# Bachelorthesis

## Konzeption und Entwicklung eines Auswertungstools für die Log Files des JRockit Garbage Collectors

**von**

**Raffael Schmid**

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

# Zusammenfassung

# Inhaltsverzeichnis

# Einleitung

TODO

# Teil I

# Projektbeschreibung

# Kapitel 1

# Aufgabenstellung

## 1.1 Ausgangslage

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

## 1.2 Ziele der Arbeit

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

## 1.3 Aufgabenstellung

Im Rahmen der Bachelorthesis werden vom Studenten folgende Aufgaben durchgeführt:

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

## 1.4 Erwartete Resultate

Die erwarteten Resultate dieser Bachelorthesis sind:

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

## 1.5 Geplante Termine

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

# Teil II

# Umsetzung

# Kapitel 2

# Analyse der Aufgabenstellung

## 2.1 Übersicht

Wie in der Aufgabenstellung definiert, ist das Ziel dieser Arbeit die Konzeption und Implementation einer Analysesoftware für die Garbage Collection Log Dateien der JRockit Virtual Machine. Grundlegendes Ziel einer solchen Software ist es, die anfallende grosse Menge an Daten übersichtlich darzustellen und verschiedene Sichten auf diese Daten zu ermöglichen. Um die Anforderungen an diese Software zu ermitteln, ist ein die Studie der Grundlagen der Garbage Collection und im spezifischen die der JRockit Virtual Machine notwendig. Anschliessend wird eine Stärken-/Schwächen-Analyse der bestehenden Richt-Client Frameworks gemacht, aufgrund welcher nach der Aufnahme der Anforderungen der Entscheid für das zu verwendende Framework getroffen werden kann. Im Anschluss an diese Tätigkeiten folgt die Konzeption und Implementation der Software. Die genauere Untersuchung der einzelnen Teilaufgaben wird in den folgenden Abschnitten gemacht:

### 2.1.1 Erarbeitung der Grundlagen Bereich der Garbage Collection

Die Erarbeitung der Grundlagen dient als Basis in zwei Bereichen:

- **Anforderungsanalyse:** Aus Sicht des Analysten respektive dem Benutzer dieser Software ist es essentiell, dass er die richtigen Daten der Garbage Collection, die richtige Sicht auf diese Daten und die richtigen Filter-Funktionen vorfindet. Voraussetzung für diese Anforderungen sind die Kenntnisse der verschiedenen Garbage Collection Algorithmen im Generellen und spezifisch die der JRockit Virtual Machine.
- **Konzept:** Die Konzeption des Domänen-Modells und des Einlese-Prozesses

der Log Dateien bedingt eine genaue Kenntnis der verschiedenen Strategien und Formaten der Ausgaben.

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

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

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

Die Anforderungen werden zusammen mit einem Performance-Analysten ermittelt und nach [8, 4.3.2 Angepasste Standardinhalte] dokumentiert. Laut dem IEEE[1] und [8, 4.5 Qualitätskriterien für das Anforderungsdokument] müssen Anforderungen folgende Kriterien erfüllen:

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

### 2.1.4 Auswahl der zu verwendenden Frameworks

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

### 2.1.5 Konzeption und Implementation

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

### 2.1.6 Bewertung

Die Bewertung der Software wird auf Basis der Anforderungen gemacht.

[1] Institute of Electrical and Electronic Engineers

# Kapitel 3

# Grundlagen Garbage Collection

Schon seit den ersten Programmiersprachen ist das Aufräumen von verwendeten Ressourcen / Speicher ein wichtiges Thema. Im Unterschied zu den ersten Sprachen, bei denen das Memory Management in der Verantwortung des Entwicklers war (explizit), findet allerdings das Recycling von Memory bei Sprachen der dritter Generation automatisch statt und macht Operatoren wie "free" unwichtig. Bei Formen dieser automatischen Speicherverwaltung spricht man von Garbage Collection[1]. In den meisten neueren Laufzeitumgebungen spricht man zusätzlich von adaptivem Memory Management was bedeutet, dass Feedback der Laufzeitumgebung zur Anpassung der Garbage Collection Strategie verwendet wird. Probleme die nur beim expliziten Memory Management auftreten sind Dangling References/Pointers[2] und Space Leaks[3]. Trotzdem sind Memory Leaks auch bei automatischer Speicherverwaltung noch möglich, nämlich dann wenn Memory noch referenziert wird, auch wenn es schon nicht mehr gebraucht wird. Der folgende Abschnitt beschreibt die Grundlagen der Java Garbage Collection und geht im zweiten Teil auf die spezifischen Eigenheiten der JRockit Virtual Machine ein.

## 3.1 Funktionsweise

Alle Techniken der Garbage Collection zielen darauf ab, die "lebenden" von den "toten" Objekten im Speicher zu unterscheiden. Sprich, es müssen die Objekte gefunden werden, welche innerhalb der Software oder des Systems nicht mehr referenziert werden. Die aktuellen Strategien lassen sich laut[2, S. 77] in zwei

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

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

[3] Man spricht von Space Leaks, wenn Memory alloziert und nicht mehr freigegeben wurde, obwohl es nicht mehr gebraucht wird.[6]

unterschiedliche Familien aufteilen: “Reference Counting” und “Tracing techniques”.

### 3.1.1 Reference counting[2, S. 77]

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

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

### 3.1.2 Tracing techniques[2, S. 77]

Bei den Tracing Techniken werden von vor jeder Garbage Collection die Objekte gesucht, auf welche aktuell noch eine Referenz zeigt. Die anderen werden zur Garbage Collection freigegeben. Diese Art von Garbage Collection Algorithmen verwenden ein Set von Objekten, bestehend aus den Stacks und Registern der aktuellen Threads und globalen Daten wie “static final” variablen, als Startpunkt für die zu markierenden Objektgraphen.

## 3.2 Ziele der Garbage Collection[6, S. 4]

Garbage Collectors unterliegen grundsätzlich folgenden zwei Bedingungen:

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

Wünschenswert für Garbage Collection Algorithmen sind folgende Punkte:

- **Effizienz:** Die Anwendung soll vom laufenden Garbage Collector möglichst wenig mitkriegen: keine langen Pausen[4], möglichst wenig verwendete Ressourcen[5]
- **Fragmentierung:** Zwecks schneller Allokation von Speicher sollte der Speicher möglichst wenig

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

[5] Ressourcen der CPU sollen der Anwendung zur Verfügung gestellt werden und nicht für Garbage Collection verwendet werden.

## 3.3 Eingliederung von Garbage Collection Algorithmen[6, S. 5]

Bei der Selektion von Garbage Collection Algorithmen gibt es verschiedene entscheidende Kriterien:

### 3.3.1 Serielle versus Parallele Collection

Eine Multi-Core Maschine ist eine mit mindestens 2 Prozessorkernen. Sofern ein paralleler Algorithmus verwendet wird, besteht auf diesen auch die Möglichkeit, auch die Garbage Collection zu parallelisieren. Meistens bringt dies zwar einen kleinen Overhead mit sich, wirkt sich aber trotzdem in einer Verkürzung der Garbage Collection Zeit aus.

### 3.3.2 Konkurrierend versus Stop-the-World

Wenn aufgrund der Garbage Collection der Heap der Laufzeitumgebung blockiert (freeze) werden muss, führt das implizit zum Stopp (Stop-the-World) der Anwendung.

### 3.3.3 Kompaktierend, Kopierend

Fragmentierung ist ein eigentlich nicht wünschenswertes Resultat der Garbage Collection. Sie tritt dann auf, wen Algorithmen verwendet werden die den Heap im Anschluss an die Speicherfreigabe weder kompaktieren noch die lebenden Objekte in einen anderen Bereich aneinanderliegend kopieren.

## 3.4 Grundlegende Algorithmen

Der Prozess der Garbage Collection beginnt bei allen Algorithmen mit der Marking-Phase. Für jedes Objekt des Root Sets[6] werden rekursiv die transitiv abhängigen Objekte auf dem Heap bestimmt. Das hat zum Ziel, dass man aufgrund des Resultats die nicht mehr referenzierten (löschbaren) Objekte herausfinden kann.

### 3.4.1 Mark & Sweep Algorithmus[5]

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.

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

**Nachteil dieses Algorithmus**

Der zwar simple Mark & Seep Algorithmus hat den Nachteil, dass eine Freigabe dieser Speicherlöcher zu einer Fragmentierung des Speichers führt. Ein fragmentierter Speicher ist deshalb nicht wünschenswert, weil es für den Allokator immer schwieriger wird, die Speicherlöcher zu füllen. In einem solchen Fall kann es bis zu einer OutOfMemoryException führen, wenn zwar insgesamt genügend Memory frei ist, allerdings nicht genügend am Stück.

### 3.4.2 Mark & Copy Algorithmus[5]

Ein alternativer Algorithmus bei dem der Heap nach der Garbage Collection nicht fragmentiert ist, ist der Mark & Copy Algorithmus. Bei diesem wird der Heap in einen "from space" und einen "to space" unterteilt. Objekte die noch immer am Leben sind, werden im Anschluss an die Markierung vom "from space" in den "to space" kopiert. Die Allokation von Speicher ist im Anschluss mit grosser Effizienz möglich, weil der Heap nach der Garbage Collection nicht fragmentiert ist.

### 3.4.3 Mark & Compact Algorithmus[4]

In gewissen Situationen respektive für die unterschiedliche Lebensdauer der Objekte (siehe Generational Garbage Collection) macht es Sinn, unterschiedliche Algorithmen einzusetzen: Mark & Compact ist ein Algorithmus, bei welchem nach der Markierungs-Phase und der Elimination der "toten" Objekte eine Kompaktierung des Speichers stattfindet. Das hat ebenfalls den Vorteil, dass im Gegensatz zu Mark & Copy nicht der doppelte Speicher benötigt wird, ist allerdings relativ Ressourcen-intensiv.

