Die Applikation wird als Rich Client implementiert. Als Basis kommen die Frameworks Netbeans und Eclipse in Frage. Die Stärken-Schwächen-Analyse soll den Entscheid bei der Evaluation der verwendeten Komponenten herleiten. Die Grundlage für diese Aufgabe ist die Anforderungsanalyse.

3.4 Evaluation Frameworks

Basierend auf der Stärken-Schwächen-Analyse wird ein Entscheid für das jeweilige Framework (Eclipse RCP oder Netbeans RCP) gefällt. Nebst der Auswahl der Rich Client Plattform muss auch eine Bibliothek zur Generierung von Diagrammen evaluiert werden.

3.5 Konzeption und Implementation

Basierend auf den Anforderungen wird das Konzept erstellt und anschliessend die Software im Sinne eines Proof of Concept implementiert.

3.6 Bewertung

Die Bewertung der Software wird auf Basis der Anforderungen gemacht.

Teil II

Grundlagen

Kapitel 4

Einführung Performanceanalyse

Als Vorbereitung für die Anforderungsanalyse und das Konzept werden verschiedene Grundlagen erarbeitet. Dieses und die weiteren beiden Kapitel dienen als kleine Einführung in das Thema des Performance und Garbage Collection Tunings.

Ganz nach dem Prinzip "never change a running system" soll an der Konfiguration der Garbage Collection nur dann etwas geändert werden, wenn man am Verhalten Probleme feststellen kann. Dies muss aber zuerst durch eine gezielte und strukturierte Performanceanalyse untersucht werden. Dieser Abschnitt beschreibt ein mögliches Schema zur Vorgehensweise aufbauend auf dem Model der *Suche nach dem Dominating Consumer* nach [10].

4.1 Motivation

Die Performance respektive Leistungsfähigkeit einer Applikation stellt eine der bedeutendsten Qualitätsanforderungen dar. Probleme in diesem Bereich führen zu verärgerten Benutzern und verlangsamen die Business-Prozesse. Die Motivation für eine Performanceanalyse entsteht, wenn die in diesem Bereich existierenden **Qualitätsanforderungen nicht erfüllt** sind. Optimal wäre, wenn diese Frage während des Entwicklungsprozesses kontinuierlich geprüft würde.

4.2 Ablauf

Die Performanceanalyse ist ein iterativer Prozess und dauert in der Regel so lange, bis die Anforderungen an das System erfüllt sind. Eine einzelne Iteration besteht aus vier Schritten[4]:

1. Identifikation der neuralgischen Punkte des Systems

2. **Suche nach dem Dominating Consumer**[1] **(siehe Abschnitt 4.3)**
3. **Sammeln von Detaildaten (siehe Abschnitt 4.4)**
4. Lösen des Problems

Die nächsten Abschnitte beziehen sich insbesondere auf die Suche nach dem Dominating Consumer und das Sammeln von Detaildaten im Kontext des Garbage Collection Tunings.

4.3 Suche nach dem Dominating Consumer

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

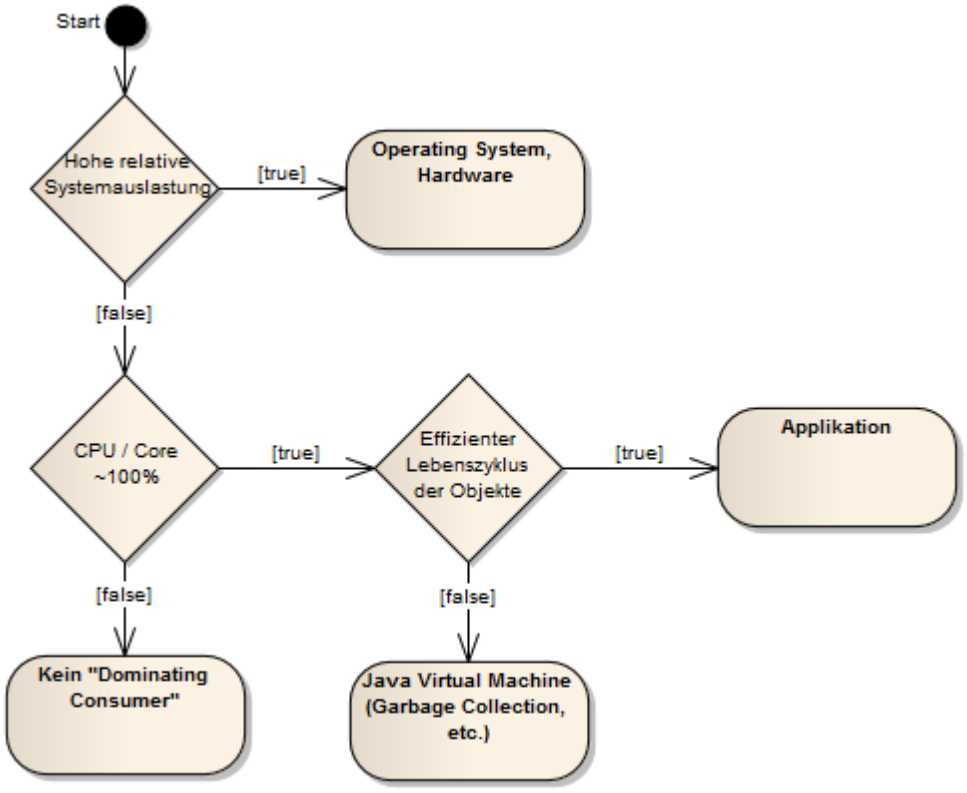

Abbildung 4.1: Suche nach dem Dominating Consumer

[1] Kirk Pepperdine bezeichnet die Aktivität, durch welche der Prozessor stark ausgelastet wird, als Dominating Consumer.

4.3.1 Hohe relative Systemlast

Die erste Frage bei der Suche nach dem Dominating Consumer ist, ob die hohe Prozessorlast durch die Applikation oder das System verursacht wird. Im zweiten Fall spricht man von einer hohen relativen Systemlast. Zur Analyse dieses Messwertes kann der Task Manager unter Windows oder vmstat unter Linux/Unix verwendet werden.

Hohe Systemauslastung kann unterschiedliche Gründe haben:

- **Exzessive Kontextwechsel:** Mehrere Prozesse können sich den gleichen Prozessor (Prozessor-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 exzessive Kontextwechsel können beispielsweise nicht adäquat gewählte Locks sein. Wenn ein Thread aufgrund der Synchronisation angehalten werden muss.[2]
- **Hohe Interaktion mit der Festplatte oder dem Netzwerk**

4.3.2 Hohe Prozessor- respektive Core-Last

Sofern die im Abschnitt 4.3.1 beschriebenen Punkte nicht zutreffen, stellt sich die Frage, ob die Auslastung des Prozessors überhaupt gross ist. Ist das nicht der Fall, gibt es womöglich keinen Dominating Consumer. Es muss dann herausgefunden werden, was die Threads daran hindert, Prozessor-Ressourcen zu verwenden. Es gibt dafür laut [10] unterschiedliche Gründe:

- **Dead Locks:** Threads schliessen sich gegenseitig aus. Das würde man in einem Thread Dump 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[3] oder Profiling wäre beispielsweise erkennbar, dass Threads oft in *read*-Operationen stecken.
- **Zu kleine Connection- und Thread-Pools**
- **Aurufe auf langsame externe Systeme**

[2] *Vmstat* zeigt auf: das System ist nicht im Leerlauf, aber die Kernelzeiten dominieren die Zeit des Benutzers.

[3] Bei einem Sampling werden in kurzen Intervallen (rund hundert mal pro Sekunde) Schnappschüsse der Threads gemacht. Es kann damit ermittelt werden, in welchen Methoden die Applikation am meisten Zeit verbringt.

4.3.3 Effizienter Objekt-Lebenszyklus

Sofern die Prozessor-Auslastung, trotz kleiner relativer Systemlast durch die Applikation, hoch ist, muss unter anderem[4] 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. Wie viele Garbage Collection Zyklen sie überlebt haben.
- **Analyse der Garbage Collection Metriken:** Wann und wie oft werden Garbage Collections durchgeführt, wie lange haben sie gedauert, wie viel Speicher haben sie befreit.

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

4.4 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 und das Tuning im Bereich der Garbage Collection. In der Regel will man mit dem Tuning eines der drei unten stehenden Zielen erreichen[7]:

- Verbesserung des Durchsatzes (siehe Abschnitt 4.4.1)
- Geringere Pausenzeiten (siehe Abschnitt 4.4.2)
- Geringerer Speicherverbrauch (siehe Abschnitt 4.4.3)

Diese drei Ziele bilden eine Dreiecksbeziehung, zum Beispiel führt eine aggressive Optimierung des Durchsatzes in der Regel zu längeren Stop-the-World[5] Pausen. Tuning bedeutet nun, zwei der Ziele konstant zu halten, während das dritte durch anpassen der Konfiguration verbessert wird. Sofern die Optimierung nicht ausreicht, muss einer der beiden anderen Parameter aufgegeben werden, um das Ziel zu erreichen.

4.4.1 Durchsatz

Worum geht es beim Durchsatz-Tuning? 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:

[4] Der Just-in-Time Compiler könnte dafür auch verantwortlich sein.

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

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

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]. 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 oder das Zuschalten von weiteren Prozessoren ist normalerweise der kostengünstigere Weg.

4.4.2 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. Diese können unterschiedlich sein:

- **Erreichen eines Schwellenwerts der Young- oder Old-Collection:** Dadurch wird eine Young- oder Old-Collection ausgelöst. Dies kann beispielsweise durch die Vergrösserung des jeweiligen Bereichs oder des gesamten Heaps herausgezögert werden.
- **Heap hat maximale Grösse erreicht:** Bevor ein OutOfMemory-Fehler den Prozess zum Stoppen bringt, wird die Applikation angehalten und eine Garbage Collection durchgeführt. Dies mit der Hoffnung, dass die Applikation anschliessend wieder weiterlaufen kann. Die Anpassung der Heap-Grösse oder das Beheben von Memory-Leaks kann diese Symptome beheben, herauszögern.
- **Allokation von Objekten ist nicht möglich:** Entweder ist in der Young-Generation kein Platz mehr vorhanden oder der Heap ist dermassen fragmentiert, dass keine freien Speicherlücken mehr gefunden werden. Das Problem kann durch vergrössern der Young-Generation oder der Wahl eines geeigneteren Garbage Collectors beseitigt werden.
- **Von der Applikation initiiert (durch System.gc()):** Benutzer versuchen teilweise, die Kontrolle über die Garbage Collection selbst zu führen. Man kann konfigurieren, dass diese durch die Applikation initiierten Aufrufe ignoriert werden.

4.4.3 Speicherverbrauch

Bei diesem Tuning geht es darum, dass der Garbage Collector mit möglichst wenig Arbeitsspeicher auskommt. 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 maximal adressierbaren Bereichs von 4GB wird noch für das Betriebssystem und andere Programme verwendet). Die Suche nach Memory-Leaks hat in der Regel nichts mit Garbage Collection Tuning zu tun.

Kapitel 5

Garbage Collection

Einige Grundkenntnisse über die Garbage Collection sind zu deren Optimierung notwendig. Dieser Abschnitt versucht einiges an Basiswissen in diesem Bereich zu vermitteln. Detailinformationen über die Garbage Collection der HotSpot VM findet man beispielsweise unter [7] oder der JRockit VM unter [5].

5.1 Einführung

Schon seit den ersten Programmiersprachen ist das Aufräumen von nicht mehr verwendeten Daten und Objekten ein wichtiges Thema. Im Unterschied zu den ersten Sprachen, bei denen das Memory Management in der Verantwortung des Entwicklers war (explizit), findet das Recycling von Memory bei Sprachen neuerer Generationen automatisch statt und macht Operatoren wie "free" unwichtig. Bei Formen dieser automatischen Speicherverwaltung spricht man von Garbage Collection. 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 an die Charakteristik der Anwendung anzupassen. Probleme die nur beim expliziten Memory Management auftreten sind Dangling References[1] (Dangling Pointers) und Space Leaks[2]. Memory Leaks sind auch bei automatischer Speicherverwaltung noch möglich, nämlich dann wenn Memory noch referenziert ist, obwohl es nicht mehr gebraucht wird. Der folgende Abschnitt beschreibt die Grundlagen der Java Garbage Collection. Das nächste Kapitel zeigt dann die Eigenheiten im Bereich der JRockit virtual Machine.

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

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

5.2 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 werden. Laut [5, S. 77] gibt es zwei unterschiedliche Strategien: Referenz-Zählung (Reference Counting) und Tracing-Techniken (Tracing Techniques).

5.2.1 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 und in nebenläufigen Anwendungen praktisch nicht möglich, die Anzahl Referenzen immer auf dem aktuellsten Stand zu halten.

5.2.2 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 Garbage Collection startet bei einer Menge von Objekten[3] die Traversierung über den gesamten Heap.

5.3 Bedingungen an Garbage Collection

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

- **Sicherheit:** Garbage Collectors dürfen nur Speicher von Objekten freigeben, welche effektiv nicht mehr gebraucht werden.
- **Umfassend:** Garbage Collectors müssen den Speicher von nicht mehr gebrauchten Objekten nach wenigen Garbage Collection Zyklen freigegeben haben. Ansonsten kommt es zu Out-of-Memory-Fehlern.

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

- **Effizienz:** Die Anwendung soll vom laufenden Garbage Collector möglichst nicht beeinträchtigt sein:

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

  - keine langen Pausen[4]
  - einen durch den Garbage Collector möglichst geringen Ressourcenverbrauch

- **Geringe bis keine Fragmentierung:** Bei starker Fragmentierung ist die Allokation von Speicher nicht 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.

5.4 Eingliederung von Garbage Collection Algorithmen

Je nach Applikation können unterschiedliche Algorithmen verwendet werden. Für die Selektion eines Algorithmus gibt es unterschiedliche Kriterien[8, S. 5]:

5.4.1 Serielle versus Parallele Collection

Algorithmen werden in serielle (mehrere GC Threads) und parallele Algorithmen eingeteilt. Auf einem Multi-Core System kann die Garbage Collection parallelisiert werden. Dies bringt zwar einen Overhead mit sich, wirkt sich aber ab gut zwei Prozessoren in verkürzten Garbage Collection Pausen aus.

5.4.2 Konkurrierend versus Stop-the-World

Einige Algorithmen bedingen, dass die Applikation während ihrer Arbeit gestoppt wird (Stop-the-World Pause), dies weil der Algorithmus nur auf einem, für andere Threads blockierten Heap, arbeiten kann. Diese Pausen sind beispielsweise für Webapplikationen sehr unerwünscht (Stottern der Applikation), können aber für Backendprozesse durchaus toleriert werden.

5.4.3 Kompaktierend, Kopierend

Die Algorithmen haben unterschiedliche Strategien, wie sie mit fragmentiertem Speicher umgehen. 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 reduziert werden.

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

5.5 Grundlage der Algorithmen

Der Prozess der Garbage Collection beginnt bei allen Algorithmen mit der Marking-Phase. Für jedes Objekt des Root Sets[5] werden rekursiv die transitiv abhängigen Objekte auf dem Heap bestimmt. Alle nicht im Resultat enthaltenen Objekte sind tot und können deshalb bereinigt werden.

5.5.1 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 Weiterentwicklungen des Algorithmus.

5.5.2 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. Die Speicherallokation findet nur in einem Space statt. Bei der Collection werden nicht überlebende Objekte gelöscht, die überlebenden anschliessend aneinandergereiht in den "to space" kopiert. Danach werden die Verantwortlichkeiten getauscht.

5.5.3 Mark & Compact Algorithmus

Für die Old Generation kann es Sinn machen, anstelle eines Mark & Copy einen Mark & Compact Algorithmus zu verwenden. Die Old Generation ist gross und die Sterberate der Objekte klein. Das Aufteilen des Speichers, bei dem dann die hälfte nicht aktiv ist, macht wenig Sinn (siehe Generationelle Garbage Collection). Nach der Markierungs-Phase und der Bereinigung der "toten" Objekte wird der Speicher dann aber kompaktiert. Der Algorithmus hat einen erhöhten Ressourcenbedarf, es wird aber im Gegensatz zum Mark & Copy kein Speicher verschwendet.

5.5.4 Tri-Coloring Mark and Sweep

Eine Variation des Mark & Sweep 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,

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

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.