### 3.4.4 Tri-Coloring Mark and Sweep

Eine Variante des Mark & Sweep Algorithmus die sich besser parallelisieren lässt ist der Tri-Coloring Mark & Sweep Algorithmus. [2, S. 79]. Im Gegensatz zur normalen Version des Mark & Sweep Algorithmus wird anstelle eines Mark-Flags ein ternärer Wert genommen der den Wert "weiss", "grau" und "schwarz" annehmen kann. Der Status der Objekte wird in drei Sets nachgeführt. Das Ziel des Algorithmus ist es, alle weissen Objekte zu finden. Schwarze Objekte sind die, die garantiert keine weisse 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.

Das Prinzip des Tri-Coloring Mark &Sweep Algorithmus ist folgende 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.

## 3.5 Generational Garbage Collection

In der Regel gibt es innerhalb einer Anwendung unterschiedliche Altersgruppen von Objekten und wir können die Objekte in Short Living, Medium Living und Long Living Objekte kategorisieren. Es gibt viele Objekte die nicht lange leben, zum Beispiel Objekte welche die Lebensdauer einer Methode haben, und wenige Objekte die lange Leben, wie zum Beispiel Daten in einem Applikations-Level Cache. Aus diesem Grund ist es oft Sinnvoll, die Garbage Collection an die entsprechende Lebensdauer der Objekte anzupassen.

Je nach Implementation wird der Heap in unterschiedliche Speicherbereiche aufgeteilt. Bei der Oracle HotSpot Virtual Machine sieht dies folgendermassen aus[3]:

- **Young Collection:** Die Young Collection ist nochmals unterteilt in Eden Space (Bereich für neue Objekte) und zwei Survivor Spaces ("from-spaceäls aktiver Bereich, "to-spaceäls der Bereich in den die Objekte nach einer Collection kopiert werden)
- **Old Collection:** In der Old Collection befinden sich die Objekte, die eine gewisse Anzahl an Young-Generation Collections (Minor Collections) überlebt haben und dann in den Bereich der alten Objekte befördert (Promotion) wurden.
- **PermGen:** Der PermGen Bereich ist keine Generation, sondern ein Non-Heap-Bereich, und wird von der VM für eigene Zwecke verwendet. Dieser Bereich ist eine Eigenheit der Oracle HotSpot Virtual Machine. Hier werden beispielsweise Class-Objekte inklusive Bytecode für alle geladenen Klassen und JIT-Informationen[7]. gespeichert.

Die JRockti Virtual Machine unterteilt die Bereiche folgendermassen:

- **Nursery[8]:** Die Nursery entspricht der Young Collection der HotSpot Virtual Machine und ist der Bereich der jungen Objekte. Bei JRockit ist es möglich, eine zusätzliche Keep-Area innerhalb der Nursery zu haben. Diese Keep-Area ist der Platz der Objekte mittlerer Lebensdauer. Ein Objekt wird also zuerst von der Nursery in die Keep-Area promoted - was

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

[8] Nursery bedeutet im übertragenen Sinn Kindergarten

den Vorteil hat, dass es wenn nicht mehr referenziert immer noch einer Young Collection unterliegt - erst dann gelangt es nach einer nochmaligen Young-Garbage-Collection in die Old Collection.

- **Old Generation:** Die Old Generation beinhaltet wie bei der HotSpot Virtual Machine die Objekte mit langer Lebensdauer.

Für weitere Details zu den JRockit Garbage Collection Eigenheiten siehe Kapitel Garbage Collection in der Oracle JRockit Virtual Machine.

# Kapitel 4

# Garbage Collection in der Oracle JRockit Virtual Machine

## 4.1 Grundlage

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

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

### 4.1.1 Old Collection

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

Die Verwendung von Thread-lokalem Speicher hat hinsichtlich den folgenden Punkten einen Vorteil:[2, S. 79]:

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

Bei der Verwendung der Concurrent Algorithmen - bei welcher die Garbage Collection neben der Anwendung statt findet - werden logischerweise auch nebenbei von der Applikation neue Objekte erzeugt. Diese neuen Objekte werden in einem sogenannten Live-Set getrackt.

### 4.1.2 Intergenerationale Referenzen

Die Idee der Generational Garbage Collection ist, dass zumindest während einer Young Collection nur ein Teilbereich des Heaps aufgeräumt wird. Während der Mark-Phase einer Young Collection werden also Referenzen in die Old Collection nicht verfolgt. Problematisch bei diesem Vorgehen ist, dass es eventuell Referenzen aus der Old Generation in die Young Generation gibt, aufgrund welcher man die Objekte im Young Space nicht löschen darft. Dieses Szenario kann auf zwei unterschiedliche Wege passieren:

- **Ein Objekt wird durch eine Promotion von der Young Generation in die Old Generation verschoben.** Diesen Vorgang merkt sich der Garbage Collector in einem **Remembered Set**.
- **Es findet eine Zuweisung der Referenz durch die Applikation statt.** Diese Problematik wird durch Write Barriers und der Aufteilung des Heaps in sogenannte Cards[1] Bei der Änderung einer Variablen respektive einer Referenz die dann vom Tenured Space auf den Young Space zeigen soll, wird auf der Card in welchem diese Änderung statt findet das "Dirty Bit" gesetzt. Objekte innerhalb von dirty Cards werden nach der Markierungsphase nochmals überprüft.

#### Concurrent Mark & Sweep

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

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

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

- **Final Marking (Nicht konkurrierend):** Änderungen im Heap während der Precleaning Phase werden nachgeführt, markiert.

Die Sweep Phase findet ebenfalls konkurrierend zur Applikation statt. Im Gegensatz zur HotSpot VM findet es in zwei separaten Schritten statt. Als erstes wird die erste Hälfte des Heaps von toten Objekten befreit. Während dieser Phase können Threads Speicher in der zwieten Hälfte des Heaps allozieren. Nach einer kurzen Synchronisationspause findet das Sweeping auf dem zweiten Teil des Heaps statt, wieder gefolgt von einer Synchronisationspause.

#### Parallel Mark & Sweep

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

#### Kompaktierung

Aufgrund der im Speicher stattfindenden Fragmentierung wird bei jeder Old Collection eine Kompaktierung des Speichers durchgeführt. Zu Beginn der Sweep-Phase oder während werden dazu Objekte auf dem Heap umher kopiert, so dass danach wieder grössere Speicherbereiche zur Allokation frei stehen.

## 4.2 Übersicht der Garbage Collection Algorithmen auf der JRockit Virtual Machine

Die Entscheidung der Garbage Collection Strategie hängt von den Anforderungen der Applikation ab. Auf der JRockit VM gibt es drei verschiedene Modi:

- **Durchsatz (throughput):** Optimiert die Garbage Collection hinsichtlich möglichst grossem Durchsatz. Hier handelt es sich um Applikationen wie Batch-Systeme, etc. Um auf der Virtual Machine dieses Ziel zu erreichen, werden Parallele Algorithmen verwendet. Parallele Algorithmen laufen nicht-konkurrierend mit dem System in mehreren parallelen Threads. Das führt zwar zu kurzen Pausenzeiten für die Applikation, allerdings steht in der restlichen Zeit sehr viel Prozessor-Zeit zur Verfügung.
- **Pausenzeit (pause time):** Die Optimierung der Pausenzeit führt dazu, dass der Benutzer der Virtual Maschine keine langen Pausenzeiten bemerkt. Das ist insbesondere für Client-Server Applikationen notwendig. Wenn die JRockit Laufzeitumgebung auf möglichst kurze Pausenzeiten eingestellt ist, wird ein konkurrierender Garbage Collection Algorithmus verwendet, bei welchem es nur sehr wenige Stop-the-World Phasen gibt.
- **Determinismus (deterministic):** Optimiert den Garbage Collector auf sehr kurze und deterministische Garbage Collection Pausen.

| **Alias** | **Aktivierung** | **Beschreibung** | **Pause** | **Durchs.** | **Heap** | **Mark** | **Sweep** |
|---|---|---|---|---|---|---|---|
| singlecon, singleconcon | | Garbage Collection findet konkurrierend statt. Der Heap besteht nur aus einer Generation. | ++ | – | single | konk. | konk |
| gencon, genconcon | -Xgc:gencon -Xgc:genconcon | Neue Objekte kommen in die Nursery. Ist diese voll, werden die Lebenden in die Old Generation verschoben. Die Old Collection findet konkurrierend statt. | ++ | – | gen | konk. | konk. |
| singlepar, singleparpar | | Wenn der Heap voll ist, wird bei gestoppter Applikation eine Garbage Collection des gesamten Heaps gemacht. | – | ++ | single | parallel | parallel |
| genpar, genparpar | -Xgc:genpar | Objekte werden in der Nursery angelegt. Ist diese voll, wird bei gestoppter Applikation parallel eine Young Collection durchgeführt. Bei vollem Heap gibs eine volle Garbage Collection. | – | ++ | gen | parallel | parallel |
| genconpar | -Xgc:genconpar | | + | + | gen | konk. | parallel |
| genparcon | -Xgc:genparcon | | + | + | gen | parallel | konk. |
| singleconpar | -Xgc:singleconpar | | + | + | single | konk. | parallel |
| singleparcon | -Xgc:singleparcon | | + | + | single | parallel | konk. |

Tabelle 4.1: Übersicht der Garbage Collection Algorithmen

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

Tabelle 4.2: Übersicht der Garbage Collection Modi

## 4.3 Garbage Collection Log Dateien

Die Auswertung der Garbage Collection findet auf Basis der Garbage Collection Log Dateien statt. Das Format dieser Log Dateien hängt mitunter auch von den aktivierten Log-Modulen der JRockit Virtual Machine ab. Der Aufbau der Log Dateien und die in der Analyse verwendeten Daten werden in diesem Abschnitt genauer beleuchtet.

### 4.3.1 Aktivierung Log Ausgaben

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

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

Listing 4.1: Format Aktivierung Log Modul

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

```
1 -Xverbose:memory
```

Listing 4.2: Garbage Collection Log (Info)

Die Umleitung der Ausgaben in eine separate Log-Datei kann mit dem Flag -Xverboselog:¡Datei-Namen¿ eingestellt werden:

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

Listing 4.3: Garbage Collection Log (Info) - Umleitung in gc.log

Zusätzlich können mit der Angabe des Log-Levels die Ausgaben verfeinert werden:

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

Listing 4.4: Garbage Collection Log (Debug) - Umleitung in gc.log

Die ersten paar Zeilen der Ausgaben des Memory Moduls im Log Level “info” ohne explizite Angabe der Garbage Collection Strategie [2] sehen folgendermassen aus:

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

Listing 4.5: Garbage Collection Log (Debug) - Umleitung in gc.log

[2] JRockit Virtual Machine verwendet damit die Default-Einstellung: Optimierung auf maximalen Durchsatz

### 4.3.2 Log Module

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

| **Modul** | **Beschreibung** | **Relevanz** |
|---|---|---|
| alloc | Informationen betreffend Speicher Allokation und "Out-of-Memory" | niedrig |
| compaction | zeigt abhängig vom Garbage Collection Algorithmus Informationen zur Kompaktierung | niedrig |
| gcheuristic | zeigt Informationen und Entscheidungen zur Garbage Collection Heuristik | mittel |
| gcpause | zeigt aktuell welche Pausen durch die Garbage Collection entstanden sind 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. | **hoch** |
| memdbg | zeit Informationen die für das Debugging von Themen im Bereich des Arbeitsspeichers relevant sein können | **hoch** |
| systemgc | zeigt Garbage Collections die durch System.gc() gestartet wurden. | niedrig |

Tabelle 4.3: Beschreibung der verschiedenen relevanten Log Modulen

Die folgenden Log Level können aktiviert werden: quiet, error, warn, info, debug, trace. Für mehr Informationen siehe [7].

# Kapitel 5

# Anforderungsanalyse

## 5.1 Einleitung

### 5.1.1 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.

### 5.1.2 Systemumfang

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

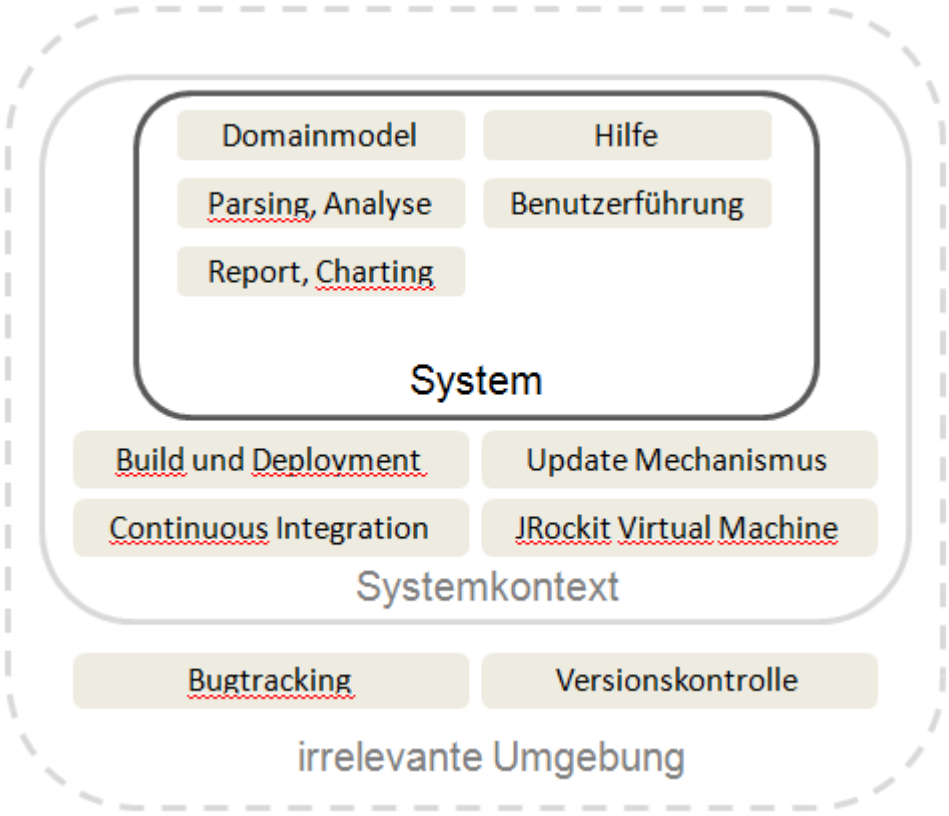



Abbildung 5.1: System und Systemkontext

**System**

Das System besteht aus den Teilen Datenmodell, Domäne, Log Parsing, Garbage Collection Analyse, Report, Charting, Hilfesystem[1], Benutzerführung.

**Systemkontext**

Zum Systemkontext gehören folgende nicht veränderbare Komponenten:

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

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

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

**Irrelevante Umgebung**

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

### 5.1.3 Stakeholder

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

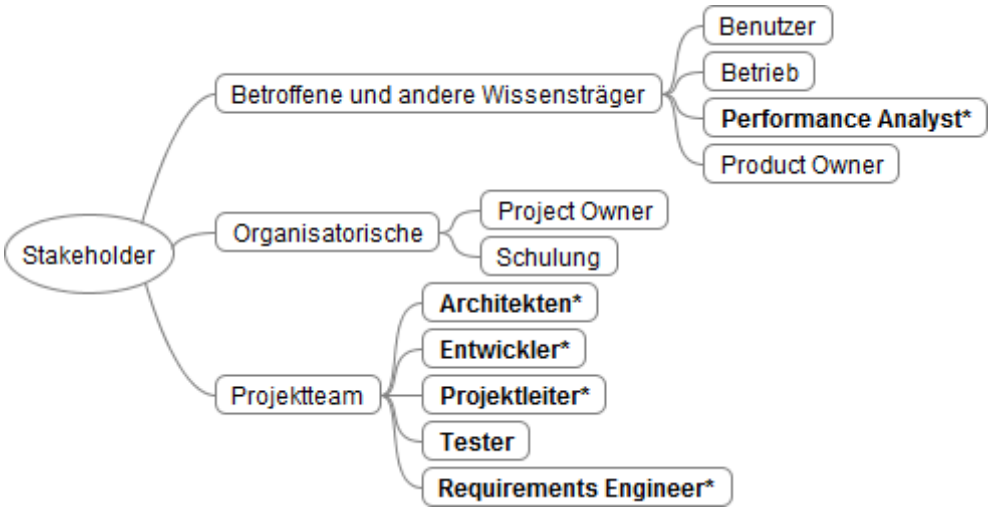



Abbildung 5.2: Übersicht der Stakeholder

### 5.1.4 Glossar

Das Glossar befindet sich auf Seite 70.

### 5.1.5 Referenzen

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

### 5.1.6 Übersicht

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

## 5.2 Allgemeine Übersicht

### 5.2.1 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.

### 5.2.2 Nutzer und Zielgruppen

#### Performance Engineers

Normalerweise Software Entwickler die viel Erfahrung, Wissen, Fähigkeiten haben und über die entsprechenden Werkzeuge verfügen, um die Ursachen für Performance-Probleme zu finden. Dabei handelt es sich nicht nur um Wissen im Bereich der Softwareentwicklung, sondern auch im Bereich des Servers (Speicher Management, etc.), und des Betriebssystems (I/O[2], etc.). Durch ihr breites Wissen sind sie mit der Unterstützung von Charts, Statistiken und Reports oft sehr schnell in der Lage, Ursache eines Performance Engpasses 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 Performance Analyse. Als gut ausgebildete Ingenieure sind sie aber mit Hilfe von Werkzeugen und Dokumentation in der Lage, Performance Problemen innert nützlicher First auf den Grund zu gehen.