5.6 Generationelle Garbage Collection

Innerhalb einer Anwendung gibt es typischerweise viele kurz lebende, einige mittellang lebende und wenig lang lebende Objekte. Aus diesem Grund wird oft mit Generationen gearbeitet und die Garbage Collection Strategie an die Lebensdauer der Objekte angepasst. Die kurz lebenden Objekte werden möglichst rasch weggeräumt, bei den anderen wird dies nicht so oft gemacht. Der Speicher wird deshalb in unterschiedliche Bereiche aufgeteilt.

5.6.1 HotSpot virtual Machine

Bei der HotSpot virtual Machine wird der Heap folgendermassen in Generationen aufgeteilt:

- **Eden-Bereich:** In Eden werden die Objekte initial erzeugt. Nach einer Garbage Collection wandern sie aber in den Survivor-Bereich.
- **Survivor-Bereiche:** Nachdem die Objekte über eine Garbage Collection in den Survivor-Bereich gelangt sind, werden sie da einige Male umher kopiert (von from-space in to-space).
- **Old Generation:** In der Old Generation befinden sich die Objekte, die eine gewisse Anzahl an Young Collections überlebt haben und dann durch eine hierhin Promotion verschoben wurden.

- **PermGen:** Der PermGen Bereich ist keine Generation, sondern ein Non-Heap-Bereich. Er wird von der VM für eigene Zwecke verwendet und ist eine Eigenheit der HotSpot VM. Hier werden beispielsweise Class-Objekte, deren Bytecode und JIT-Informationen[6] gespeichert.

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 in einem sogenannten Remembered Set nachgeführt.
- Es findet eine **Zuweisung der Referenz durch die Applikation** statt. Diese Problematik wird durch Write Barriers (zusätzliche Instruktionen) und der Aufteilung des Heaps in sogenannte Cards[7] gemacht. Bei der Änderung einer Referenz (die dann von der Old in die Young Generation zeigt) wird auf der Card, in welcher das Objekt liegt, das "Dirty Bit" gesetzt. Objekte innerhalb von dirty Cards werden nach der Markierungsphase nochmals überprüft.

5.6.2 JRockit virtual Machine

Bei der JRockit virtual Machine sind die Generationen folgendermassen angelegt:

- **Nursery[8]:** Die Nursery entspricht der Young Generation bei der HotSpot VM und beinhaltet die jungen Objekte. Bei JRockit ist es möglich, eine zusätzliche Keep-Area innerhalb der Nursery zu definieren. Diese Keep-Area ist der Platz der Objekte mittlerer Lebensdauer. Ein Objekt wird also zuerst von der Nursery in die Keep-Area verschoben - es unterliegt dann immer noch einer Young Collection - erst nach einer erneuten Garbage Collection gelangt es in die Old Generation.
- **Old Generation:** Die Old Generation beinhaltet wie bei der HotSpot VM die Objekte mit langer Lebensdauer.

Für weitere Details zu den Eigenheiten der JRockit Garbage Collection siehe Kapitel 6.

[6] Der Just-in-Time-Compiler optimiert im laufenden Betrieb den Bytecode von oft verwendeten Methoden.
[7] Eine Card ist typischerweise ein ungefähr 512 Byte grosser Bereich auf dem Heap.
[8] Nursery bedeutet im übertragenen Sinn Kindergarten.

Kapitel 6

Garbage Collection JRockit

Obwohl sich die Garbage Collection Algorithmen und Strategien der JRockit VM gegenüber der HotSpot VM in der Tiefe stark unterscheiden, haben sie oberflächlich betrachtet viele Gemeinsamkeiten. Dieser Abschnitt geht auf die Eigenheiten der JRockit virtual Machine ein.

6.1 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 Garbage Collection Threads optimiert. Die Garbage Collection der JRockit VM arbeitet entweder mit oder ohne Generationen. Es gibt folgende Algorithmen:

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

6.1.1 Optimierungen

Der verwendete Algorithmus wurde für die JRockit an einigen Orten verbessert. Dieser Abschnitt beschreibt die wichtigsten Optimierungen.

Verwendung von threadlokalem Speicher

Im Unterschied zum normalen Tri-Coloring Mark & Sweep Algorithmus verwendet die 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 im Voraus geladen (Prefetching) werden, was die Geschwindigkeit des Algorithmus erhöht.

Bei der Verwendung der Concurrent Algorithmen, bei diesen läuft die Applikation nebenbei mit, werden parallel dazu auch neue Objekte instanziert (referenziert). Diese werden in einem sogenannten Live-Set geführt, 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 deshalb die Objekte auf dem Heap umher kopiert, so dass danach wieder grössere freie Speicherbereiche verfügbar sind.

6.2 Nebenläufige Algorithmen

Bei den Concurrent Mark & Sweep Algorithmen auf der JRockit handelt es sich eigentlich um "mostly concurrent" Algorithmen. Das heisst, sie können nicht in allen Phasen konkurrierend zur Applikation stattfinden. Damit wenigstens ein Teil der **Markierungsphase** konkurrierend abläuft, ist sie in verschiedene Phasen unterteilt:

- 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 dem vorherigen Schritt werden nachgeführt und noch markiert.
- Final Marking (Nicht konkurrierend): Änderungen im Heap während der Precleaning Phase werden nachgeführt, markiert. Diese Phase ist nicht konkurrierend, dauert aber nur sehr kurz.

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.

6.3 Parallele Algorithmen

Bei den parallelen Algorithmen findet die Garbage Collection mit allen verfügbaren Prozessoren statt. Dazu werden aber alle Threads der Applikation gestoppt. Dies ist für Server-Applikationen nicht sinnvoll, führt aber bei Backend-Applikationen zu einem erhöhten Durchsatz.

6.3.1 Übersicht und Auswahl Algorithmen

Die nachfolgende Tabelle zeigt eine Übersicht der aktuell verfügbaren Algorithmen auf der JRockit virtual Machine. Diese können per Argument auf der Kommandozeile direkt selektiert werden. Manchmal werden sie aber aufgrund heuristischer Auswertungen im laufenden Betrieb gewechselt.

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 6.1: Auswahl des Garbage Collection Algorithmus

6.4 Garbage Collection Modi

Anstelle einer direkten Auswahl eines Algorithmus (siehe Abschnitt 6.3.1), kann auch eine grundlegende Strategie angegeben werden. Die VM entscheidet dann heuristisch, welcher spezifische Algorithmus gewählt wird. Die verfügbaren Modi entsprechen den Tuningzielen, die es für die Garbage Collection gibt (siehe Abschnitt 4.4):

- **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. Es werden parallele Algorithmen verwendet. 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 6.2: Übersicht der Garbage Collection Modi

6.5 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.

6.5.1 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 in eine Datei geschrieben. Das Format für die Kommandozeile ist wie folgt:

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

Listing 6.1: 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 6.2: Garbage Collection Log (Info)

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

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

Listing 6.3: 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 6.4: Einstellung des Log-Levels