[2] IO steht für Input/Output und betrifft Netzwerk wie auch Harddisk

### 5.2.3 Randbedingungen

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

### 5.2.4 Annahmen

## 5.3 Use Cases

### 5.3.1 Systemfunktionalität

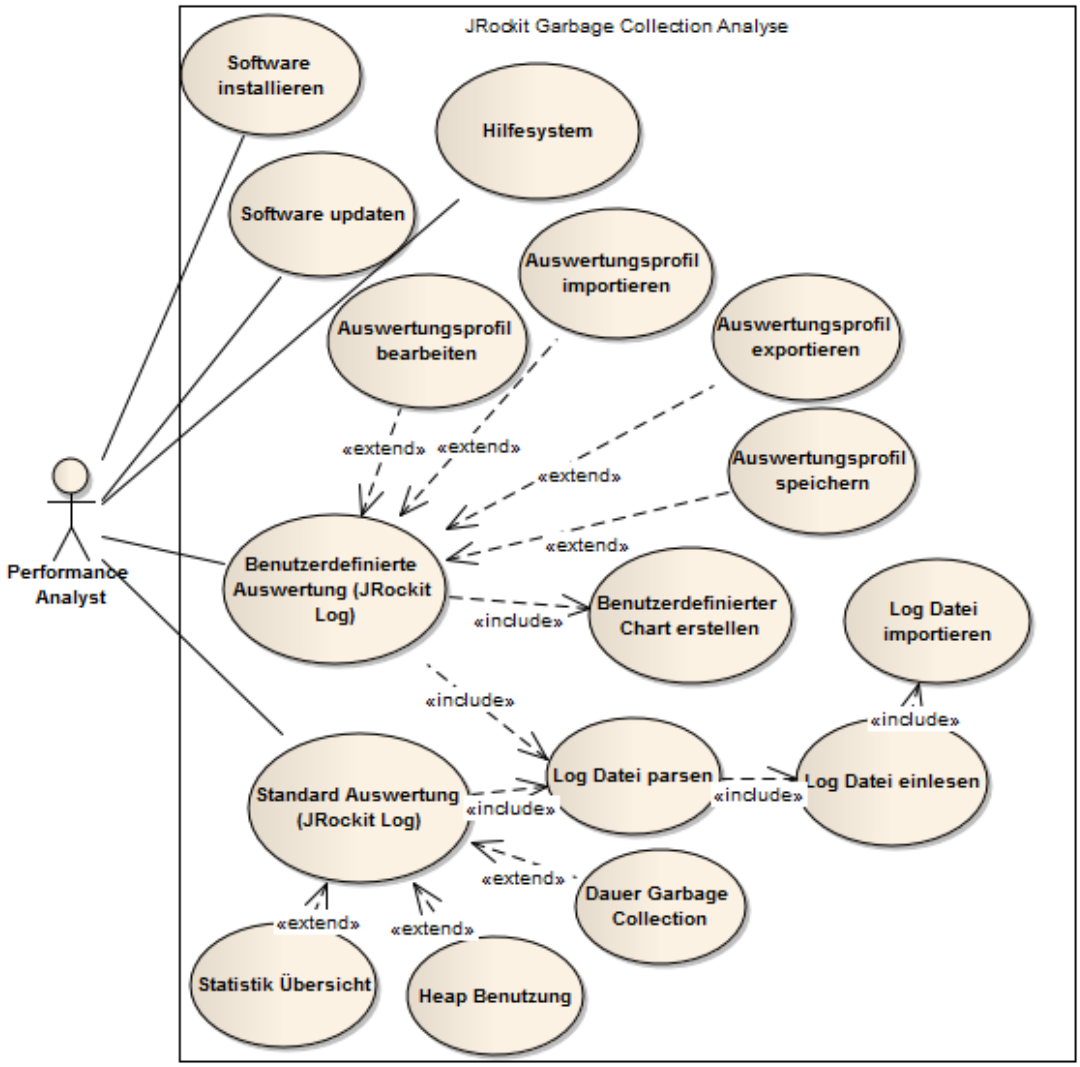



Abbildung 5.3: Systemfunktionalität als Use-Case-Diagramm

Die ersten beiden Teile dieses Abschnitts definieren die Use Cases des Systems nach Rolle sortiert. Als Basis dient die Schablone zur Use-Case-Spezifikation in

[8, S. 78-79].

### 5.3.2 Basissoftware

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-01 |
| Name | Software installieren |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: gross<br>Technologisches Risiko: mittel |
| Kritikalität | gross |
| Quelle | Raffael Schmid |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Der Benutzer kann die Software in seiner Entwicklungsumgebung installieren. |
| Akteure | Anwender, Entwicklungsumgebung |
| Auslösendes Ereignis | Anwender möchte eine Garbage Collection Log Datei analysieren. |
| Vorbedingung | Richtige Entwicklungsumgebung ist bereits ohne Analysesoftware installiert. |
| Nachbedingung | Es sind keine Fehler aufgetreten. |
| Ergebnis | Entwicklungsumgebung ist bereit für Garbage Collection Auswertungen. |
| Hauptszenario | 1. Anwender Startet Entwicklungsumgebung<br>2. Anwender gibt Update-Seite an<br>3. Anwender selektiert zu installierendes Softwarepaket<br>4. Softwarepaket wird installiert |
| Alternativszenarien | - |
| Ausnahmeszenarien | - |
| Qualitäten | - |

Tabelle 5.1: Use-Case: Software installieren

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-02 |
| Name | Software updaten |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: gross<br>Technologisches Risiko: mittel |
| Kritikalität | Mittel |

| | |
|---|---|
| Quelle | Raffael Schmid |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Der Benutzer kann die Software updaten. |
| Akteure | Anwender, Update-Server |
| Auslösendes Ereignis | Anwender hat die Software bereits zu einem früheren Zeitpunkt installiert. Sofern ein neues Update vorhanden ist, möchte er dieses installieren. |
| Vorbedingung | Richtige Entwicklungsumgebung und Software wurden bereits in einer früheren Version installiert. |
| Nachbedingung | Es sind keine Fehler aufgetreten. |
| Ergebnis | Entwicklungsumgebung und Analysesoftware sind auf dem neusten Stand für Garbage Collection Auswertungen. |
| Hauptszenario | 1. Anwender Startet Entwicklungsumgebung<br>2. Anwender sucht und findet Updates für die Analysesoftware<br>3. Anwender selektiert eines oder mehrere dieser Softwarepakete<br>4. Software wird aktualisiert |
| Alternativszenarien | - |
| Ausnahmeszenarien | - |
| Qualitäten | - |

Tabelle 5.2: Use-Case: Software updaten

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-03 |
| Name | Garbage Collection Log Datei importieren |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: gross<br>Technologisches Risiko: gering |
| Kritikalität | gross |
| Quelle | Raffael Schmid |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Der Benutzer importiert die sich auf dem Dateisystem befindende Log-Datei. |
| Akteure | Anwender, Log Datei Importer |
| Auslösendes Ereignis | Anwender startet Garbage Collection Log Analyse |
| Vorbedingung | Log Datei 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 Log-Datei ist in der Ansicht “Log-Dateien” ersichtlich und kann von da im Analysefenster geöffnet werden. |
| Hauptszenario | 1. Anwender öffnet Import-Dialog<br>2. Anwender navigiert zum Ordner<br>3. Anwender selektiert Datei(en) und importiert diese |
| Alternativszenarien | - |
| Ausnahmeszenarien | - |
| Qualitäten | - |
| Erweiterungen | UC-03.1 |

Tabelle 5.3: Use-Case: Garbage Collection Log Datei importieren

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

Tabelle 5.4: Use-Case: Speichern der importierten Dateien

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

Tabelle 5.5: Use-Case: Garbage Collection Log Datei einlesen

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

| | |
|---|---|
| Nachbedingung | Kein Fehler ist aufgetreten. |
| Ergebnis | Domänenmodell ist fertig abgefüllt. |
| Hauptszenario | Jede einzelne Zeile in der Log-Datei wird analysiert und wenn nötig werden die wichtigen Daten extrahiert. Die Daten werden in das für diese Log Datei angelegte Domänenmodell geschrieben. |
| Alternativszenarien | - |
| Ausnahmeszenarien | - |
| Qualitäten | Performance (QRQ-F-04), Korrektheit (QRQ-F-05) |

Tabelle 5.6: Use-Case: Garbage Collection Log Datei importieren

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-06 |
| Name | Standardauswertung anzeigen |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: gross<br>Technologisches Risiko: gross |
| Kritikalität | gross |
| Quelle | Raffael Schmid |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Für eine schnelle Übersicht steht eine Standard-Auswertung zur Verfügung. Dieser soll eine kurze Übersicht über die Garbage Collection geben und beinhaltet zwei Charts (Heap Benutzung, Dauer Garbage Collection). |
| Akteure | Anwender, Log Datei Analyzer, Report Engine |
| Auslösendes Ereignis | Die Applikation hat die Datei fertig eingelesen und geparst. |
| Vorbedingung | Die Log Datei wurde ohne Fehler eingelesen und befindet sich im strukturierten Format im Arbeitsspeicher. |
| Nachbedingung | - |
| Ergebnis | Dem Benutzer wird ein Analsye-Screen angezeigt. |
| Hauptszenario | 1. Applikation hat die Log-Datei fertig importiert.<br>2. Dem Benutzer wird ein Screen mit verschiedenen Tabs angezeigt. Auf jedem Tab wird dem Benutzer eine unterschiedliche Sicht auf die Daten gezeigt. |
| Alternativszenarien | Benutzerdefinierte Auswertung |
| Ausnahmeszenarien | - |
| Qualitäten | Korrektheit (QRQ-F-05) |
| Erweiterungen | UC-06.1, UC-06.2, UC-06.3, UC-06.4 |

Tabelle 5.7: Use-Case: Standardausertung anzeigen

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-06.1 |
| Name | Anzeige Statistik Übersicht |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: gross<br>Technologisches Risiko: gross |
| Kritikalität | gross |
| Quelle | Raffael Schmid |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Der Analyse-Screen wurde geöffnet, dem Benutzer zeigen sich unterschiedliche Tabs. Auf dem ersten befinden sich verschiedene statistische Auswertungen der Log Datei:<br>• Übersicht und Grösse der verschiedenen Bereiche auf dem Heap: Initiale Grösse, endgültige Grösse<br>• Aktivitäten des Garbage Collectors: Anzahl Young Collections, Anzahl Old Collections<br>• Anzahl Garbage Collector Events (Bsp: Change Garbage Collector Strategy, etc.)<br>• Garbage Collection Zeiten (Total, Durchschnittliche, Zeit in Old Generation Garbage Collection, Prozentuale Zeit in Old Generation Garbage Collection) |
| Qualitäten | Korrektheit (QRQ-F-05) |

Tabelle 5.8: Use-Case: Anzeige Statistik Übersicht

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-06.2 |
| Name | Anzeige Heap Benutzung |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: gross<br>Technologisches Risiko: gross |
| Kritikalität | gross |
| Quelle | Raffael Schmid |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Die Heap Usage (Heap Benutzung) zeigt dem Benutzer anhand einer Grafik, zu welchem Zeitpunkt wieviel Speicher des Heaps verwendet wurde. Zusätzlich werden die Zeitpunkte inklusive entsprechendem Typ (Old- / Young-Collection) der Garbage Collection angezeigt. |
| Qualitäten | Korrektheit (QRQ-F-05) |

Tabelle 5.9: Use-Case: Anzeige Heap Benutzung

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

Tabelle 5.10: Use-Case: Anzeige Dauer Garbage Collection

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-A07 |
| Name | Profil (benutzerdefinierte Auswertung) erstellen |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: niedrig<br>Technologisches Risiko: niedrig |
| Kritikalität | Niedrig |
| Quelle | Raffael Schmid |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Der Benutzer kann ein eigenes Profil erstellen. Das Profil dient dazu, um darauf eigene Charts hinzuzufügen. siehe UC-07.1. |
| Akteure | Anwender, Log Datei Analyzer, Report Engine |
| Auslösendes Ereignis | Die Applikation hat die Datei fertig eingelesen und geparst. |
| Vorbedingung | Die Log Datei wurde ohne Fehler eingelesen und befindet sich im strukturierten Format im Arbeitsspeicher. |
| Nachbedingung | - |
| Ergebnis | Dem Benutzer wird ein Analsye-Screen angezeigt. |
| Hauptszenario | Unabhängig vom Zyklus der Garbage Collection Analyse kann der Benutzer ein eigenes Profil erstellen. Ein Profil besteht initial aus einem Übersichtsfenster der Garbage Collection und kann um eine vielzahl an Charts erweitert werden. |
| Alternativszenarien | - |

| | |
|---|---|
| Ausnahmeszenarien | - |
| Qualitäten | Korrektheit (QRQ-F-05) |
| Erweiterungen | UC-A07.1, UC-A07.2, UC-A07.3 |

Tabelle 5.11: Use-Case: Profil (benutzerdefinierte Auswertung) erstellen

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-07.1 |
| Name | Chart definieren |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: niedrig<br>Technologisches Risiko: niedrig |
| Kritikalität | Niedrig |
| Quelle | Adrian Hummel (Performance Engineer) |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Der Benutzer erstellt auf dem Profil ein neues Chart und die sich darauf befindenden Serien[3]. |
| Qualitäten | - |

Tabelle 5.12: Use-Case: Chart definieren

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-07.2 |
| Name | Auswertungsprofil speichern |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: niedrig<br>Technologisches Risiko: niedrig |
| Kritikalität | Niedrig |
| Quelle | Adrian Hummel (Performance Engineer) |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Profile können gespeichert werden und bleiben über einzelne Sitzungen bestehen, ausser der Benutzer löscht es explizit. |
| Qualitäten | - |

Tabelle 5.13: Use-Case: Profil speichern, löschen

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-07.3 |
| Name | Auswertungsprofil exportieren |
| Autoren | Raffael Schmid |

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

| | |
|---|---|
| Priorität | Wichtigkeit für Systemerfolg: niedrig<br>Technologisches Risiko: niedrig |
| Kritikalität | Niedrig |
| Quelle | Adrian Hummel (Performance Engineer) |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Die vom Anwender definierten Profile können in eine Datei exportiert werden. |
| Qualitäten | - |

Tabelle 5.14: Use-Case: Auswertungsprofil exportieren

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-07.4 |
| Name | Auswertungsprofil importieren |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: niedrig<br>Technologisches Risiko: niedrig |
| Kritikalität | Niedrig |
| Quelle | Adrian Hummel (Performance Engineer) |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Exportierte Profile können via ein Kanal ausgetauscht und in der anderen Entwicklungsumgebung importiert werden. |
| Qualitäten | - |

Tabelle 5.15: Use-Case: Auswertungsprofil importieren

| **Abschnitt** | **Inhalt / Erläuterung** |
|---|---|
| Bezeichner | UC-8 |
| Name | Hilfesystem |
| Autoren | Raffael Schmid |
| Priorität | Wichtigkeit für Systemerfolg: niedrig<br>Technologisches Risiko: mittel |
| Kritikalität | Mittel |
| Quelle | Raffael Schmid |
| Verantwortlicher | Raffael Schmid |
| Kurzbeschreibung | Dem Benutzer steht eine eine indexbasierte[4] und eine kontextsensitive[5] Hilfe zur Verfügung. |
| Akteure | Anwender |
| Auslösendes Ereignis | Anwender hat Plugin installiert, weiss nicht wie eine Analyse gestartet werden kann. |

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

[5]Hilfe zur aktuellen View oder Aktion des Benutzers

| | |
|---|---|
| Vorbedingung | Richtige Entwicklungsumgebung und Garbage Collection Log Analysis Plugin ist bereits in einer früheren Version installiert. |
| Nachbedingung | - |
| Ergebnis | Anwender kennt Software |
| Hauptszenario | • **Indexbasierte Hilfe:** Der Benutzer kennt sich im Thema Garbage Collection und auf der Analyse-Software noch nicht aus. Er holt sich Hilfe über die indexbasierte Hilfe.<br>• **Kontextsensitive Hilfe:** Der Benutzer befindet sich in einem Fenster oder möchte eine Aktion ausführen (Context), das Hilfesystem zeigt ihm dazu die notwendingen Informationen. |
| Alternativszenarien | - |
| Ausnahmeszenarien | - |
| Qualitäten | QRQ-S-03 (Internationalisierung) |

Tabelle 5.16: Use-Case: Hilfesystem

## 5.4 Funktionale Anforderungen

Aus den im Abschnitt Use Cases definierten Anforderungen ergibt sich die abschliessende Liste der funktionalen Anforderungen:

| Identifik. | Vers. | Titel | Beschreibung | Use Case | Abnahmekriter. | Prio. |
|---|---|---|---|---|---|---|
| FRQ-01 | 1.0 | Installation | Software muss als Erweiterung in der Entwicklungsumgebung[6] installiert werden können. | UC-01 | Entwickler mit durchschnittlichen Kenntnissen benötigen für die Installation in eine bestehende Entwicklungsumgebung dauert weniger als 5 Minuten. | gross |
| FRQ-02 | 1.0 | Updaten | Die Software kann mit geringem Aufwand aktualisiert werden. | UC-02 | Entwickler mit durchschnittlichen Kenntnissen für den Update weniger als 3 Minuten. | mittel |
| FRQ-03 | 1.0 | Garbage Collection Log Datei importieren | Log Dateien können importiert werden und werden anschliessend in einem Fenster dargestellt. | UC-03 | - | gross |
| FRQ-03.1 | 1.0 | Importierte Dateien speichern | Informationen über importierte Log-Dateien werden gespeichert und bleiben über die Zeit der Benutzersession bestehen. | UC-03.1 | - | gross |

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

| | | | | | | |
|---|---|---|---|---|---|---|
| FRQ-04 | 1.0 | Garbage Collection Log Datei einlesen | Geöffnete Garbage Collection Log Datein werden ins Memory gelesen. | UC-04 | Der Einleseprozess bei einer Datei mit 100000 Zeilen dauert weniger als 2 Sekunden. | gross |
| FRQ-05 | 1.0 | Garbage Collection Log Datei parsen | Die eingelesenen JRockit Garbage Collection Log Datei wird geparst. Aus den Daten wird ein Domänenmodell aufgebaut. | UC-05 | Das Parsen einer Log Datei mit 100000 Zeilen dauert nicht länger als 8 Sekunden. | gross |
| FRQ-06 | 1.0 | Standardauswertung anzeigen | Der Benutzer kann eine vordefinierte Anzeige öffnen. | UC-06 | - | gross |
| FRQ-06.1 | 1.0 | Anzeige Übersicht Garbage Collection | Der erste Tab des Analysefensters zeigt die aggregierten Daten der Garbage Collection. | UC-06.1 | Die Genauigkeit der berechneten und angezeigten Werte ist mindestens ein Zehntel (0.1). | gross |
| FRQ-06.2 | 1.0 | Anzeige Heap Benutzung | Der zweite Tab des Analysefensters zeigt den Speicherbedarf über die Zeit. | UC-06.2 | Die Genauigkeit der berechneten und angezeigten Werte ist mindestens ein Zehntel (0.1). | gross |
| FRQ-06.3 | 1.0 | Anzeige Dauer Garbage Collection | Der dritte Tab des Analysefensters zeigt die Dauer der einzelnen Garbage Collections über die Zeit. | UC-06.3 | Die Genauigkeit der berechneten und angezeigten Werte ist mindestens ein Zehntel (0.1). | mittel |
| FRQ-07 | 1.0 | Profil (Benutzerdefinierte Auswertung) erstellen | Benutzer kann Profil erstellen, um anschliessend darin benutzerdefinierte Charts zu erstellen. | UC-B07 | - | klein |

| | | | | | | |
|---|---|---|---|---|---|---|
| FRQ-07.1 | 1.0 | Chart definieren für Profil | Für ein erstelltes Profil kann der Benutzer aus den Log-Daten ein eigenes Chart definieren. | UC-B07.1 | - | klein |
| FRQ-07.2 | 1.0 | Profil speichern | Definiertes Profil wird automatisch gespeichert und bleibt über die Dauer der Sitzung bestehen. | UC-07.2 | - | klein |
| FRQ-07.3 | 1.0 | Profil exportieren | Profil kann in Textdatei exportiert werden. | UC-07.3 | - | klein |
| FRQ-07.4 | 1.0 | Profil importieren | Profil kann aus Textdatei importiert werden. | UC-07.4 | - | klein |
| FRQ-08 | 1.0 | Hilfesystem | Dem Benutzer werden eine indexbasierte und eine kontextsensitive Hilfe zur Verfügung gestellt. | UC-08 | Die Hilfe ist in Deutsch und Englisch verfügbar. | klein |

Tabelle 5.17: Funktionale Anforderungen

## 5.5 Qualitätsanforderungen

### 5.5.1 Software

| Identifik. | Vers. | Titel | Beschreibung | Abnahmekriter. | Prio. |
|---|---|---|---|---|---|
| QRQ-S-01 | 1.0 | Erweiterbarkeit | Nebst der Analyse von Garbage Collection Logs der JRockit Virtual Machine sollen später auch andere Formate unterstützt sein. | Erweiterung um ein weiteres Log-Format soll den Aufwand von 5 PT[7] nicht überschreiten. | mittel |
| QRQ-S-02 | 1.0 | Testabdeckung | Um den langfristigen Erfolg dieser Software zu gewährleisten muss eine entsprechende Testabdeckung vorhanden sein - dies um insbesondere die Regression zu vermeiden. | 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 ermöglicht. | - | klein |
| QRQ-F-04 | 1.0 | Performance (Antwortzeiten) | Auch grosse Datenmengen (Log Dateien) müssen schnell eingelesen werden. | Der Import einer Log-Datei von 100000 Zeilen dauert kürzer als 10 Sekunden. | mittel |
| QRQ -F-05 | 1.0 | Korrektheit (angezeigten Werte) | Die berechneten und angezeigten Werte sind exakt. | berechnete und angezeigte Werte haben eine Genauigkeit auf mindestens einem Zehntel (0.1). | gross |