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

6.5.2 Beispiel einer Garbage Collection Logdatei

Eine Logdatei besteht in der Regel, abhängig von der Dauer der Messung und der Aktivität der virtual Machine, aus mehreren tausend Zeilen. Die ersten paar befinden sich exemplarisch auf der nächsten Seite.

```
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 6.5: Garbage Collection Logdatei

6.5.3 Log Module

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

Modul	**Beschreibung**	**Relevanz**[1]
alloc	Informationen betreffend Speicher Allokation und "Out-of-Memory" Meldungen	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 der Applikation zwecks Garbage Collection gemacht wurden und 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 Detailinformationen im Bereich Speicherverwaltung.	**hoch**
systemgc	Zeigt Aufrufe durch die Applikation auf System.gc().	niedrig

Tabelle 6.3: Beschreibung der verschiedenen relevanten Log Modulen

[1] Relevanz für Garbage Collection Tuning

Teil III

Anforderungsanalyse

Kapitel 7

Anforderungsanalyse

Die Evaluation von Rich Client Framework und Charting-Bibliothek sowie die Konzeption der funktionalen Aspekte ist getrieben durch die Anforderungsanalyse. Es standen drei Methoden zur Auswahl: Use Cases, Requirements Engineering nach IEEE 830 und User Stories. Auf Ebene der Customer Requirements[1] wird die Methode der Use Cases eingesetzt. Sie zeichnet sich dadurch aus, dass man die Anforderungen nicht nur textuell sondern auch modellbasiert mit UML definieren kann. Für die Dokumentation der Development-Requirements, beinhaltet auch die Qualitätsanforderungen, wird der Standard IEEE 830 verwendet. Die damit definierten Anforderungen lassen sich gut in einzelne Entwicklungspakete verpacken. Auch die Struktur der Anforderungsanalyse wurde anhand dieser Definition aufgebaut.

7.1 Einleitung

7.1.1 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.

[1] Nach Standard IEEE 830[17] werden die Anforderungen in Customer-Requirements (Anforderungen aus Sicht des Kunden) und Development-Requirements (Anforderungen aus Sicht des Entwicklers) unterteilt.

7.1.2 Systemumfang

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

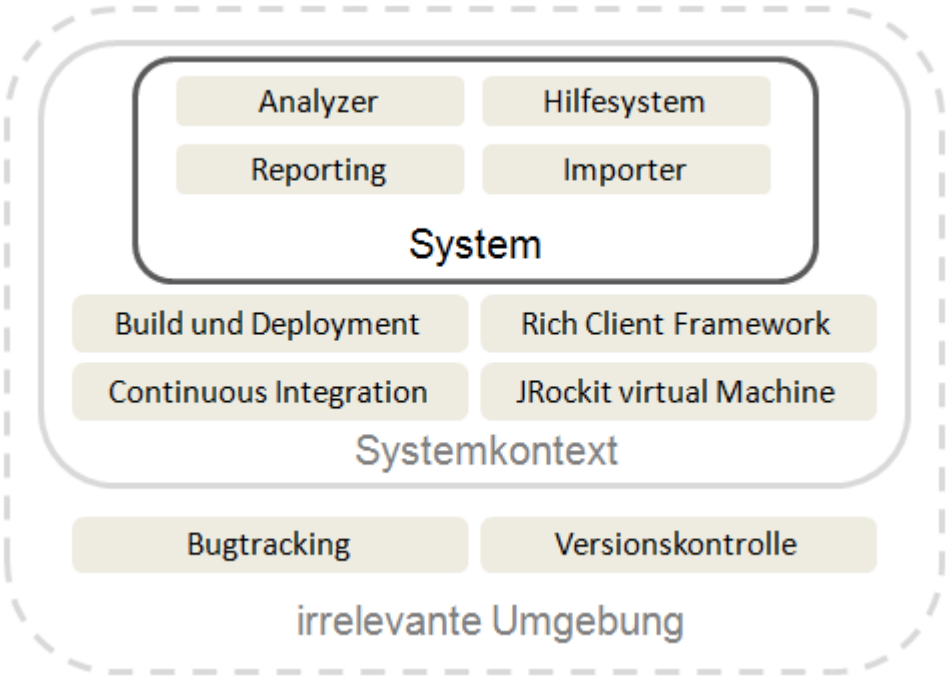

Abbildung 7.1: System und Systemkontext

System

Das System besteht aus den Teilen Importer, Analyzer, Reporting und Hilfesystem. Diese Teile sind Bestandteil der Entwicklung und können den Anforderungen entsprechend konzipiert und umgesetzt werden.

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.

- **Rich Client Framework:** Als Basis der Analysesoftware wird ein Rich Client Framework verwendet. Die Struktur und Architektur dieser Bibliothek beeinflusst den Aufbau der Software massgeblich.
- **JRockit virtual Machine:** Die Schnittstelle zur JRockit Virtual Machine findet über deren Logdateien statt. Die genauere Beschreibung befindet sich im Abschnitt 6.5.

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, Historisierung der einzelnen Artefakte. Issuetracker und Versionskontrolle arbeiten eng zusammen, so dass Issues mit der eingecheckten Version in Verbindung gebracht werden können.

7.1.3 Stakeholder

Für die Anforderungsanalyse sind im Diagramm unten nur die mit Asteriks gekennzeichneten Rollen[2] relevant. Die anderen ab dem Zeitpunkt des Releases der Software.

Abbildung 7.2: Übersicht der Stakeholder

[2] Verschiedene Rollen können von einer Person wahrgenommen werden.

7.1.4 Glossar

Das Glossar befindet sich im Anhang A.

7.1.5 Referenzen

Die Referenzen innerhalb dieses Abschnittes befinden sich im Literaturverzeichnis.

7.1.6 Übersicht

Die Anforderungsanalyse ist folgendermassen aufgebaut: Gestartet wird mit der allgemeinen Übersicht. Hier sind Architektur, Nutzer- und Zielgruppen und Randbedingungen dokumentiert. Die Customer-Requirements werden mit Use Cases definiert, die Development Requirements richten sich nach dem Standard IEEE 830.

7.2 Allgemeine Übersicht

7.2.1 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.

7.2.2 Nutzer und Zielgruppen

Performance Engineers

Software Entwickler welche die Erfahrung, das Wissen und die Fähigkeit haben, 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. Sie haben sich auch eine strukturierte Vorgehensweise angeeignet und verfügen über die Kenntnisse der wichtigsten Werkzeuge. Durch ihr breites Wissen sind sie mit der Unterstützung von Charts, Statistiken und Reports in der Lage, Ursachen 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. Durch ihre tägliche Arbeit kommen sie mit den Anforderungen an die Performance in Kontakt. Als gut ausgebildete Ingenieure sind sie aber mit Hilfe von Werkzeugen und Dokumentationen

in der Lage, Performanceproblemen innert nützlicher Frist auf den Grund zu gehen.

7.2.3 Randbedingungen

JRockit R28

Der Fokus der Analysesoftware stützt sich auf die Logdateien der JRockit Virtual Machine Release 28. Auswertungen für Garbage Collection Logdateien anderer Hersteller werden eventuell zu einem späteren Zeitpunkt implementiert.

7.3 Use Cases

Dieser Abschnitt zeigt die Use Cases für die Analysesoftware, daraus wird am Ende der Anforderungsanalyse die Liste der Anforderungen ermittelt. Zur Übersicht dient das Use Case Diagramm im nächsten Abschnitt. Es definiert die Systemgrenze, Anwender und die einzelnen Use Cases. Diese sind anschliessend nach der in [11, S. 78-79] definierten Schablone beschrieben.

7.3.1 Übersicht

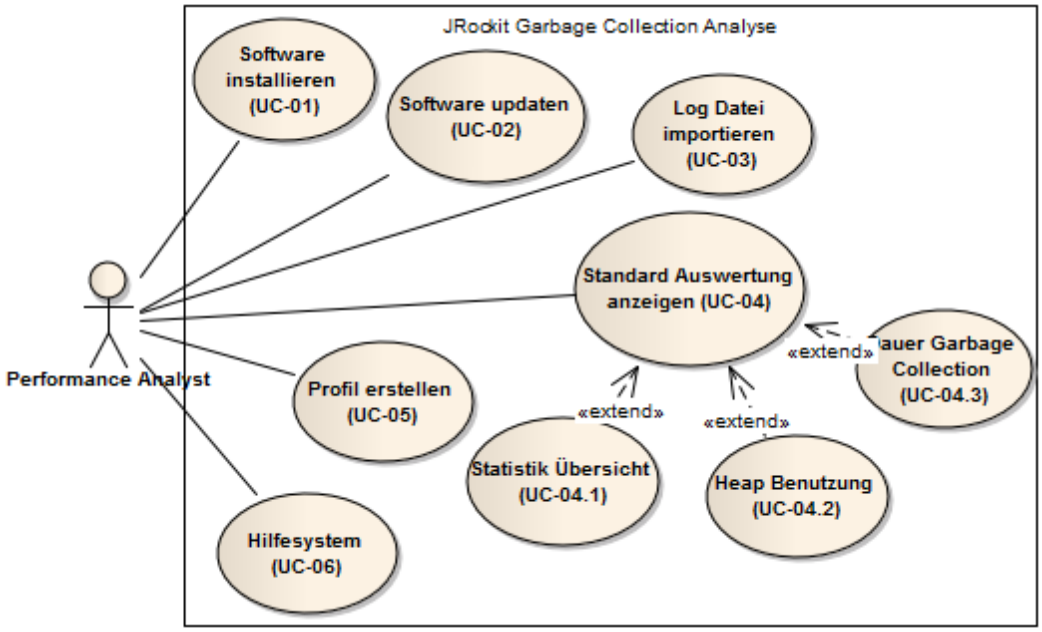

Abbildung 7.3: Systemfunktionalität als Use-Case-Diagramm

7.3.2 Beschreibung

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, Rich Client Framework
Auslösendes Ereignis	Anwender möchte eine Garbage Collection Logdatei analysieren.
Vorbedingung	Richtige Entwicklungsumgebung ist bereits ohne Analysesoftware installiert.
Nachbedingung	Es sind bei der Installation 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 installierende Softwarepakete und entscheidet sich, diese zu installieren. 4. Softwarepaket wird installiert.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Usability

Tabelle 7.1: 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 aus der Entwicklungsumgebung aktualisieren.
Akteure	Anwender, Rich Client Framework
Auslösendes Ereignis	Anwender hat die Software bereits zu einem früheren Zeitpunkt installiert. Es ist eine neue Version der Software auf der Update-Seite.
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 Auswertungen.
Hauptszenario	1. Anwender startet Entwicklungsumgebung. 2. Anwender sucht und findet Updates für die Analysesoftware. 3. Anwender selektiert eines oder mehrere dieser Softwarepakete und entscheidet sich, diese zu installieren. 4. Software wird aktualisiert.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Usability

Tabelle 7.2: 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 kann eine sich auf dem Dateisystem befindende Logdatei importieren.
Akteure	Anwender, 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 7.3: 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 kann der Benutzer eine Standardauswertung öffnen. Diese soll eine kurze Übersicht über die Garbage Collection geben und beinhaltet zusätzlich zwei Charts (Heap Benutzung, Dauer Garbage Collection).
Akteure	Anwender, Analyzer, Reporting
Auslösendes Ereignis	Der Benutzer hat eine Garbage Collection Logdatei importiert und möchte nun die Analyse starten.
Vorbedingung	Applikation hat die Logdatei fertig importiert.
Nachbedingung	-
Ergebnis	Dem Benutzer wird ein Analyse-Screen angezeigt.
Hauptszenario	1. Bevor das Analysefenster für die 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. 2. Dem Benutzer wird ein Screen mit verschiedenen Tabs angezeigt. Auf jedem Tab wird dem Benutzer eine andere Sicht auf die Daten gezeigt.

Alternativszenarien	• Benutzerdefinierte Auswertung • Format der Logdatei wird nicht unterstützt, die Datei wird nicht im Analysefenster geöffnet, dem Benutzer wird eine Fehlermeldung angezeigt.
Ausnahmeszenarien	-
Qualitäten	Korrektheit, Performance, Usability
Erweiterungen	UC-04.1, UC-04.2, UC-04.3, UC-04.4

Tabelle 7.4: 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 4.4.1)
Qualitäten	Korrektheit, Performance, Usability

Tabelle 7.5: 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 Benutzung* zeigt dem Benutzer anhand einer Grafik, zu welchem Zeitpunkt wie viel Speicher des Heaps verwendet wurde. Zusätzlich werden die Zeitpunkte inklusive entsprechendem Typ (Old- / Young-Collection) der Garbage Collection angezeigt.