### 5.5.2 Basisframework

[7] Personentage

| Identifik. | Vers. | Titel | Beschreibung | Abnahmekriter. | Prio. |
|---|---|---|---|---|---|
| QRQ-F-01 | 1.0 | Verbreitung | Die Software wird als Erweiterung für eine Entwicklungsumgebung bereitgestellt. Auch weil die Entwickler-Community auf diesen Plattformen am grössten ist, spielt die Verbreitung des verwendeten Frameworks eine grosse Rolle. | - | gross |
| QRQ-F-02 | 1.0 | Plattform-unabhängig | Die Software soll auf den gängigsten Betriebssystemen Windows und Apple OSX laufen. | Das Framework läuft auf den Plattformen Windows und Mac OSX. | gross |
| QRQ-F-03 | 1.0 | Lokalisation | Framework muss Unterstützung für Lokalisation bereitstellen. | Das Framework bietet Unterstützung für die Mehrsprachigkeit. | klein |
| QRQ-F-04 | 1.0 | Modularisierung | Das Framework muss Unterstützung für Modularisierung bieten, damit die Software in unterschiedliche Komponenten aufgeteilt werden kann (siehe Erweiterbarkeit QRQ-S-01). | Framework bietet Unterstützung für Modularisierung. | mittel |
| FRQ-F-05 | 1.0 | Offline Betriebsmodus | Der Anwender soll die Software auch im Offline-Modus[8] benutzen können. | Eigenständige Software, keine Web Applikation | gross |
| FRQ-F-06 | 1.0 | Installation als Erweiterung | Die Software wird als Erweiterung in einer Entwicklungsumgebung installiert. | - | gross |

Tabelle 5.19: Qualitätsanforderungen Basisframework

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

# Kapitel 6

# Auswahl Frameworks und Komponenten

## 6.1 Rich Client Frameworks

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

### 6.1.1 Übersicht

#### Eclipse RCP

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

Auf die Version 3.0 wurde die Architektur von Eclipse relativ stark umgestellt und modularisiert. Nun handelt es sich um einen relativ kleinen Kern der Applikation, der die eigentliche Funktionalität der Applikation als Plugins lädt. Diese Funktionalität basiert auf Eclipse Equinox, einer Implementation der OSGi Spezifikation. Die grafischen Benutzeroberflächen sind in SWT implementiert. Eclipse ist für 14 verschiedene Systeme und Architekturen bereitgestellt

und gilt somit als plattformunabhängig [11].

Die Plattform kann nun auch als Framework zur Entwicklung von Desktop Applikationen oder Plugins für die Entwicklungsumgebung entwickeln verwendet werden. Bei eigenen Desktop-Anwendungen spricht man auch von Eclipse-RCP Anwendungen.

#### Netbeans RCP

Wie bei Eclipse RCP handelt es sich bei Netbeans RCP ebenfalls um ein Framework zur Entwicklung von Desktop Anwendungen. Dem Entwickler wird ein API für typische Anforderungen in diesem Bereich zur Verfügung gestellt. Der Kern der Netbeans Plattform besteht ebenfalls aus einem Modul-Loader und im Bereich der grafischen Benutzeroberfläche wird Swing verwendet.

### 6.1.2 Auswertung Rich Client Frameworks

Die erste und zweite Spalte zeigen die Anforderungen aus Abschnitt Basisframework mit der jeweiligen Gewichtung. Die weiteren Spalten zeigen die erreichte Punktzahl zusammen mit dem aus Punktzahl und Gewicht errechnete Produkt.

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

Tabelle 6.1: Auswertung Rich Client Frameworks

#### Bewertung

Die Bewertung der einzelnen Anforderungen wurde geschätzt und wird mittels folgender Tabelle in eine Zahl umgewandelt:

| **Bewertung** | sehr schlecht | schlecht | mittel | gross | sehr gross |
|---|---|---|---|---|---|
| **Punktzahl** | 1 | 2 | 3 | 4 | 5 |

Tabelle 6.2: Schema Vergabe der Punkte

**Gewichtung**

Die Gewichtung wird aus der Priorität der einzelnen Anforderung sowie der nachfolgenden Tabelle in eine Zahl umgewandelt:

| **Priorität** | sehr klein | klein | mittel | gross | sehr gross |
|---|---|---|---|---|---|
| **Gewicht** | 1 | 2 | 3 | 4 | 5 |

Tabelle 6.3: Schema Gewichtung der Prioritäten

### 6.1.3 Entscheid

Ausschlaggebend für die Wahl der Rich Client Plattform ist die Verbreitung. Trotzdem dass die Technologie und die Architektur der drei Frameworks teilweise grosse Unterschiede aufweisen, unterscheiden sie sich im Funktionsumfang nur unwesentlich.

# Kapitel 7

# Konzept

## 7.1 Allgemein

### 7.1.1 Zweck

### 7.1.2 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.

### 7.1.3 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 Entwicklern beteiligen. Der Entwicklungsaufwand kann skaliert werden.
- Jedermann kann Erweiterungen entwickeln oder Fehler beheben.

Bei der Wahl der Lizenz muss gleichzeitig in Betracht gezogen werden, dass die Lizenzen der verwendeten Bibliotheken ebenfalls eingehalten werden. Übersicht der verwendeten Bibliotheken:

Tabelle 7.1: Verwendete Bibliotheken und deren Lizenzen

| Bibliothek | Beschreibung | Lizenz |
|---|---|---|

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

| | | |
|---|---|---|
| Eclipse Framework | Framework zur Erstellung von Destkop-Anwendungen. | Eclipse Public License |
| JFreeChart | Wird für im Bereich des Reportings verwendet um Graphiken und Charts anzuzeigen. | Lesser General Public License[3] |

## 7.2 Eclipse Rich Client Framework

### 7.2.1 Speicherung des Zustands einer View

Die Speicherung des Zustands einer View kann implementiert werden, indem die Methode *saveState(memento:IMemento)* überschrieben wird. *IMemento* ist eine Eclipse-Klasse und gleichzeitig die Abstraktion eines Mementos. Memento ist ein Design Pattern und wurde durch die Gang of Four[4] in [1, S. 283] zum ersten Mal definiert. Mementos dienen zur Serialisierung von Objekten und haben den Vorteil, dass auch Klassen aus einem Memento einer anderen Version deserialisiert werden können. Das Eclipse-Framework persistiert die Zustände von Views mittels eines XMLMemento entsprechend im XML-Format ab. Die Deserialisierung erreicht man mit dem Überschreiben der Methode *init(site:IViewSite, memento:IMemento)*.

## 7.3 Architektur

### 7.3.1 Problemstellung

Aufgrund der Anforderung QRQ-S-01 (Erweiterbarkeit) muss die Analyse-Software auch hinsichtlich anderer Log-Formate erweiterbar sein. Die Applikation wird also nicht zwingendermassen mit der Erweiterung für JRockit Log Dateien 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 soll in zwei Komponenten aufgeteilt werden:
  - Basissoftware
  - Erweiterung JRockit

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

[3] http://www.gnu.org/licenses/lgpl.html
[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

- Die Architektur der Applikation muss es zulassen, dass zu einem späteren Zeitpunkt andere Log-Formate als zusätzliche Erweiterungen hinzugefügt werden.
- Die Basissoftware stellt Extension-Points[5] bereit, über welche sich die Erweiterungen registrieren.

### 7.3.2 Übersicht

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

- Basissoftware (Core Feature)
  - Update der Software
  - Garbage Collection Log importieren
  - Garbage Collection Log einlesen
  - Profil erstellen, speichern, exportieren, importieren
  - Hilfesystem
- JRockit Extension (JRockit Extension Feature)
  - Garbage Collection Log parsen
  - Standardauswertung: Heap, Dauer Garbage Collection
  - Benutzerdefinierte Auswertung (Administration der Charts)

### 7.3.3 Projektstruktur

Wie im Abschnitt Installation der Software erläutert, besteht die Software aus zwei getrennten Features. Das Core-Feature ist die Basis und verantwortlich für den gesamten Import-Prozess (Import-Wizard, Leseprozess der Log-Datei, Anzeige der Menus, Profil-Verwaltung, etc.). Die JRockit Extension ist eine für die Garbage Collection Logs der JRockit geschriebene Erweiterung. Sie ist für das Parsing der Log-Dateien, die Aufbereitung der Daten und die Anzeige der Charts zuständig. Beinhaltet aber keine Core-Funktionalität.