Qualitäten	Korrektheit, Performance, Usability

Tabelle 7.6: 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	Die Anzeige *Dauer Garbage Collection* zeigt dem Benutzer über die Zeit, wie lange die einzelne Garbage Collection gedauert hat.
Qualitäten	Korrektheit, Performance, Usability

Tabelle 7.7: 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 dargestellt werden. Eine Serie definiert, welche Daten auf der X- respektive Y-Achse angezeigt werden sollen. hinzugefügt werden. Die Profile sind persistent und können exportiert wie auch importiert werden.
Akteure	Anwender, Analyzer, Reporting
Auslösendes Ereignis	Die Applikation hat die Datei fertig eingelesen und geparst.
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	Der Benutzer kann ein eigenes Profil erstellen. Ein Profil besteht initial aus einem Übersichtsfenster und kann um benutzerdefinierte Charts, jeweils auf einem eigenen Tab dargestellt, erweitert werden. Sollte die Entwicklungsumgebung mit der Analysesoftware geschlossen werden, bleiben die erstellten Profile erhalten.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Korrektheit

Tabelle 7.8: 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	Der Benutzer kann auf eine indexbasierte[3] und eine kontextsensitive[4] Hilfe zugreifen.
Akteure	Anwender, Hilfesystem
Auslösendes Ereignis	Anwender hat Plugin installiert, weiss nicht wie eine Analyse gestartet wird.
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 zusätzliche Informationen.
Alternativszenarien	-
Ausnahmeszenarien	-
Qualitäten	Internationalisierung, Usability

Tabelle 7.9: Use-Case: Hilfesystem

7.4 Funktionale Anforderungen

Die folgende Liste der funktionalen Anforderungen an die Analysesoftware (stand 6. Dezember 2011) wurde hinsichtlich Korrektheit, Machbarkeit und Notwendigkeit geprüft und entsprechend priorisiert:

Identifik.	Vers.	Titel	Beschreibung	U.C.[5]	Quelle	Abnahmekriterium	Prio.
FRQ-01	1.0	Installation	Analysesoftware kann als Erweiterung in der Entwicklungsumgebung[6] installiert werden.	UC-01	Raffael Schmid (Project Owner)	Entwickler mit Kenntnissen der Entwicklungsumgebung benötigt für die Installation in eine bestehende Entwicklungsumgebung weniger als 5 Minuten.	gross
FRQ-02	1.0	Updaten	Analysesoftware kann aktualisiert werden.	UC-02	Raffael Schmid (Project Owner)	Entwickler mit Kenntnissen der Entwicklungsumgebung benötigt für den Update weniger als 3 Minuten.	mittel
FRQ-03	1.0	Garbage Collection Logdatei importieren	Logdatei kann in die Ansicht *Logdateien* importiert werden.	UC-03	Raffael Schmid (Project Owner)	-	gross

[5] Use Case

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

FRQ-04	1.0	Zustand Ansicht Logdateien speichern	Die sich in der Ansicht Logdateien befindenden Einträge werden gespeichert.	UC-03.1	Raffael Schmid (Project Owner)	-	gross
FRQ-05	1.0	Garbage Collection Logdatei einlesen	Geöffnete Logdatei kann ins Memory geladen werden.	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 eingelesene JRockit Garbage Collection Logdatei wird geparst. Die Daten werden syntaktisch und semantisch analysiert und als einen Objektgraphen gespeichert.	UC-04	Raffael Schmid (Project Owner)	Das Parsen einer Logdatei mit 100000 Zeilen dauert nicht länger als 8 Sekunden.	gross
FRQ-07	1.0	Standard-auswertung anzeigen	Der Benutzer kann eine standardisierte, nicht veränderbare Anzeige öffnen.	UC-04	Raffael Schmid (Project Owner)	-	gross
FRQ-08	1.0	Anzeige Statistik Übersicht	Der erste Screen (Tab) auf dem Analysefenster zeigt 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 definieren. 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 dem 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 Exportdatei gespeichert werden.	UC-05	Adrian Hummel (Performance Analyst)	-	klein

FRQ-15	1.0	Charts importieren	Die in einer Exportdatei gespeicherten 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 7.10: Funktionale Anforderungen

7.5 Qualitätsanforderungen

7.5.1 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 werden.	Raffael Schmid (Project Owner)	Erweiterung um ein weiteres Logformat soll den Aufwand von 5 Personentage 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	Lange Operationen werden für den Benutzer mittels einer Progress-Anzeige visualisiert.	Raffael Schmid (Project Owner)	Dem Benutzer wird ein Monitor bereitgestellt.	mittel
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
QRQ-S-07	1.0	Performance	Schnelles Einlesen von grossen Dateien.	Raffael Schmid (Project Owner)	Der Import einer Log-Datei von 100000 Zeilen dauert kürzer als 10 Sekunden.	mittel

7.5.2 Basisframework

Die an das Framework existierenden Anforderungen werden hier getrennt aufgelistet. Sie werden besonders für die Evaluation benötigt.

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[7] 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 7.12: Qualitätsanforderungen Basisframework

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

Teil IV

Konzept

Das Konzept ist in fünf Kapitel unterteilt. Im ersten befindet sich die Auswahl des Rich Client Frameworks und der Bibliothek zur Anzeige der Charts. Im zweiten die Architekturthemen der Software. Der dritte und vierte Teil konzipiert die funktionalen und Qualitätsanforderungen. Im fünften Teil befindet sich das Konzept im Bereich Infrastruktur und Change Management (Issue Tracking, Versionskontrolle, Continuous Integration, etc.).

Kapitel 8

Auswahl Frameworks und Komponenten

Im Vorfeld des Software-Konzepts wird die Evaluation der verschiedenen Rich Client Frameworks durchgeführt. Dieser Abschnitt beinhaltet die Ziele *Analyse über Stärken und Schwächen der bestehenden Java Rich Client Technologien* und *dokumentierten Auswahlkriterien und Entscheidungsgrundlagen*. Der Abschnitt ist in drei Teile gegliedert. Der erste Teil beschreibt das Bewertungsschema, der zweite und dritte die Evaluation von Rich Client Framework und Bibliothek zur Anzeige der Diagramme.

8.1 Bewertung

Die Evaluation und Bewertung des Rich Client Frameworks und der Charting Bibliothek basiert auf dem nachfolgend beschriebenen Berechnungsmodell.

Auf der Basis der Anforderungen werden die Bewertungskriterien definiert. Die Gewichtung der einzelnen Kriterien leitet sich aus der Priorität ab:

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

Tabelle 8.1: Schema Umwandlung Priorität in Gewicht

Die Bewertung der einzelnen Kriterien basiert auf dem aus der Evaluation erhaltenen Erfahrungswert und wird folgendermassen umgewandelt:

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

Tabelle 8.2: Schema Vergabe der Punkte

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

Punktezahl = Gewichtung x Bewertung

8.2 Evaluation Rich Client Framework

Grundsätzlich kommt für die Entwicklung der Analysesoftware auch die Java GUI-Bibliothek (Bibliothek zur Programmierung von Grafischen Benutzer Oberflächen) Swing in Frage. Viele der Anforderung (Deployment, Installation, Update, Modularisierung, Internationalisierung, etc.) werden von Rich Client Frameworks aber bereitgestellt und stehen bereits mit geringem Aufwand zur Verfügung. In die Evaluation werden deshalb nur Rich Client Frameworks mit einbezogen. Von denen kommen aufgrund der funktionalen Anforderung QRQ-F-05 (Offline Betriebsmodus) und QRQ-F-06 (Installation als Erweiterung) Eclipse RCP[1] 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 seltener. In der Folge werden die drei Frameworks kurz beschrieben und dann anhand der Anforderungen verglichen.

8.2.1 Eclipse RCP

Bis zur Version 2.1 war Eclipse bekannt als eine Open Source Entwicklungsumgebung für Programmierer. Der Vorgänger hiess *Visual Age for 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. Die Plattform kann nun auch als Framework zur Entwicklung von Desktop Applikationen oder Plugins verwendet werden - nicht nur als Entwicklungsumgebung. Nachfolgend einige Details in Bezug auf die Anforderungen:

- **Verbreitung:** Eclipse RCP ist relativ stark verbreitet, die Suche auf Amazon nach "Eclipse RCP“ (Kategorie: Bücher) liefert 53 Resultate (stand 27. November 2011).
- **Plattformunabhängigkeit:** Eclipse ist für 14 verschiedene Systeme und Architekturen bereitgestellt und gilt somit als plattformunabhängig[15].
- **Lokalisation:** Wird von Eclipse durch die Erweiterung der Java-Lokalisation unterstützt.

[1] RCP steht für Rich Client Plattform

- **Modularisierung:** Die Modularisierung in Eclipse Anwendungen wird auf der Basis von Plugins gemacht. Ein Plugin ist eine Menge von Klassen mit einer definierten Schnittstelle (welche Klassen, Packete werden importiert, exportiert).
- **Installation als Erweiterung:** Eclipse bietet an, Plugins und Features in die bestehende Entwicklungsumgebung zu installieren.

8.2.2 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. Auch hier einige Aspekte in Bezug auf die Anforderungen:

- **Verbreitung:** Netbeans RCP ist nicht sehr verbreitet, die Suche auf Amazon nach "Netbeans RCP"(Kategorie: Bücher) liefert 7 Resultate (stand 27. November 2011).
- **Plattformunabhängigkeit:** Netbeans RCP basiert vollumfänglich auf Java und ist deshalb auf allen Plattformen, für die eine Java Virtual Machine existiert, verfügbar. Es gilt deshalb als plattformunabhängig (siehe [16]).
- **Lokalisation:** Wird von Netbeans durch die Erweiterung der Java-Lokalisation unterstützt.
- **Modularisierung:** Die Modularisierung in Netbeans kann mit Modulen (äquivalent Plugin in Eclipse Applikationen) gemacht werden.
- **Installation als Erweiterung:** Netbeans bietet es ebenfalls an, Module in die bestehende Entwicklungsumgebung zu installieren.

8.2.3 Auswertung

Die erste und zweite Spalte zeigen die Anforderungen aus Abschnitt 7.5.2. Die dritte Spalte enthält das aus der Priorität abgeleitete Gewicht (siehe Abschnitt 8.1). Die weiteren Spalten zeigen pro Kandidat die Bewertung zusammen mit dem errechneten Produkt (Gewicht x Bewertung).

Anforderung[2]	**Nummer**	**Gewicht.**[3]	**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)

[2] siehe Abschnitt 7.5.2
[3] Gemäss Priorität der Anforderung

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)
Offline Betriebsmodus	(QRQ-F-05)	4	5 (20)	5 (20)	5 (20)
Installation als Erweiterung	QRQ-F-06	4	5 (20)	5 (20)	5 (20)
Total			**105**	**89**	**93**

Tabelle 8.3: Auswertung Rich Client Frameworks

Ausschlaggebend für die Wahl der Rich Client Plattform ist die Verbreitung. Trotz der Unterschiede in Architektur und den verwendeten Technologien sind sie hinsichtlich Funktionsumfang praktisch identisch. Laut [12] können häufig auftretende Use Cases an eine Rich Client Applikation sowohl mit der Netbeans als auch mit der Eclipse Plattform umgesetzt werden.

8.3 Evaluation Bibliothek Charting

Für die grafische Darstellung der Daten als Diagramm wird eine Bibliothek verwendet werden. Im Bereich der nicht-kommerziellen Produkte wird dafür sehr oft Eclipse BIRT (Business Intelligence and Reporting Tools) und JFreeChart eingesetzt.

8.3.1 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 Vorlagen die Diagramme und Grafiken.

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

8.3.2 JFreeChart

JFreeChart ist ein Open-Source-Framework mit welchem eine Vielzahl verschiedener Diagramme erzeugt werden kann. 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.

8.3.3 Auswertung

Anforderung[4]	**Nummer**	**Gewicht.**[5]	**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 8.4: Auswertung Charting-Bibliothek

8.3.4 Entscheid

JFreeChart eignet sich insbesondere bei Desktop-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.

[4] Es wurden nur die für die Evaluation der Charting Bibliothek relevanten Anforderungen aus Abschnitt 7.5.1 und 7.5.2 übernommen.

[5] Gemäss Priorität der Anforderung

Kapitel 9

Architektur

Als Basis für die Konzeption der funktionalen Anforderungen wird in diesem Abschnitt die Architektur der Software definiert. Nebenbei befinden sich darin auch allgemeine Informationen wie das Thema der Lizenzierung.

9.1 Allgemein

9.1.1 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.

9.1.2 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.

9.2 Übersicht

Aufgrund der Anforderung QRQ-S-01 (Erweiterbarkeit) muss die Analysesoftware auch hinsichtlich anderer Logformate erweiterbar sein. Die Applikation

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

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:

- Die Applikation wird in zwei Komponenten aufgeteilt:
  - Basissoftware
  - Erweiterung JRockit

  Die Basissoftware kann unabhängig von den Erweiterungen installiert werden, für die Auswertung wird die entsprechende Erweiterung allerdings benötigt. Eine Erweiterung kann ohne Basissoftware nicht gebraucht werden.
- Die Architektur der Applikation muss es zulassen, dass zu einem späteren Zeitpunkt auch Garbage Collection Logs von anderen virtuellen Maschinen analysiert werden.
- Die Basissoftware stellt Extension-Points[3] bereit, über welche sich die Erweiterungen registrieren.

9.2.1 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. Die Aufgaben werden 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 bereitstellen 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

[3] Ein Extension-Point ist ein Mechanismus, mit dem zwei Eclipse-Plugins miteinander kommunizieren können.

Logs der JRockit geschriebene Erweiterung. Sie ist für das Parsen der Logdateien sowie die Aufbereitung und das Bereitstellung der Daten zuständig. Beinhaltet aber keine Basisfunktionalität.

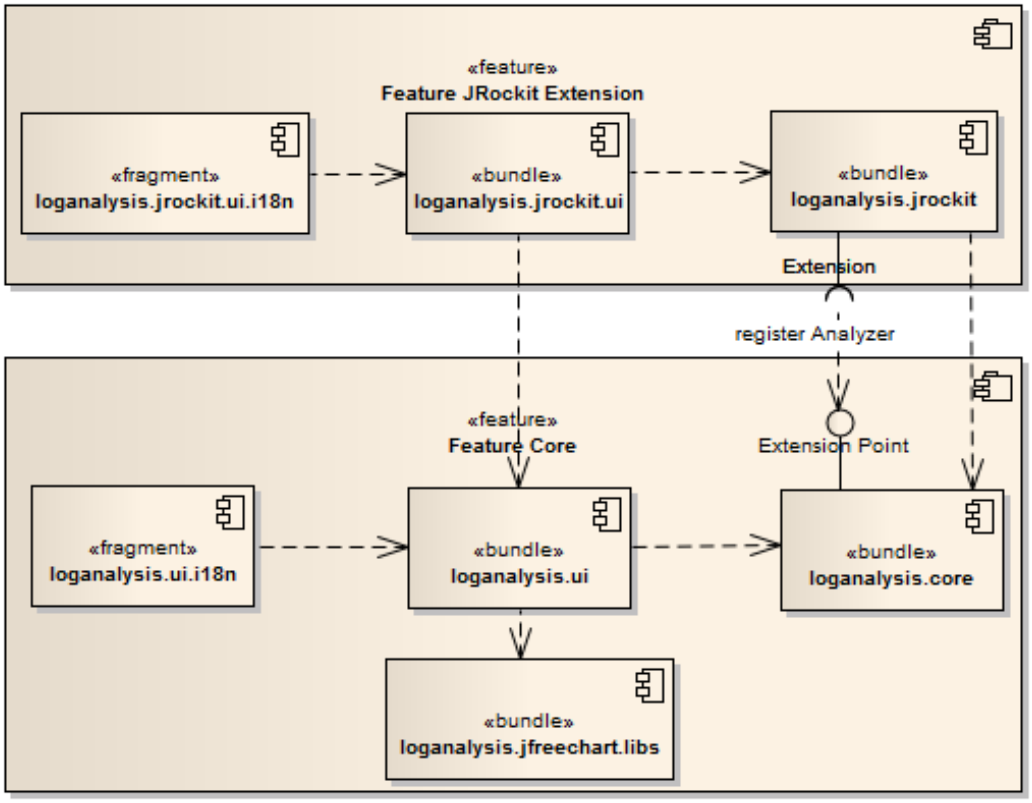

Abbildung 9.1: Architektur: Komponentendiagramm

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

[4] JFace und SWT werden in Eclipse als Library für die Präsentationsschicht verwendet.

Regel beinhalten sie Konfigurationsdateien oder Test-Klassen:

- Test-Projekte[5]
  - 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)

[5] Der Test-Code befindet sich in eigenen Projekten (siehe Abschnit 11.2).

[6] Features sind in sich lauffähige Softwarekomponenten mit definiertem Umfang und Abhängigkeiten auf Plugins. Zur Defininition wird ein eigenes Eclipse-Projekt erstellt.

[7] Eingebundene externe Bibliotheken werden als Plugins gepackt, damit deren Klassen durch den Modul-Loader geladen werden können.

[8] Definiert gegen welche Plattform die Anwendung entwickelt wird.

[9] Definiert bereitgestellte Features.

Kapitel 10

Funktionalität

Dieser Abschnitt orientiert sich an den funktionalen Anforderungen und konzipiert diese jeweils einzeln.

10.1 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. Update-Seiten im Eclipse-Umfeld bestehen aus Features (Eclipse Feature-Projekte). Features bestehen aus unterschiedlichen Plugins (Eclipse Plugin-Projekte). 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 9.2.1).
- **JRockit Erweiterung:** Umfasst alle Plugins zur JRockit Erweiterung und hat zugleich die Abhängigkeit auf das Basissoftware-Feature.

10.2 Update (FRQ-02)

Der Update eines Features auf der Update-Seite ist erkennbar an der neuen 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 danach notwendig.

10.3 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* (siehe Abschnitt 10.5.1) geladen und in der Ansicht *Logdateien* angezeigt, der Inhalt ist zu diesem Zeitpunkt allerdings noch nicht im Arbeitsspeicher.

10.4 Importierte Dateien speichern (FRQ-04)

Der im Abschnitt B.1 beschriebene Mechanismus wird verwendet, damit nach einem Neustart der Entwicklungsumgebung die importierten Logdateien nicht weg sind.

10.5 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 in Form des im nächsten Abschnitt definierten Domänenmodells geladen.

[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.

10.5.1 Domänenmodell

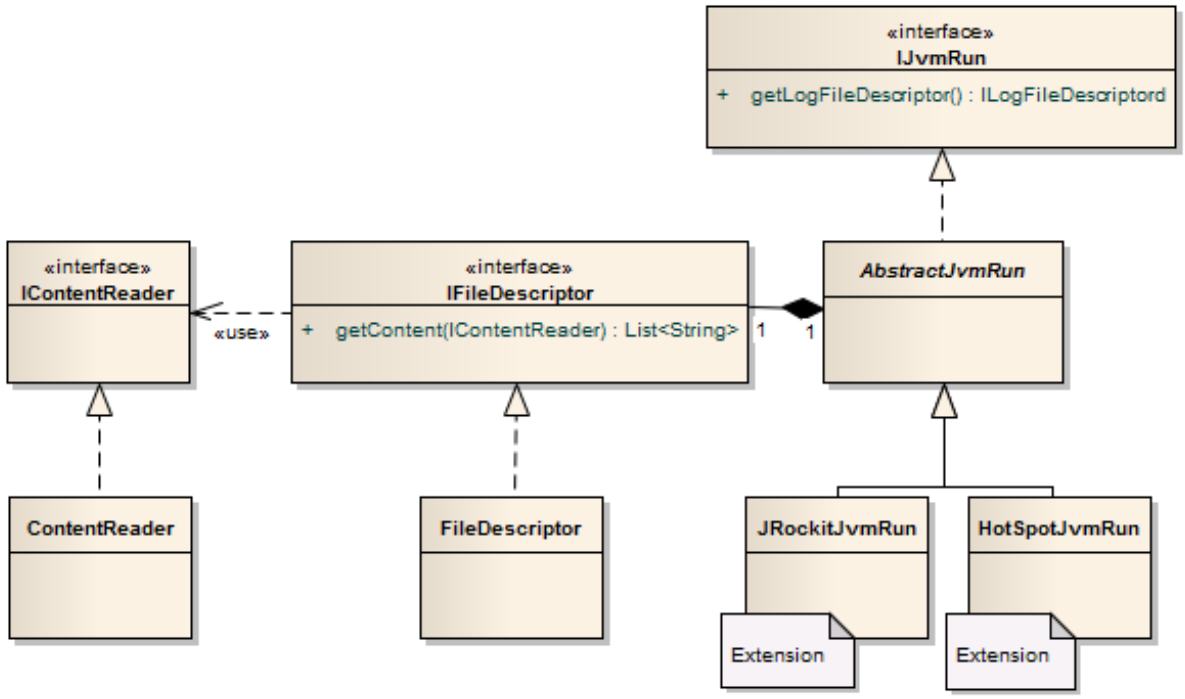

Abbildung 10.1: Domänenmodell: Eingelesene Datei

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 des Modells hinter einer Garbage Collection heisst *AbstractJvmRun* und wird erst von der jeweiligen Erweiterung realisiert.

10.6 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 6.5.3). Für die Analyse sind nicht alle Einträge relevant, es werden nur die wichtigsten angeschaut. Die einzelnen Parser für die Garbage Collection Logdatei werden auf der Basis von regulären Ausdrücken selber implementiert. Auf die Verwendung eines Parser-Generators wird aus folgenden Gründen verzichtet:

- Bei der Verwendung von Parser-Generatoren muss in der Regel eine Bibliothek zusammen mit der Software ausgeliefert werden. Aufgrund der