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

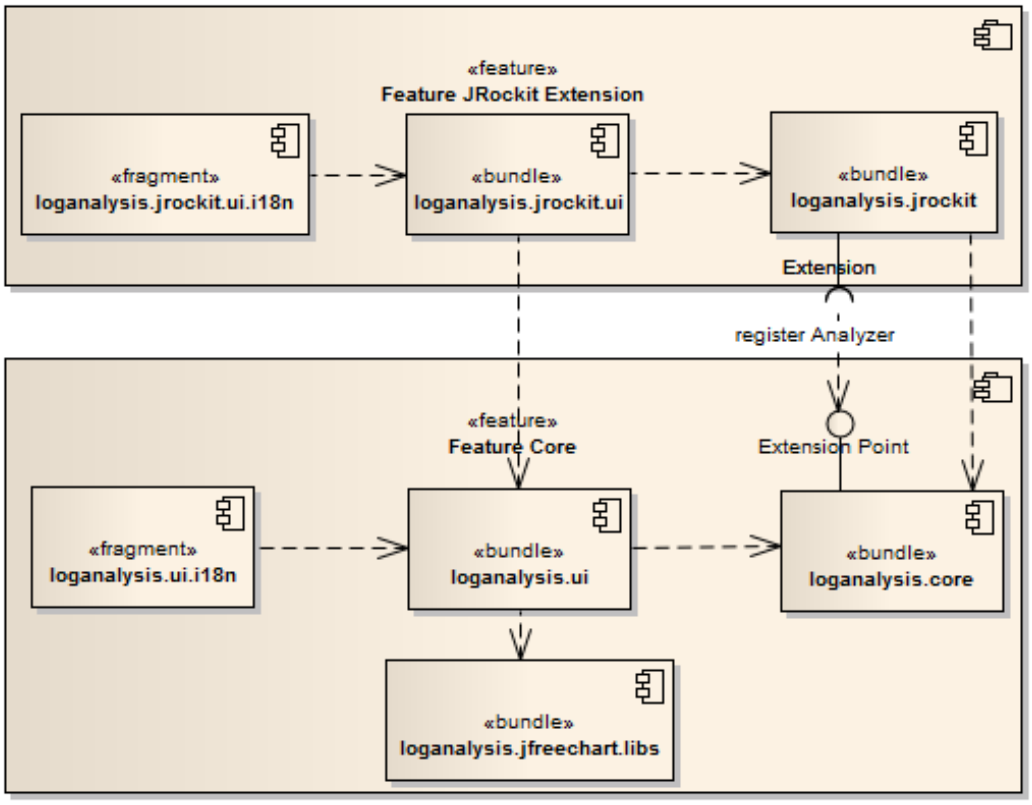



Abbildung 7.1: Architektur: Komponentendiagramm

Das Core Feature besteht aus dem Modul User Interface ("loganalysis.core.ui") und einem von JFace und SWT[6] unabhängigen Teil ("loganalysis.core"). Öffnet der Benutzer eine Garbage Collection Log Datei, wird diese durch das Core Feature eingelesen und an alle verfügbaren Extensions weitergeleitet. Die erste Extension welche den Inhalt der Datei versteht, öffnet seine dafür vorgesehenen Reports und Charts. Jede Extension hat ein basierend auf dem Log-Format eigenes Domänen-Modell.

### Weitere Projekte

Einige Plugins wurden im vorherigen Abschnitt nicht erwähnt:

- Test-Projekte[7].
  - core.test

[6] JFace und SWT wird in Eclipse als Library für den Presentation Layer verwendet

[7] Der Test-Code befindet sich in eigenen Projekten. Siehe Abschnit 7.6.1 Testabdeckung (QRQ-S-02)

  - core.ui.test
  - jrockit.test
  - jrockit.ui.test

- Features[8]
  - loganalysis.feature
  - loganalysis.jrockit.feature
- Thirdparty Bibliotheken[9]
  - loganalysis.jfreechart.libs (JFreeChart Library)
- Targetplattform[10]
  - loganalysis.targetplatform (beinhaltet die Target-Plattform)
- Update-Seite[11]
  - loganalysis.updatesite (definiert und generiert die Update-Seite)

### 7.3.4 Ablauf Garbage Collection Analyse

Als Abstraktion für eine Log-Datei wird die Klasse IFileDescriptor verwendet. Beim öffnen einer Analyse wird via den Context die Extension gesucht[12], welche den Inhalt[13] der Datei versteht. Sobald die entsprechende Extension gefunden wurde, wird der Inhalt geparst und das Domänenmodell instanziert. Danach wird der Analyse-Editor mit dem entsprechenden Modellobjekt geöffnet. Der Editor ist ebenfalls sehr proprietär für jedes Log-Format und befindet sich in der jeweiligen Erweiterung (Beispiel JRockitAnalysisEditor).

[8] Feature-Projekte definieren im Eclipse-Umfeld ein in sich lauffähiges Software-Packet. Via einen Xml-Deskriptor werden die abhängigen Plugin-Projekte definiert und in das Feature gepackt.

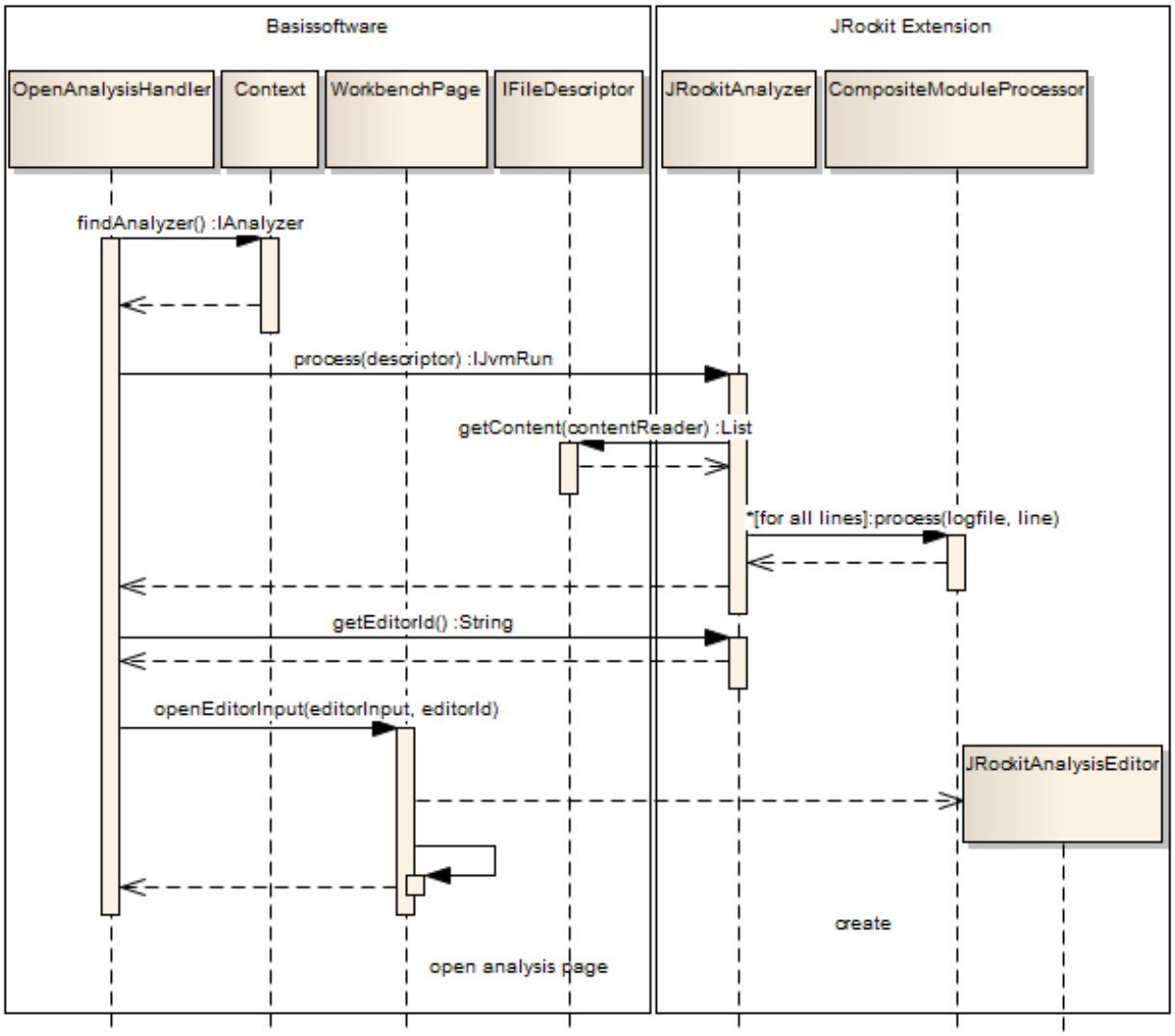
[9] Thirdparty Bibliotheken werden ebenfalls in ein Plugin gepackt, da innerhalb der Eclipse-Runntime nur Plugins installiert werden können. Die “Plugin-Hülle” definiert die exportierten und importierten Packete und die Abhängigkeiten.

[10] Die Targetplattform ist eine Konfiguration welche definiert, gegen welche Plattform die Anwendung entwickelt wird.

[11] Die Konfiguration innerhalb eines Projekts zur Erstellung einer Update-Seite definiert die auf der Seite publizierten Features.

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

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




create editor

## 7.4 Basissoftware

### 7.4.1 Installation (FRQ-01)

### 7.4.2 Installation der Software

Zur Installation der Software benötigt man die Eclipse-Entwicklungsumgebung in der Version 3.7. Darin integriert befindet sich ein Update-Manager, der Software-Komponenten von Lokal oder dem Netzwerk installieren kann. Auch Updates werden über diesen Mechanismus installiert. Die Analyse-Software wird via eine Update-Seite bereitgestellt. Der Software-Build durch das Continuous Integration System publiziert die Artefakte (Features, Plugins) auf einen via Internet zugänglichen Rechner, von welchem der Update-Manager die Software herunterlädt um anschliessend zu importieren. Update-Seiten im Eclipse-Umfeld bestehen aus Features (Eclipse Feature-Projekt). Features bestehen aus unterschiedlichen Plugins (Eclipse Plugin-Projekt). Eclipse[14] ist in der Lage, Plugins zur Laufzeit zu installieren, starten, deinstallieren.

Die Update-Seite für diese Software wird folgendermassen aufgebaut:

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

### 7.4.3 Update (FRQ-02)

Der Update eines Features auf der Update-Seite ist erkennbar durch eine Änderung der Major respektive Minor Version oder aber durch Änderung des an die Feature-Datei angehängten Zeitstempels[15]. Beim Starten der Software wird überprüft, ob ein sich auf dem Server befindendes Feature aktualisiert wurde. Der Benutzer kann diese im laufenden Betrieb der Applikation installieren. Danach ist allerdings ein Neustart der Applikation nötig.