Anforderung QRQ-S-06 (siehe Abschnitt 7.5.1) soll die Grösse der Software möglichst klein sein. Die Implementation eines Parser auf der Basis von Regulären Ausdrücken kann ohne zusätzliche Bibliothek gemacht werden.

- Parser-Generatoren sind proprietär und die Einarbeitungszeit ist gross.

10.6.1 Parser

Der Parser für die Logdateien der JRockit VM ist nach dem Chain-of-Responsibility Pattern[14] aufgebaut. Dieses Pattern ermöglicht die schnelle Erweiterung (Anforderung QRQ-S-01) der Software um weitere Parser. Pro Format einer Log-Zeile kann ein Parser in die Reihe 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 anderer Log-Module denkbar (Einträge des Nursery- oder Allocation-Modules, etc.). Ablauf und Aufbau des Parsers sind in den folgenden beiden Grafiken ersichtlich:

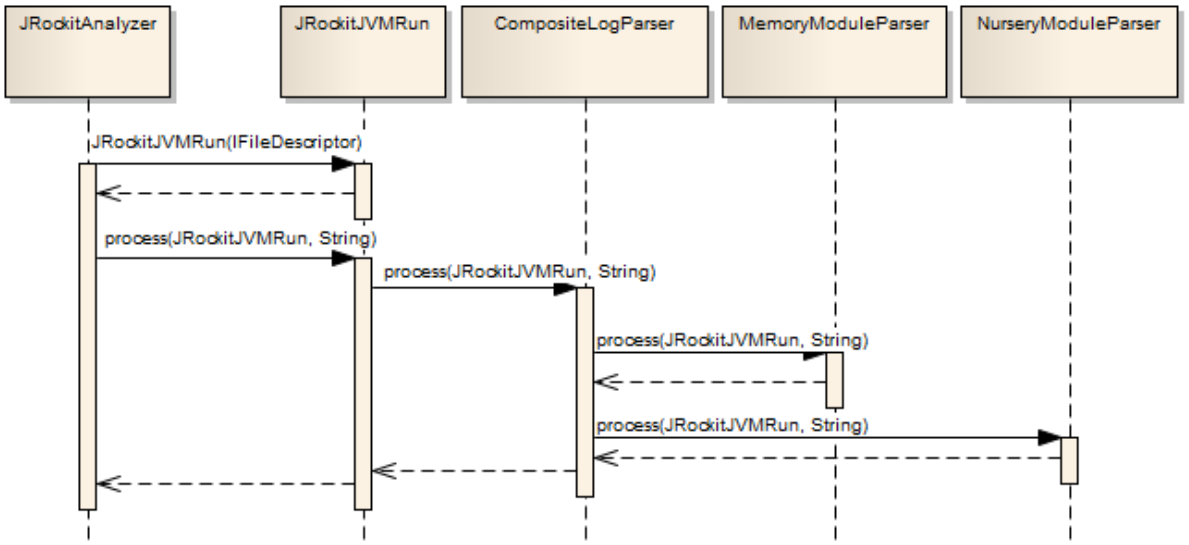

Abbildung 10.2: Sequenzdiagramm Aufruf Parser-Hierarchie

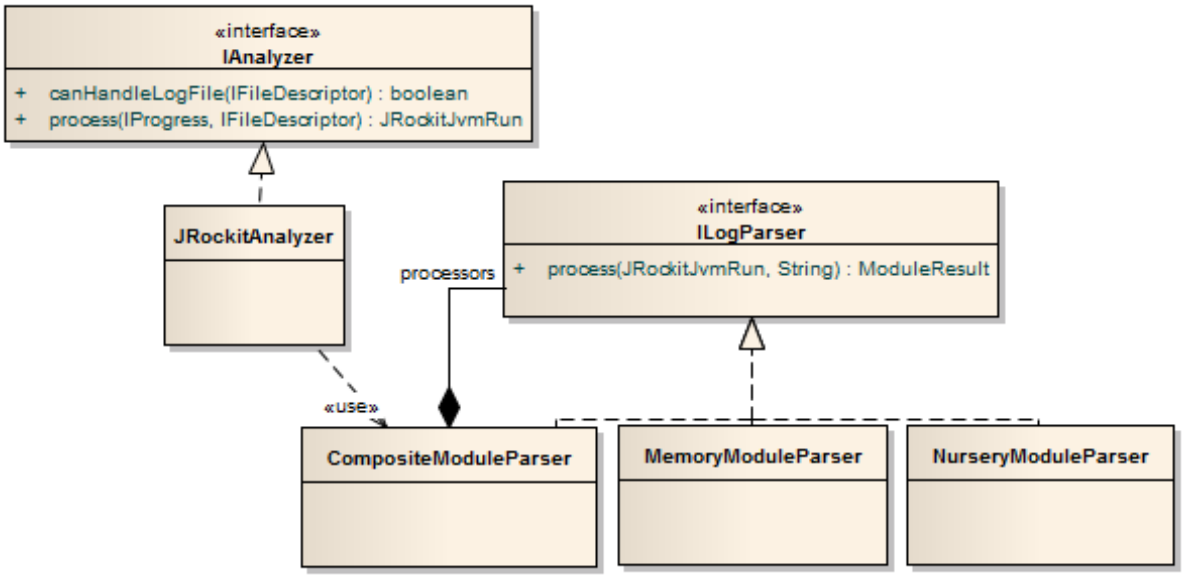

Abbildung 10.3: Klassendiagramm Parser

MemoryModuleParser

Das Parsen innerhalb des *MemoryModuleParser* (Parser der die Log-Ausgaben des Memory-Log-Modules prozessiert) 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 reguläre Ausdrücke implementiert. Die Werte werden mittels Gruppierungs-Funktion[2] extrahiert.
- **Syntactic Analyzer:** Der syntaktische Analyser verarbeitet die vom Lexer extrahierten Tokens. Er führt eine semantische Analyse durch und speichert die Werte in strukturierter Form als Objektgraphen.

10.6.2 Auswertung Logdatei

Bereits in Abschnitt 6.5.2 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 gemacht werden können, ohne dabei aber auf die Ausgaben des Debug-Log-Levels[3] einzugehen.

[2] Bei regulären Ausdrücken lassen sich einzelne Werte aus einem String mittels Gruppierungsfunktion extrahieren.

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

Garbage Collection Algorithmus

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

Listing 10.1: Logdatei: Ausgabe initialer Garbage Collection Algorithmus

Die eigentlich erste und wichtigste Entscheidung beim Garbage Collection Tuning ist die Wahl der Strategie. Sie wird im Header der Logdatei angezeigt und entspricht entweder der Standardeinstellung oder Konfiguration durch den Anwender. Ausgewertet aus dem Eintrag werden zwei Dinge: für was die Garbage Collection optimiert ist (Durchsatz oder Pausenzeiten) und welcher Algorithmus eingestellt ist.

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

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

Listing 10.2: Logdatei: Initiale und maximale Heap-Kapazität und Grösse des Old- und Young-Generation

Ebenfalls im Header der Logdatei befindet sich die Angabe über die initiale und maximale Heap-Kapazität und die Grösse der Young-Generation (Nursery). Die Grösse der Old-Generation (Tenured Space) errechnet sich aus der Differenz zwischen Young-Generation 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 10.3: 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 ersichtlich.

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 10.4: 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 10.5: Logdatei: Wechsel Garbage Collection Strategie

Aus unterschiedlichen Gründen kann es zu einem Wechsel der Garbage Collection Strategie kommen. Dieser kann, zusammen mit dem Grund, ausgewertet werden.

10.6.3 Domänenmodell JRockit Garbage Collection

Nach der syntaktischen und sematischen Analyse wird eine Instanz des Domänenmodells erzeugt. Dieses wurde aufgrund der Randbedingungen (siehe Abschnitt 7.2.3) analog der Struktur der Garbage Collection in der JRockit virtual Machine abstrahiert. Die rohen Daten sind also danach über diesen Objektgraphen verfügbar.

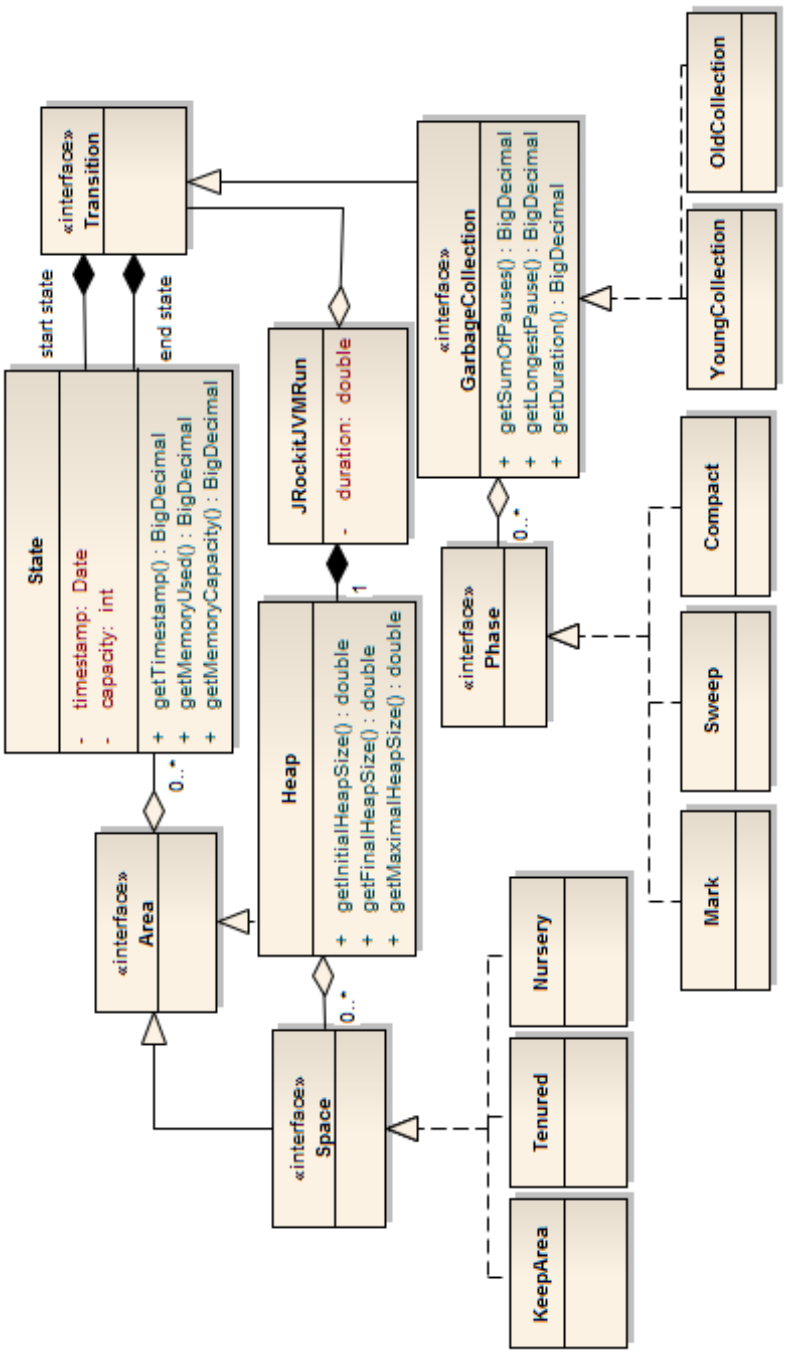

Abbildung 10.4: Domänenmodell: Garbage Collection (JRockit Implementation)

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 (*State*). Beim Übergang (*Transition*) vom einen in den anderen Zustand findet eine Garbage Collection statt. Für jeden Zustand gibt es Statusinformationen welche die aktuellen Metriken (Kapazität, Auslastung, etc.) für den entsprechenden Speicherbereich beschreiben. Das starten einer Transition wird durch einen Event ausgelöst (im Diagramm nicht ersichtlich). Events werden aufgrund von heuristischen Daten der Lauzeitumgebung ausgelöst - zum Beispiel wenn die Nursery oder die Old Generation ihre Speichergrenzen erreichen. Eine Garbage Collection besteht aus unterschiedlichen Phasen (*Mark*, *Sweep*, *Compact*).

10.7 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.

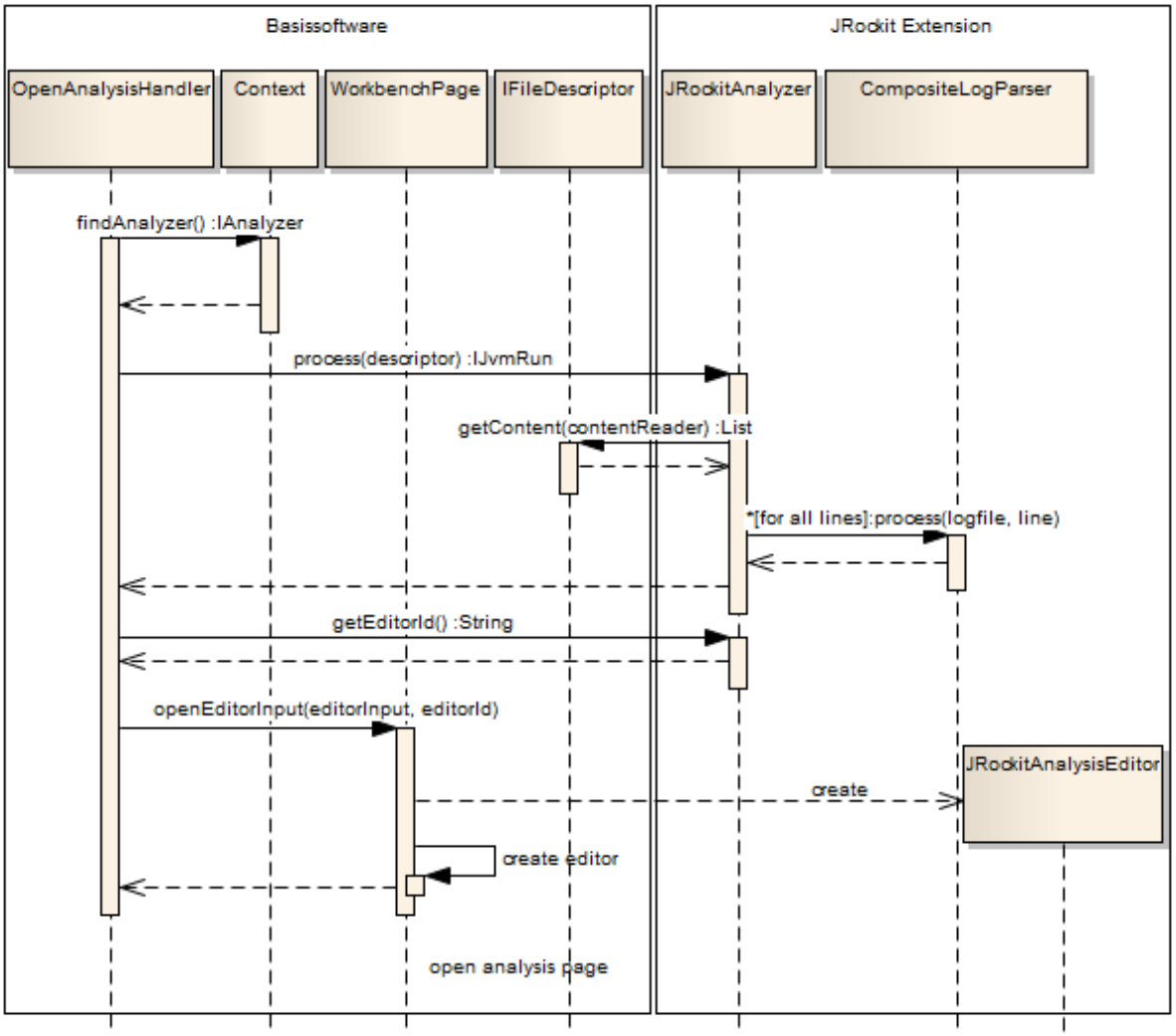

Abbildung 10.5: Sequenz-Diagramm Öffnen der Analyse

Der Ablauf beim Öffnen des Analysefensters 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 (siehe Abschnitt 10.6.3).
3. Der Editor wird geöffnet.

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

10.8 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-Generation
- Speicherbedarf maximal und durchschnittlich: Aus den Informationen des Speicherverbrauchs vor und nach jeder Garbage Collection wird der durchschnittliche und der maximale Bedarf berechnet.
- Kapazität maximal und durchschnittlich: 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 stattgefunden.
- Anzahl Garbage Collections: Wie viele Garbage Collections hat es insgesamt gegeben.
- Anzahl Old Collections: Wie viele Old Collections hat es gegeben.
- Anzahl Young Collections: Wie viele 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 4.4.1)
- Durchschnittliches Intervall 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

10.9 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.

10.9.1 Datenquellen

Dieser Abschnitt zeigt woher die Daten für die jeweiligen Achsen kommen:

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()

10.10 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.

10.10.1 Datenquellen

Dieser Abschnitt zeigt woher die Daten für die jeweiligen Achsen kommen:

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 10.6.3

10.11 Profil erstellen (FRQ-11)

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

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

10.11.1 Domänenmodell zur Persistierung der Profile

Als Struktur zur Speicherung der Profile zusammen mit den definierten Diagrammen und den sich darauf befindenden Datenserien dient folgendes Modell:

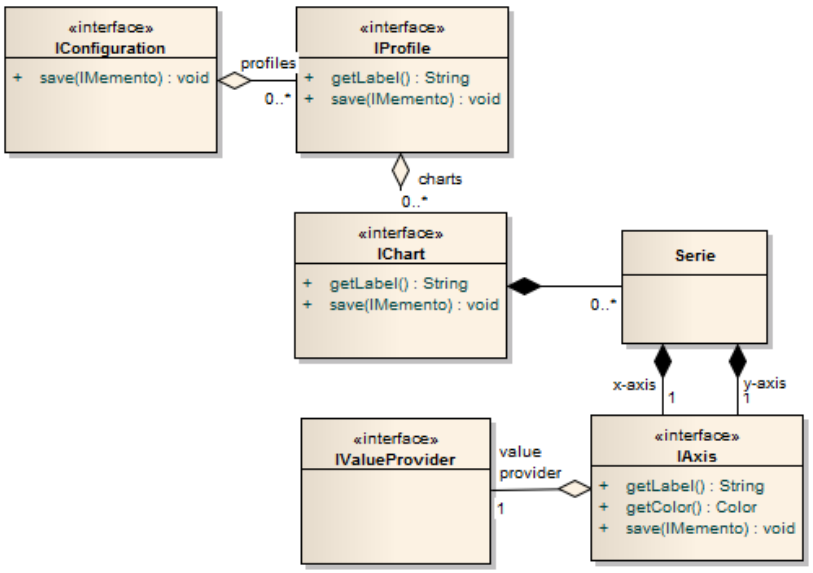

Abbildung 10.6: Domänenmodell: Profile

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

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.

10.12 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 auf das dahinter liegende View-Modell (siehe Abschnitt 10.11.1) propagiert. Die Administrationsoberfläche für einen Chart umfasst initial zwei Abschnitte:

- Serie erstellen, definieren und hinzufügen
- Serie löschen

10.13 Profil speichern (FRQ-13)

Mittels des Memento-(siehe Anhang B.1) und Visitor-Patterns[3, S. 331] werden die Profile im Eclipse-Context gespeichert. Dies hat den Vorteil, dass man sich über das Speicher-Format keine Gedanken machen muss.

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

Kapitel 11

Generelle Aspekte

Dieser Abschnitt konzipiert die nichtfunktionalen Anforderungen (Qualitätsanforderungen).

11.1 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 werden als HTML-Seiten umgesetzt und vom internen Browser angezeigt. Es gibt zwei verschiedene Hilfetypen:

- **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:** Hilfeseiten, welche den Bezug auf eine Aktion oder ein Fenster haben, werden über einen Schlüssel mit der jeweiligen Komponente verlinkt.

11.2 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. Für jedes Modul wird ein eigenes Test-Projekt erstellt. Als Testing-Framework wird der defakto Standard JUnit verwendet. JUnit ist sowohl von allen Entwicklungsumgebungen als auch von Maven und Tycho unterstützt.

[1] Fragmente sind Erweiterungen für ein Projekt (Host) und haben vollen Zugriff auf die Klassen ihres Host-Bundles.

11.3 Internationalisierung (QRQ-S-03))

Die Analysesoftware soll in den Sprachen Deutsch und Englisch verfügbar sein. Die gewählte Sprache innerhalb der Erweiterung wird von der Entwicklungsumgebung übernommen[2] und kann nicht via ein Menu geändert werden. Der Eclipse Lokalisierungsmechanismus basiert auf der Java *Locale* und löst die Ressourcen für die ausgewählte Sprache aus Properties-Dateien auf. Je nachdem wo der Text platziert ist, wird die Sprachressource auf unterschiedliche Weise aufgelöst:

- **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. Die extrahierten Sprachressourcen werden im Klassenpfad abgelegt.
- **Texte, Labels in Deskriptoren:** Die sich in den Eclipse-Deskriptoren (plugin.xml, Manifest.MF) befindenden sprachabhängigen Plugin-Informationen wie Organisation, Name und Beschreibung werden ebenfalls in Properties-Dateien abgelegt. Diese werden bei der Installation oder während einem Update angezeigt. Der Ort und Name dieser Dateien ist per Konvention *\${plugin}\OSGI-INF\I10n\bundle_${lang}.properties*. Der Inhalt wird vom Eclipse-Framework geladen.
- **Hilfesystem:** Die HTML-Hilfeseiten können ebenfalls in unterschiedlichen Sprachen definiert und angezeigt werden. Siehe Abschnit 11.1.

11.4 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

[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.

der Fall wäre, würde er die Arbeit beenden und das bereits Erledigte aufräumen.

11.5 Korrektheit (QRQ-S-05))

Um mit Java-Applikationen genaue Werte zu berechnen wird unter [1] empfohlen, die Klasse *BigDecimal* anstelle von *Double* und *Long* zu verwenden. Die in der Analysesoftware angezeigten Werte haben eine Genauigkeit von mindestens einem Zehntel (0.1).

Kapitel 12

Infrastruktur

Build-Automatisierung, Issue Tracking und Versionsverwaltung sind im Context der Analysesoftware. Dieser Abschnitt beschreibt das Projektsetup (Build, Integration, Konfigurations und Change Management).

12.1 Verwendete Werkzeuge

12.1.1 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. Im Bereich der Eclipse Rich Client Entwicklung kann entweder PDE Build, ein auf Apache Ant basiertes Build-System für Eclipse RCP Applikationen[13] oder die Maven-Integration Tycho[1] verwendet werden. Als Werkzeug zum automatisierten Build der Software wird Maven Tycho 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 ist einfach und verständlich. PDE Build basiert auf der Build-Software Ant und ist aufwändig zu konfigurieren.
- Maven ist de facto Standard bei den Build-Werkzeugen und wird von allen Continuous Integration Systemen unterstützt.

12.1.2 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-

[1] http://tycho.sonatype.org
[2] Möglichst viel wird standardisiert nur Abweichungen konfiguriert.

Requests, etc. Sie lässt sich auch gut mit allen gängigen Systemen zur Versionsverwaltung integrieren.

12.1.3 Versionsverwaltung

Als Versionsverwaltungssysteme kommen mehrere Werkzeuge in Frage, darunter befinden sich auch CVS und Subversion. Git[3] ist ein verteiltes Versionsverwaltungssystem und ist konzeptionell besser als Subversion und CVS[4]. Eine empfehlenswerte Einführung in Git kann unter [2] gefunden werden. Als zentrales Repository wird Github[5] verwendet. Nur von diesem zentralen Repository werden Release-Builds durchgeführt.

12.1.4 Continous Integration

Continuous Integration Systeme dienen zur Steigerung der Softwarequalität. Sie machen dies, indem sie alle Tests und den Gesamtbuild der Software periodisch, beispielsweise jede Stunde, durchführen. In letzter Zeit hat das Continuous Integration System *Hudson* an Popularität gewonnen. Es erfüllt die Anforderungen aus dem Projekt:

- **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.
- **Einfache Installation und Einrichten des Projektes**

12.2 Konfigurationsmanagement

Jedes Deployment entspricht einer Version in Jira (siehe Abschnitt 12.1.2). Die Versionsnummern werden nach folgendem Muster aufgebaut:

<Major>.<Minor>.<Build>

Gestartet wird mit Version 1.0.0.

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

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

[5] http://github.com

12.2.1 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.

12.2.2 Versionsverwaltung

Das zentrale Repository dieses Projektes befindet sich auf Github[6]. Es ist jederzeit verfügbar und kann von allen Clients (Entwickler, Continuous Integration) jederzeit angesprochen werden. Ein neuer Entwickler klont das Repository auf seinen Rechner.

Übersicht

Das Repository wird folgendermassen aufgesetzt:

- **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 aus dem *release*-Branch statt. Dazu wird daraus ein neuer Branch erzeugt. Die Bugfix-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).

[6] 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 die Bugfixes zurück in den *release*-Branch (merge). Die Zuordnung der Commits mit den Jira-Bugs wird über den Kommentar beim Git-Commit mit dem Jira-Bug 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. Ausgehend von zwei Feature-Anfragen ist das Vorgehen 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 den *feature-1.1.0*-Branch zurück in den *master*-Branch (merge). Die Zuordnung der Commits mit den Jira-Features wird über den Commit-Kommentar gemacht. Die Anpassung des Kommentars von vorherigen Commits kann mittels einem Rebasing (siehe [2]) gemacht werden.

Teil V

Umsetzung

Kapitel 13

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 C.

13.1 Funktionale Anforderungen

13.1.1 Installation (FRQ-01)

Die Installation der beiden Features *Core* und *JRockit Extension* kann über das Netzwerk gemacht werden[1]. Diese Funktionalität ist Bestandteil des Eclipse-Frameworks: beide Features werden über eine Installations-Seite bereitgestellt. Die Features wiederum bestehen aus den einzelnen in Abschnitt 9.2.1 beschriebenen Projekten. Der Release der Artefakten wird durch das Continuous Integration System Hudson gemacht und auf die Update-Seite kopiert.

[1] Aktuell befindet sich der Installations-Server noch in einem nicht öffentlichen Netzwerk.

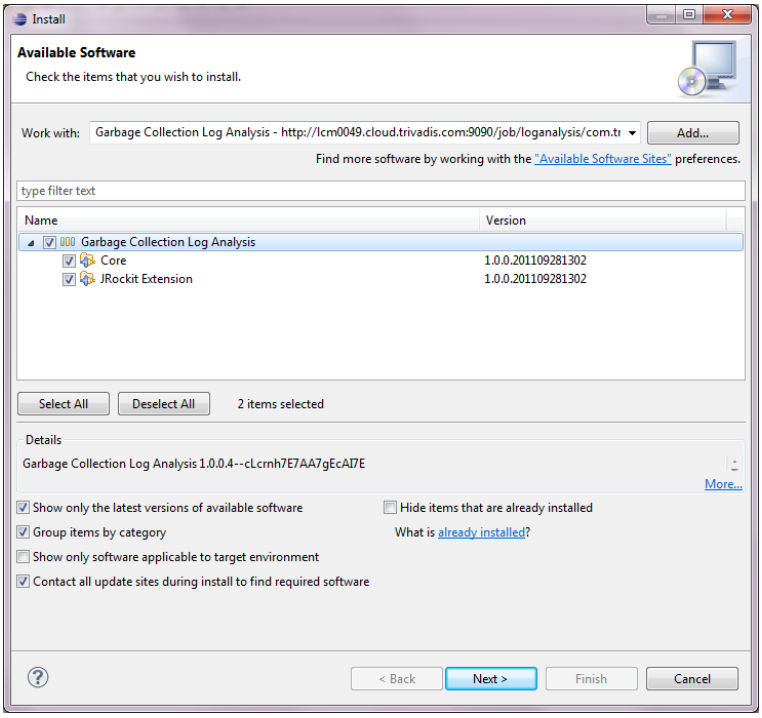

Abbildung 13.1: Installation Garbage Collection Log Analyse

13.2 Update (FRQ-02)

Ein Update entspricht prinzipiell einer Neuinstallation der Software. Die einzelnen Module werden zur Laufzeit ausgewechselt und deren Classloaders ausgewechselt. Der Update-Bildschirm unterscheidet sich nur marginal vom Installations-Bildschirm, er ist aber über einen eigenen Menü-Eintrag aufrufbar.

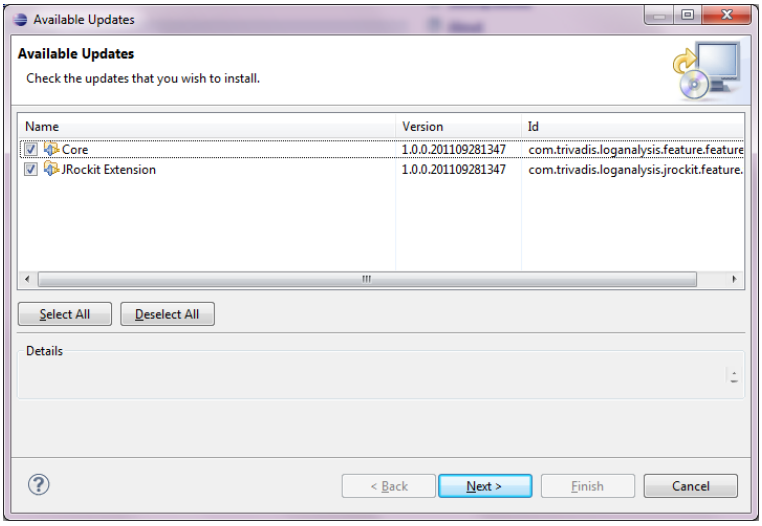

Abbildung 13.2: Update Garbage Collection Log Analyse

13.2.1 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 B.1) statt. Der Import findet über einen Wizard[2] statt. Der Benutzer wählt das Verzeichnis und die zu importierenden Logdateien aus.

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

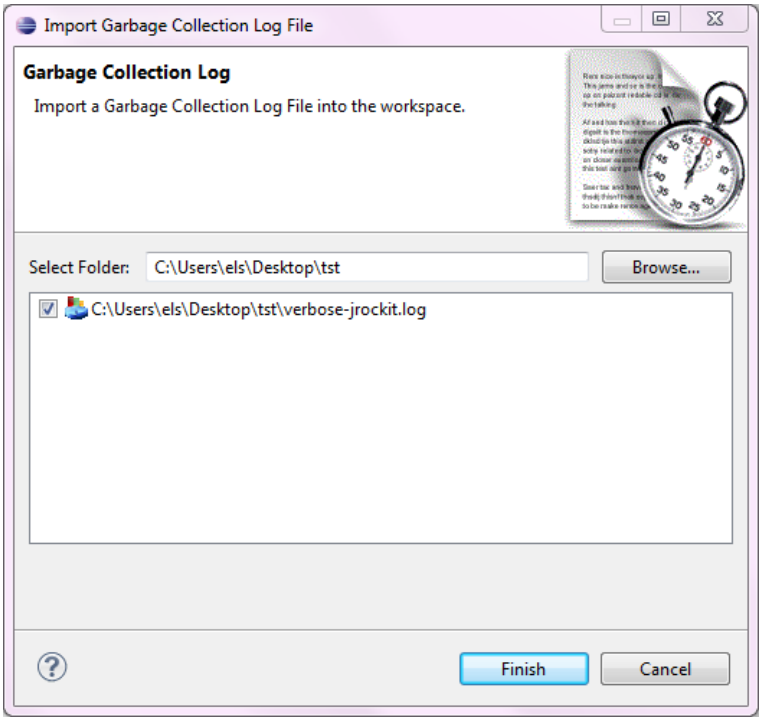

Abbildung 13.3: Logdatei importieren

13.2.2 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.

13.2.3 Garbage Collection Logdatei parsen (FRQ-06)

Die Log-Ausgaben des Memory-Moduls (Log-Level: info) werden mittels Regulären Ausdrücken geparst und in die strukturierte Form (siehe Abschnitt 10.6.3) gebracht. Die Analyse wird realisiert durch die Implementation eines Analyzers. Folgende Methoden müssen implementiert werden:

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

Durch die Abstraktion des Interfaces *Analyzer<T>* können später weitere Implementationen für neue Log-Formate erstellt werden.

13.2.4 Standardauswertung anzeigen (FRQ-07)

Für die importierten Dateien besteht die Möglichkeit, sie in der Standardauswertung anzuzeigen. Die Standardauswertung zeigt die im Konzept definierten Ansichten (siehe Abschnitt 10.8):

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

Als Beispiel die Ansicht des Heap-Diagrammes:

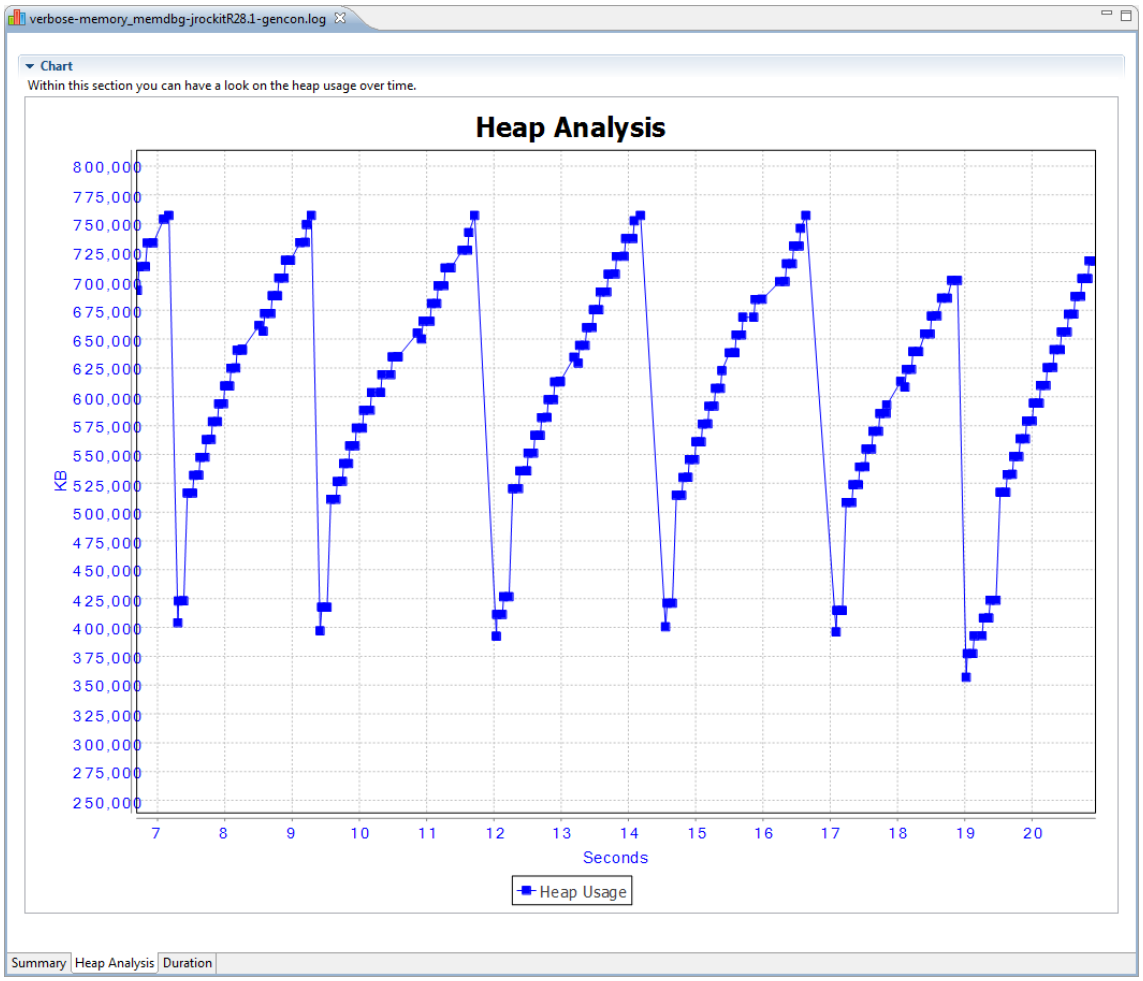

Abbildung 13.4: Standardauswertung: Heap Analyse

13.2.5 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). Speichern (FRQ-13), exportieren (FRQ-14) und importieren (FRQ-15) wird über das Memento-Pattern gemacht. Für weiter Informationen zum Memento siehe Anhang B.1.

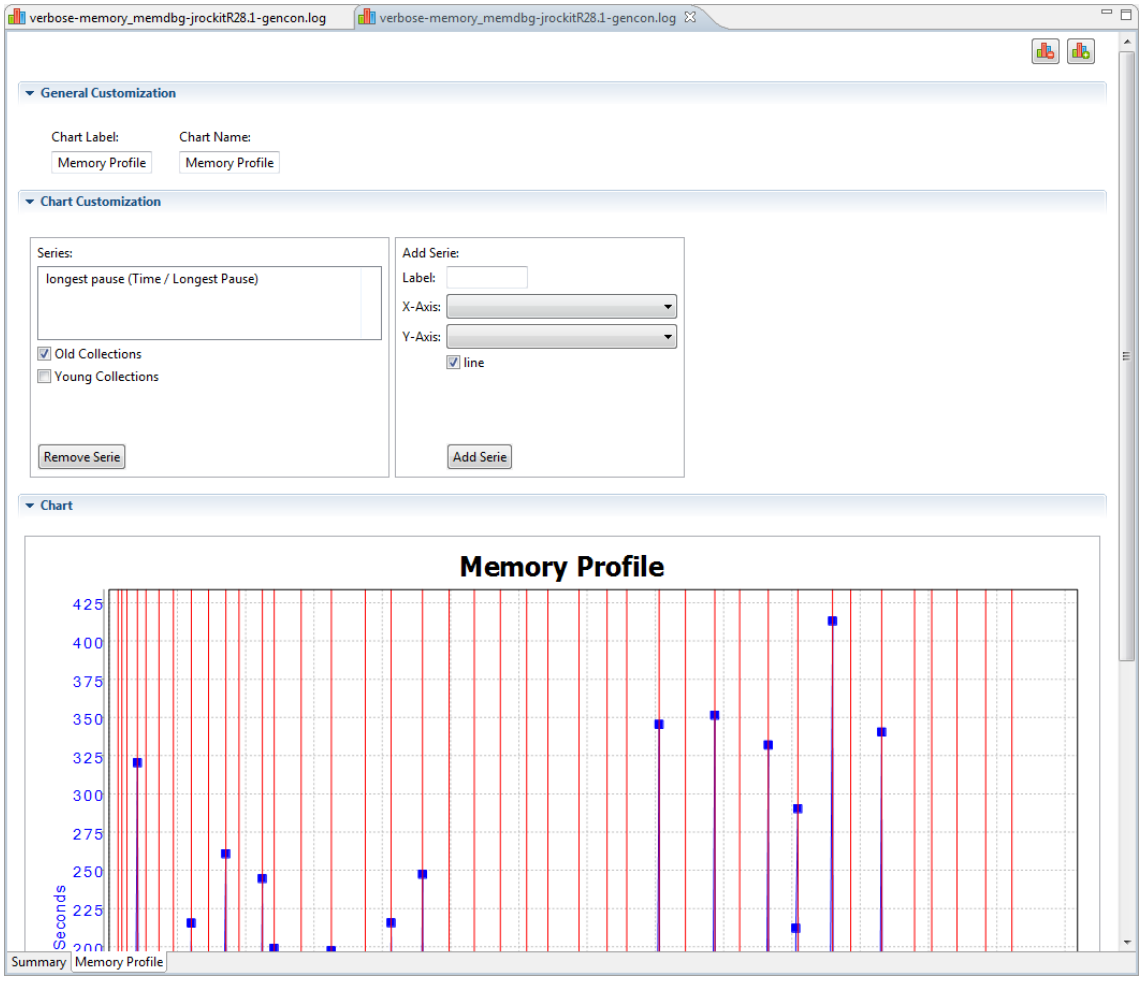

Abbildung 13.5: Benutzerdefinierte Auswertung definieren

13.2.6 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

umgesetzt, diese können aber sehr einfach um weitere Sprachen ergänzt werden. Die Sprache wird durch den Benutzer in der Datei *eclipse.ini* definiert oder von der Java Laufzeitumgebung übernommen.

13.3 Qualitätsanforderungen Software

13.3.1 Erweiterbarkeit (QRQ-S-01)

Die Analyse für JRockit Logs wurde als Erweiterung implementiert. Es besteht die Möglichkeit, dass sich mehrere Erweiterungen bei der Basissoftware registrieren und damit auch das Parsen von anderen Logdateien möglich wird. Um ein neues Log-Format zu unterstützen, müssen folgende Aufgaben ausgeführt werden:

- Plugin erstellen in welches die Funktionalität getan wird.
- Feature erstellen welches aus dem oben definierten Plugin besteht.
- Feature auf der Update-Seite hinzufügen, so dass anschliessend auch dieses installiert werden kann.
- Implementation 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

13.3.2 Testabdeckung (QRQ-S-02)

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

13.3.3 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.

13.3.4 Usability (QRQ-S-04)

Lange dauernde Operationen (einlesen, parsen der Daten) werden vom Eclipse-Framework asynchron gestartet. Dem Benutzer wird ein Progress-Monitor angezeigt. Es besteht also die Möglichkeit, den Prozess zu unterbrechen.

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

Alle zu berechnenden Werte befinden sich im Datentyp *BigDecimal*. Dies führt zwar zu einem erhöhten Ressourcenverbrauch, ist aber für die geforderte Genauigkeit nötig.

Kapitel 14

Review und Ausblick

Die Implementation beinhaltet alle funktionalen Anforderungen. Die Qualitätsanforderungen *Testabdeckung* (QRQ-S-02) und *Internationalisierung* (QRQ-S-03) sind teilweise umgesetzt. Dieser Abschnitt beleuchtet den Scope der Software und zeigt auf, wo weiteres Entwicklungspotential liegt.

14.1 Was das Tool leistet

Wenn die Auslastung einer CPU konstant hoch ist aber nicht durch das System verursacht wird, kann unter Umständen das Tuning des Garbage Collectors Sinn machen. Zur Auswertung einer laufenden virtuellen Maschine, kann auch die Software JRockit Mission Control von Oracle verwendet werden. Oft ist allerdings der Zugriff via Mission Control auf den Server nicht möglich[1]. In diesem Fall müssen zur Analyse der Garbage Collection die erstellten Logdateien verwendet werden. Mit der Analysesoftware kann diese Aufgabe vereinfacht werden, indem beispielsweise der Verlauf der Garbage Collection Zeiten oder des durch die Applikation benötigten Arbeitsspeichers angeschaut werden kann. Wichtig dabei können auch Metriken wie Durchsatz oder die durchschnittliche Pausenzeit sein.

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. Um eine Auswertung einer Logdatei zu machen, muss eine Logdatei mit den Ausgaben der Speicherverwaltung erstellt werden[2].

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.

[1] Es gibt Firewall-Regeln oder man befindet sich ausserhalb des Server-Subnetzes

[2] Die Aktivierung des Loggers kann über das Argument -Xverbose:memory gemacht werden

14.2 Ausblick

14.2.1 Funktionsumfang

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 noch nicht möglich.

Weiter ist man beim Performance Tuning in der Regel nicht an den Informationen über die ganze Zeit interessiert, die Möglichkeit zur Wahl von Start- und Endzeitpunkt der auszuwertenden Daten wäre deshalb wünschenswert.

14.2.2 Weitere Daten

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 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 14.1: 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 zu einer Garbage Coolection führt.
- **Dauer der einzelnen Garbage Collection Phasen (Initial Marking, Precleaning, Final Marking):** Nicht alle diese 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.

14.2.3 Andere Log-Formate

Wie bereits erwähnt, können die Logs der Version R27 noch nicht ausgewertet werden. Weil diese Version noch an vielen Orten im Einsatz ist, muss bald die Kompatibilität mit Release 27 hergestellt werden.

Im Bereich der HotSpot virtual Machine ist seit Version 1.6.0_14 des Java Runtime Environments eine Vorversion des G1 Garbage Collectors[3] verfügbar. Die Funktionsweise dieses Algorithmus unterscheidet sich stark von den bisherigen Versionen des Mark & Sweep Algorithmus. Die Logdateien dieses Collectors können aktuell noch durch keine Software ausgewertet werden[7]. Mit der Auswertungsmöglichkeit für diesen Algorithmus könnte man sich auf dem Markt besser positionieren.

[3] G1 ist auch unter dem Namen Garbage First Garbage Collector bekannt.

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] Kirk 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] Kai Tödter. Eclipse rcp vs netbeans platform. http://www.toedter.com/blog/?p=14, 2007.

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

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

[15] 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].

[16] Wikipedia. Netbeans ide — wikipedia, die freie enzyklopädie, 2011. [Online; Stand 27. November 2011].

[17] Wikipedia. Software requirements specification — wikipedia, die freie enzyklopädie, 2011. [Online; Stand 3. Dezember 2011].

Anhang

Anhang A

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 Desktop 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.	-
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
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 definieren die Abhängigkeiten, die Exportierten Klassen und die Erweiterungspunkte.	Eclipse
Rich Client Framework	Software-Bibliothek, mittels welchem sich Rich Client Applikationen entwickeln lassen.	-

Rich Client Plattform	Der Begriff Rich Client Plattform wird in diesem Dokument oft 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-Seite im Eclipse-Umfeld ist eine per Konvention aufgebaute Webseite die via das Http-Protokoll zugänglich ist und Software-Komponenten (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
User Interface	Als User Interface (Benutzeroberfläche) werden alle Schnittstellen zwischen Mensch und Maschine bezeichnet. Diese können textbasiert oder grafisch sein (GUI).	-

Tabelle A.1: Glossar

Anhang B

Eclipse Rich Client Framework

B.1 Zustand von Komponenten 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 eignen sich besonders zur Persistierung von hierarchischen Strukturen. Das Eclipse-Framework speichert die Zustände von Views mittels eines XMLMemento im Eclipse-Context ab. Die Initialisierung der Komponente wird wiederum durch das Überschreiben der Methode *init(site:IViewSite, memento:IMemento)* der Klasse *ViewPart* gemacht.

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

Anhang C

Bedienungsanleitung

C.1 Installation der Software

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

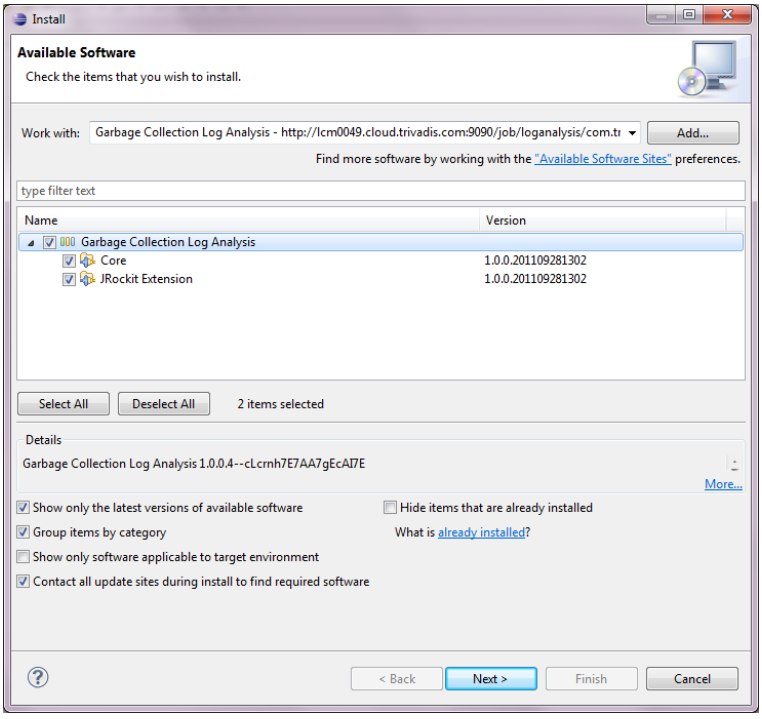

Abbildung C.1: Installation Garbage Collection Log Analyse

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.

C.2 Update

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

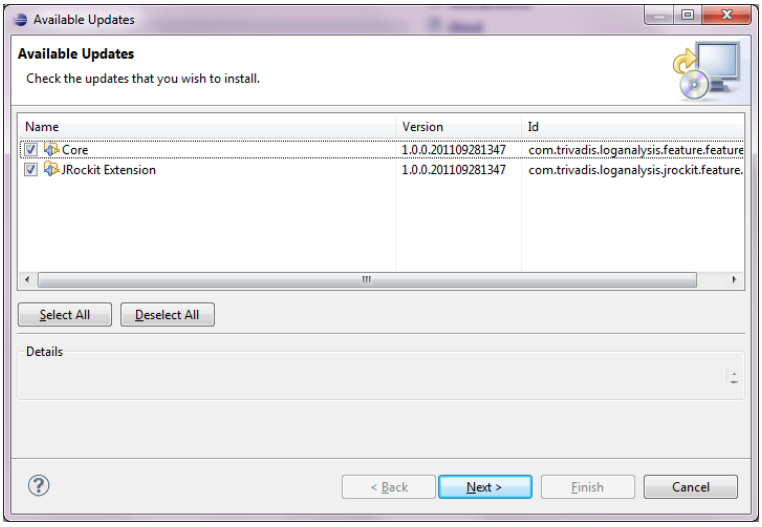

Abbildung C.2: Update Garbage Collection Log Analyse

C.3 Dashboard

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

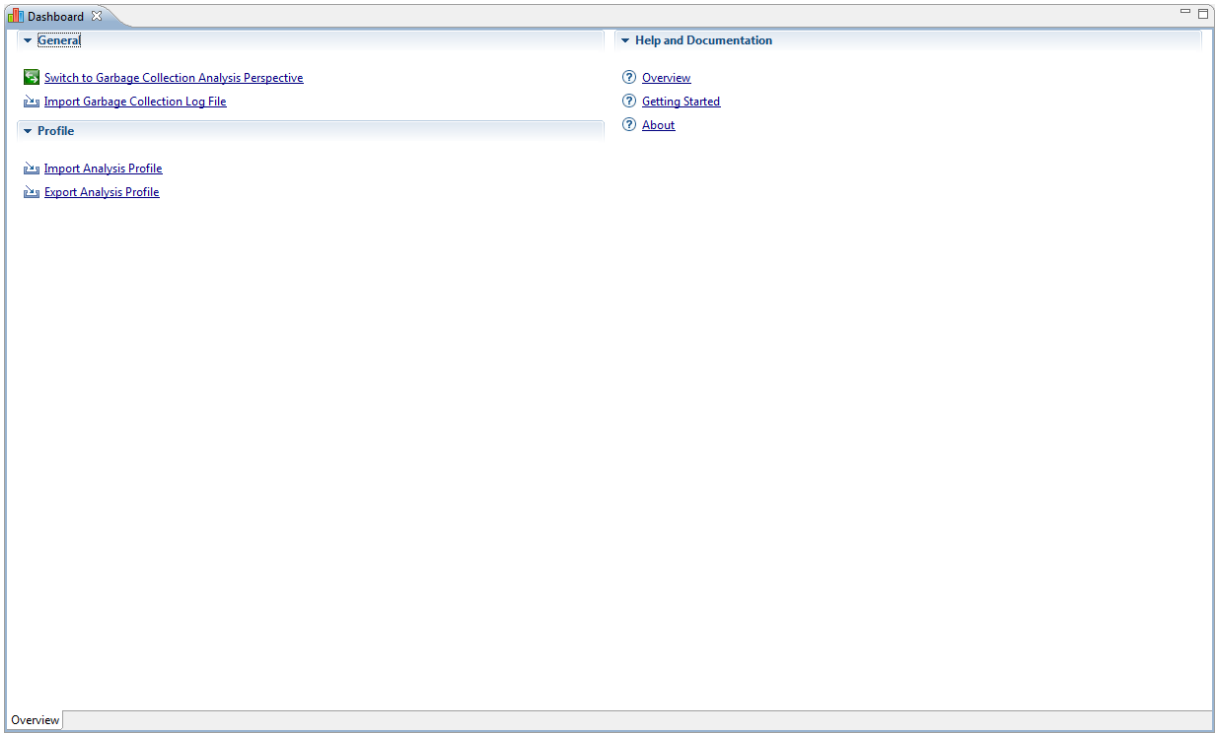

Abbildung C.3: Dashboard

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

C.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.

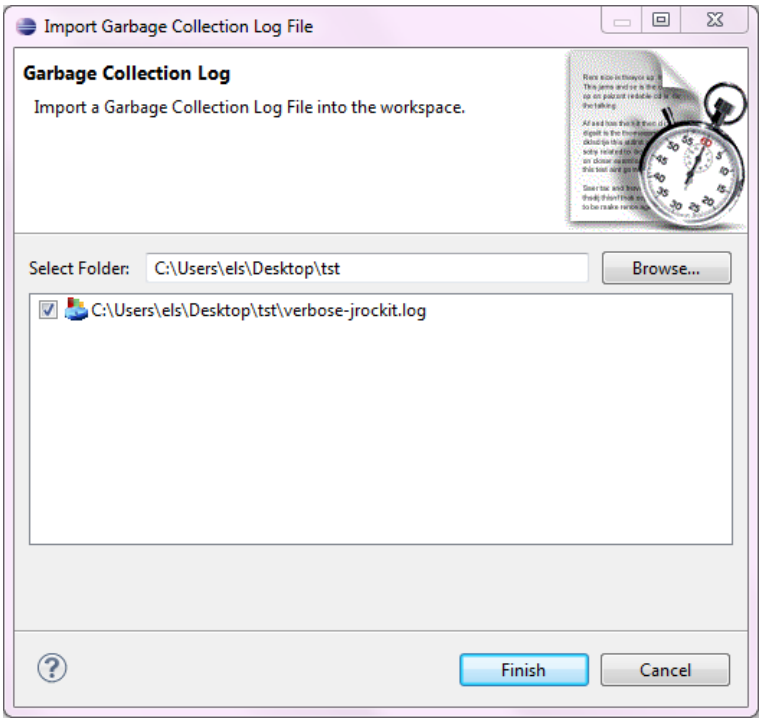

Abbildung C.4: Logdatei importieren

C.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.

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.

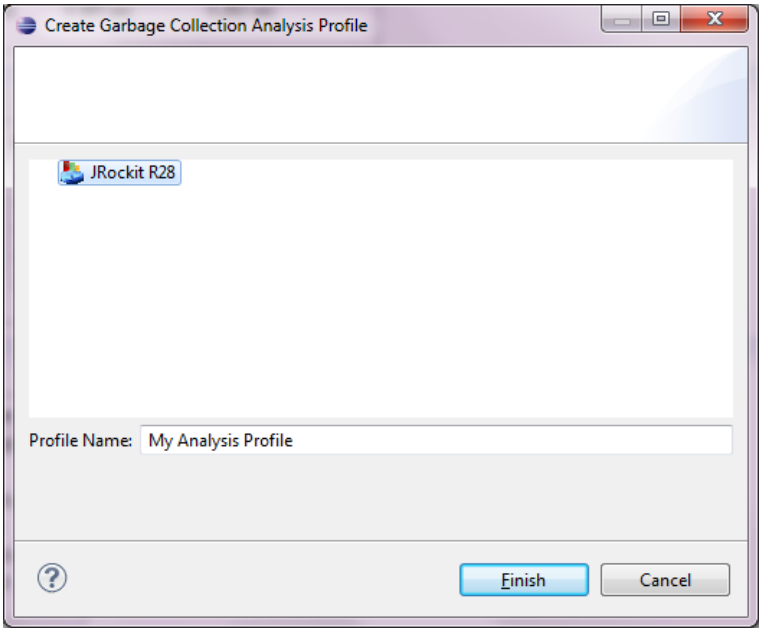

Abbildung C.5: Profil erstellen

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

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

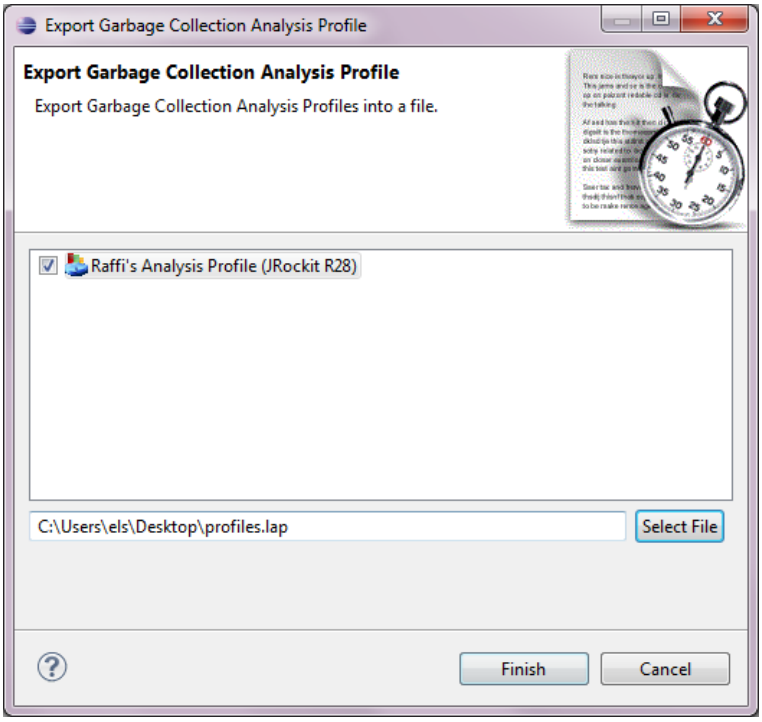

Abbildung C.6: Profil exportieren

Profil importieren

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

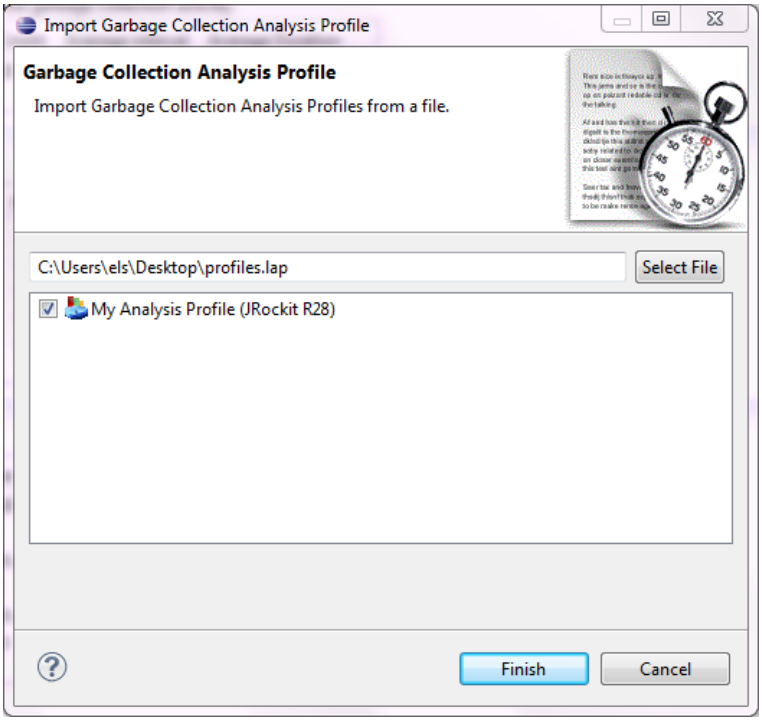

Abbildung C.7: Profil importieren

C.6 Garbage Collection Analyse

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

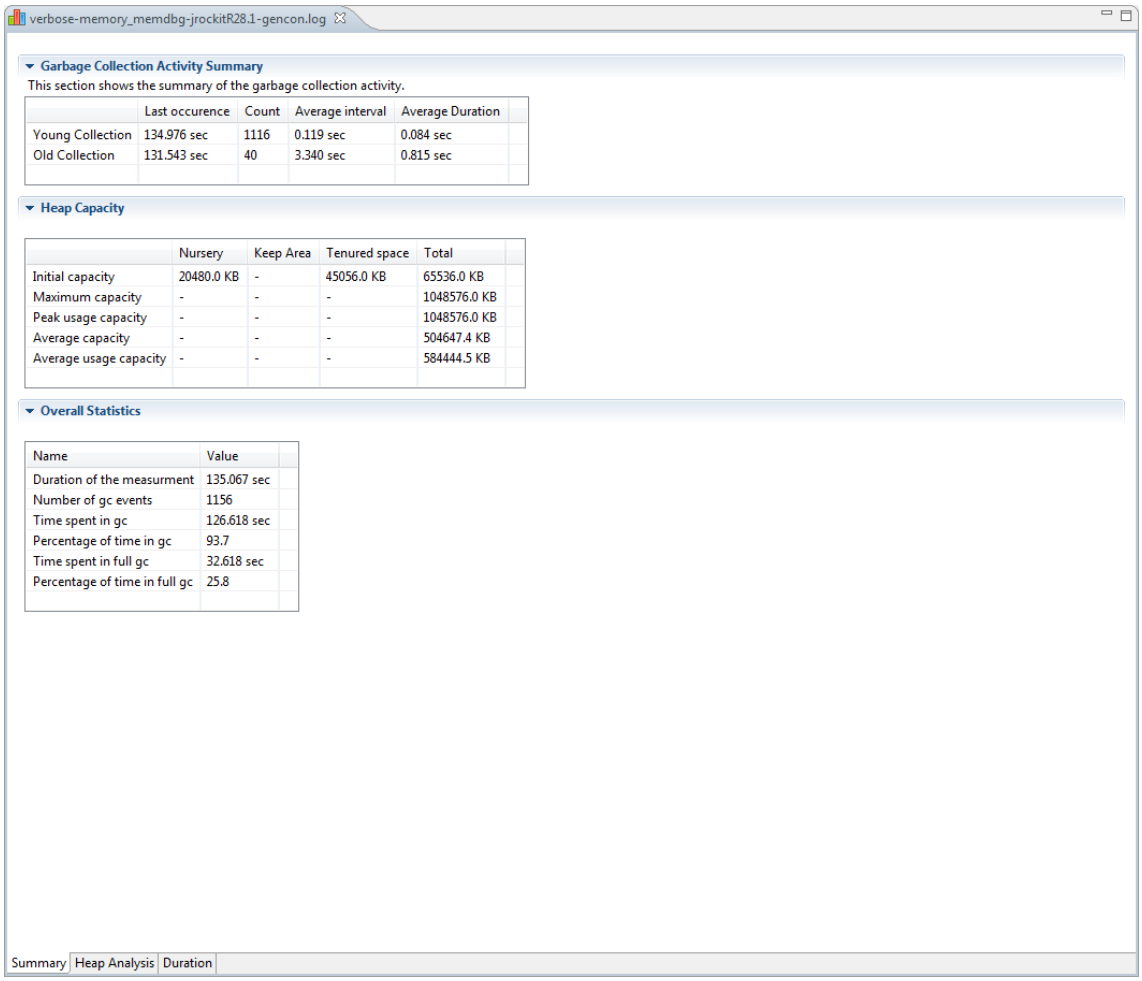

Abbildung C.8: Standardauswertung: Zusammenfassung

Heap Analyse

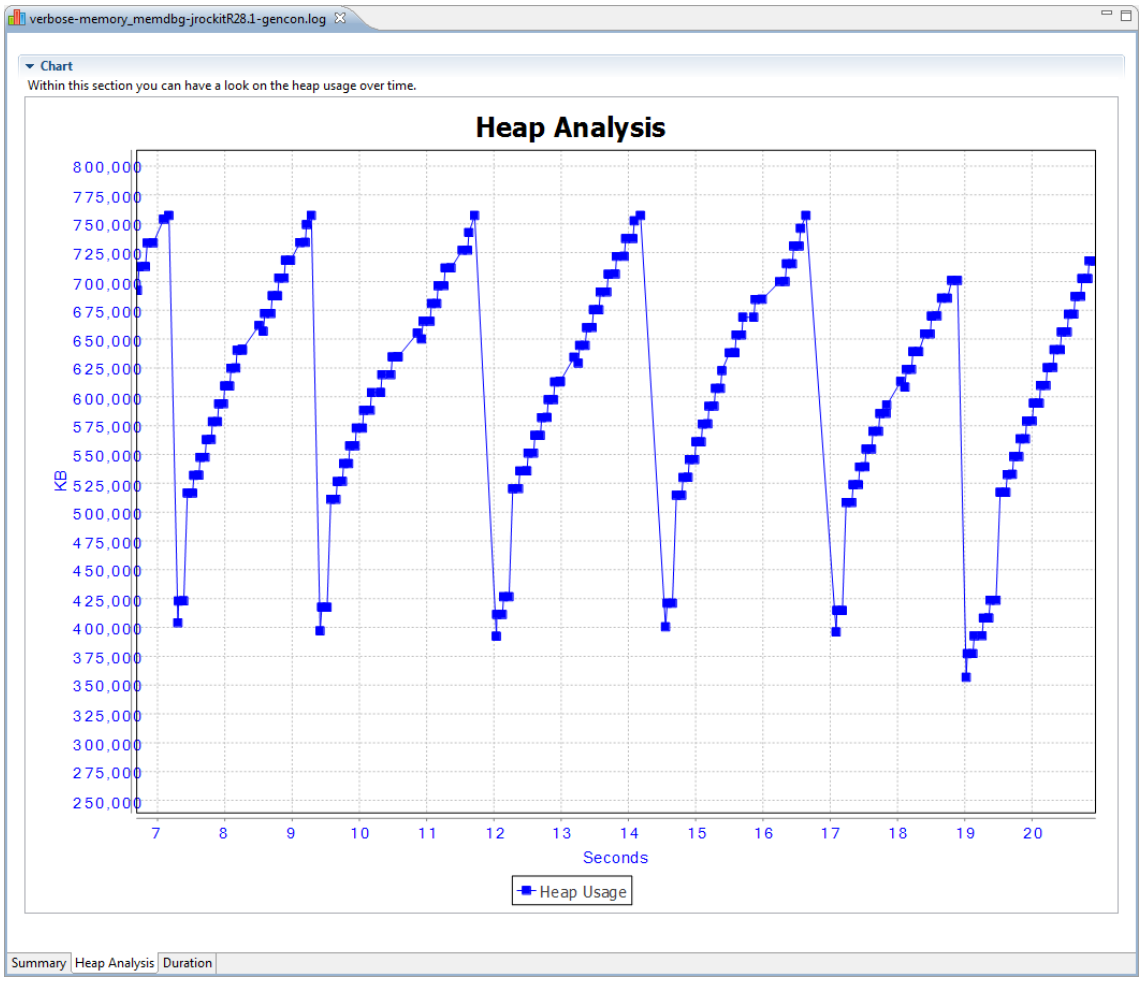

Abbildung C.9: Standardauswertung: Heap Analyse

Dauer Garbage Collection

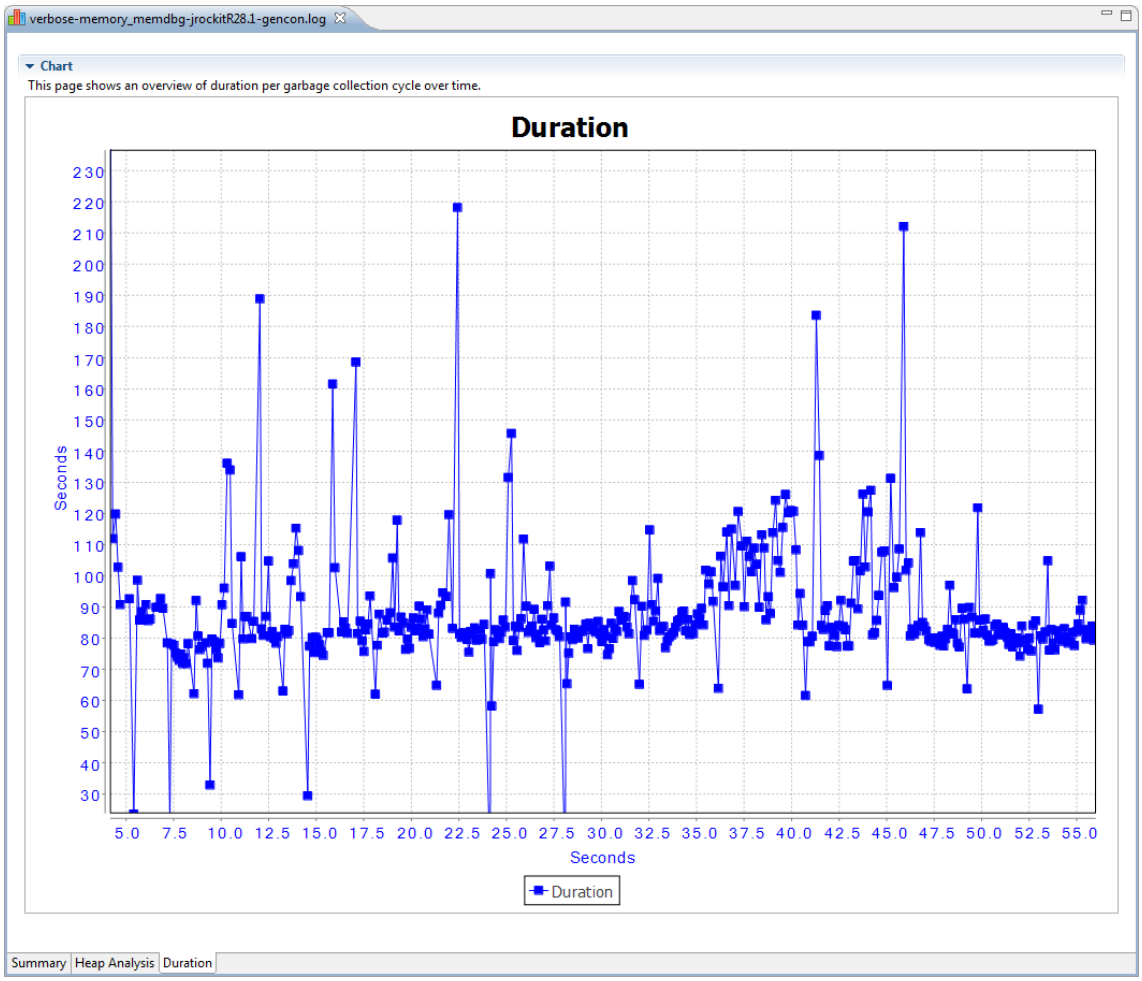

Abbildung C.10: Standardauswertung: Dauer Garbage Collection

Benutzerdefinierte Auswertung (Profile)

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

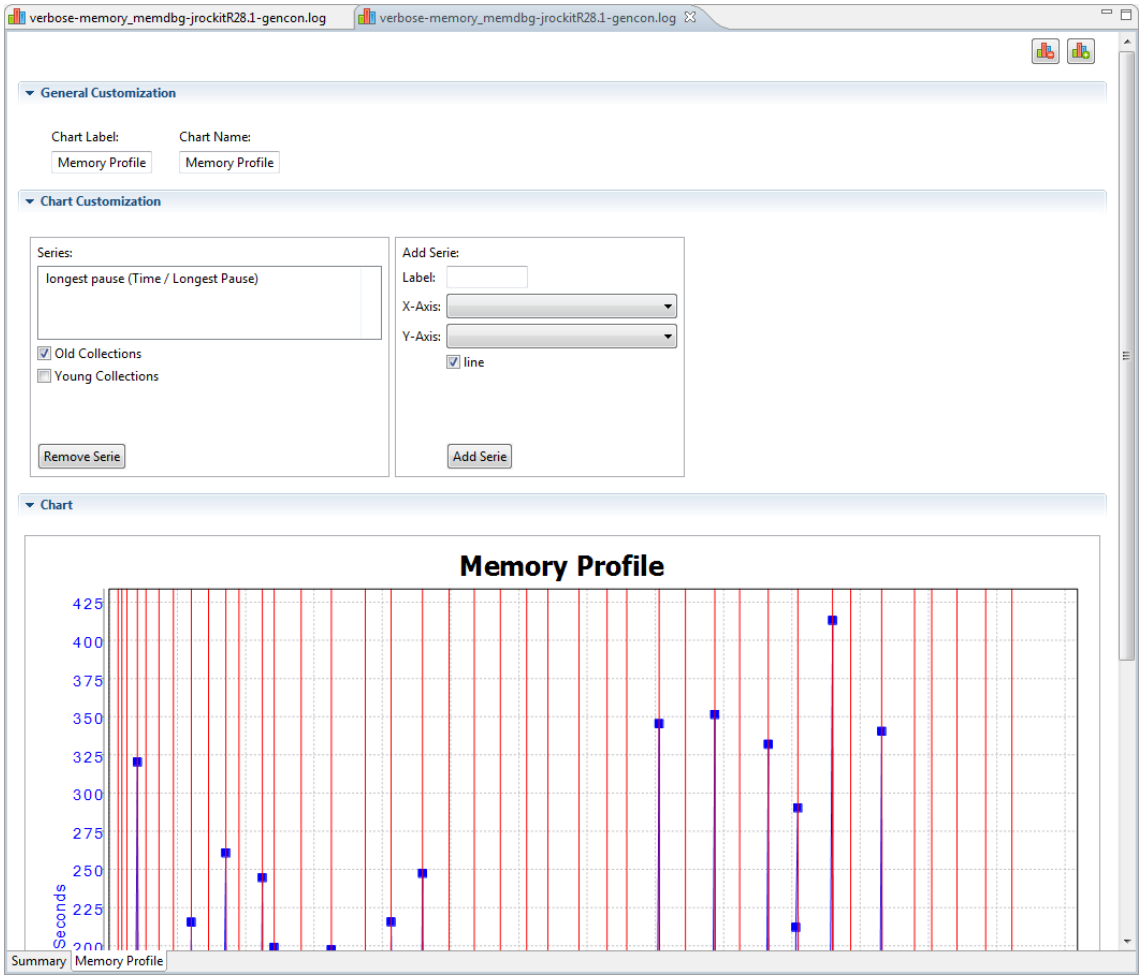

Abbildung C.11: Benutzerdefinierte Auswertung definieren

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 D

Informationen

D.1 Inhalt Datenträger

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

Tabelle D.1: Inhalt Datenträger

D.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 D.1: Checkout Quelltext Repository

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