### 7.4.4 Datei importieren (FRQ-03)

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

1. Import-Wizard öffnen
2. Auswahl des Ordners
3. Selektion einer oder mehrerer Log-Dateien
4. Bestätigung der Eingaben

[14] in der Basis ist es Equinox, die Implementation des OSGi Standards

[15] Mittels der Versionnummer x.x.qualifier erreicht man, dass der Zeitstempel ans Dateiende gehängt wird.

5. Anschliessend wird die Log-Datei als Instanz von IFileDescriptor in er Ansicht Log-Dateien angezeigt.

#### Importierte Dateien speichern

Der oben beschriebene Mechanismus wird verwendet, damit nach einem Neustart der Entwicklungsumgebung die importierten Log-Dateien nicht verloren gehen.

### 7.4.5 Datei einlesen (FRQ-04)

Nach dem Import einer Datei befindet sich zumindest ein Objekt vom Typ *FileDescriptor* in der Log-Dateien Ansicht. Durch das Öffnen einer dieser Dateien wird der Inhalt der Datei zur weiteren Verarbeitung in eine Liste geladen. Die Daten sind allerdings noch immer unstrukturiert und werden erst beim Parsen in eine strukturierte Form gebracht.

#### Domänenmodell

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

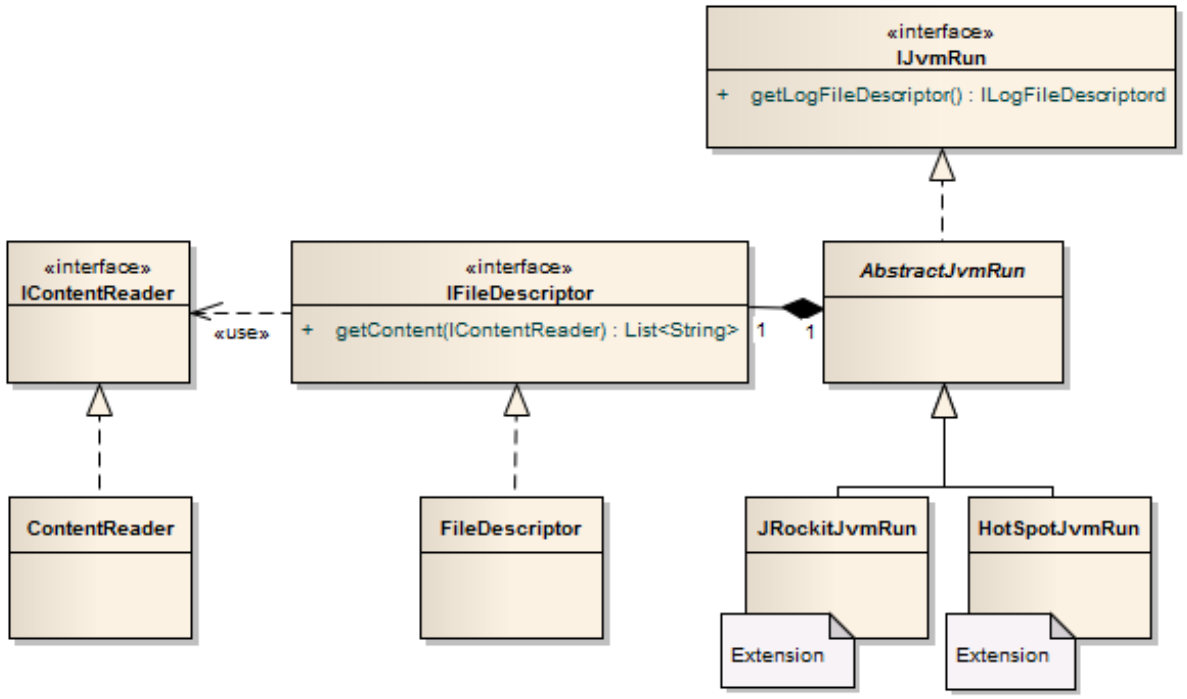



Abbildung 7.3: Domänenmodell: Top-Down Ansicht

### 7.4.6 Profil erstellen (FRQ-07)

Die Ansicht Profile zeigt die vom Benutzer erstellten Profile, initial mit dem Standard Profil. Die Beschreibung des Analysefensters wird durch Profile gemacht und es gibt zwei verschiedene Arten von Profilen:

- **Unveränderlich Profil:** Aktuell gibt es nur das Standard-Profil welches unveränderlich ist. Dieses Analysefenster kann der Benutzer nicht beinflussen, es dient dem Zweck einer ersten kurzen Übersicht über den Inhalt der Daten.
- **Veränderlich Profil:** Ein veränderliches Profil wird zur Speicherung des vom Benutzer erstellten Analysefensters verwendet. Alle Änderungen die der Benutzer am Analysefenster macht (Chart hinzufügen, Chart konfigurieren), werden via ein Data-Binding (siehe Abschnitt **??**) an das Profil propagiert. Durch das Speichern des Profils hat der Benutzer die Möglichkeit, die selbe Analyse auch zu einem späteren Zeitpunkt nochmals oder aber an einer anderen Log-Datei auszuführen.

### Domänenmodell

*IConfiguration* die zur Gruppierung der Profile. Pro Extension wird eine Konfiguration mit einer unbestimmten Anzahl an Profilen gespeichert. Innerhalb eines Profils können unterschiedliche Diagramme (*IChart*) angelegt werden, welche wiederum durch Achsen (*IAxis*) und deren Daten. Die Abfrage der Daten findet über sogenannte *IValueProvider* statt. Diese definieren den Weg, wie die Daten für eine Achse aus dem Domänen-Modell gelesen werden.

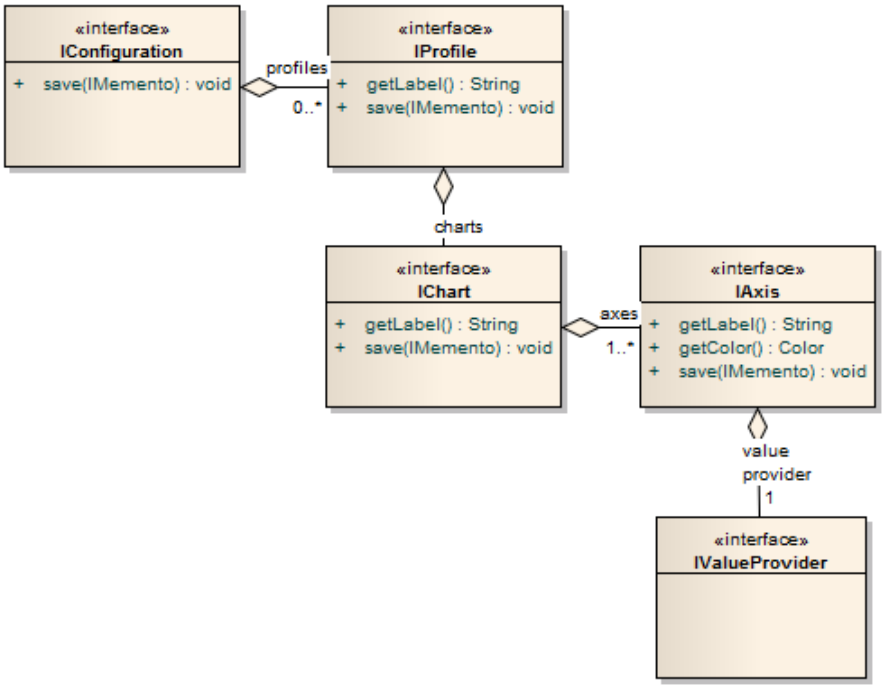



Abbildung 7.4: Domänenmodell: Profile

### Charts definieren (FRQ-07.1)

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

- Serie erstellen, definieren und hinzufügen

- Serie löschen

#### Profil speichern (FRQ-07.2)

Mittels des Memento-(siehe Abschnitt Speicherung des Zustands einer View) und Visitor-Patterns[1, S. 331] wird das Profil in ein Memento serialisiert. Dieses Memento wird von der Eclipse-Umgebung beim Beenden einer Sitzung auf die Harddisk serialisiert und beim Starten wieder deserialisiert in die Objekte.

#### Profil exportieren, importieren (FRQ-07.3/4)

Die Analysesoftware stellt zum Sichern und Verteilen von Profilen einen Import- / Export-Mechanismus zur Verfügung. Beide sind über die Import- / Export-Wizards zugänglich oder können via rechter Mouseklick inder Ansicht Profile geöffnet werden. Der Profile-Export und -Import basiert wie das Speichern auf dem im Abschnitt Speicherung des Zustands einer View beschriebenen Memento-Pattern. Zur Serialisierung in eine Datei wird das XMLMemento verwendet.

#### Hilfesystem (FRQ-08)

Das Hilfesystem der Eclipse Entwicklungsumgebung ist als Client-Server-Lösung implementiert. Beim Start der Entwicklungsumgebung wird zusätzlich ein Jetty-Server gestartet, der die Hilfeseiten und Dienste wie die Suche und Indizierung bereitstellt. Die Hilfeseiten werden in zwei unterschiedliche Arten unterteilt:

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

## 7.5 JRockit Erweiterung

### 7.5.1 Allgemein

#### Domänenmodell JRockit Garbage Collection

Die Daten der Log-Datei repräsentieren einen Lauf einer JVM (*JRockitJVMRun*) bestehend aus einem Heap und den darin enthaltenen Bereichen Keep-Area, Nursery und Tenured Space. Jeder dieser Bereiche hat unterschiedliche Zustände in welchen die verschiedenen Messgrössen gemessen und gespeichert werden. Zustandsübergänge finden durch Transitionen respektive durch eine

Garbage Collection statt, es kann sich dabei um Young oder Old Collections handeln. Das starten einer Transition wird durch einen Event ausgelöst (hier im Diagramm nicht ersichtlich), Events werden aufgrund von heuristischen Daten der Lauzeitumgebung geworfen - zum Beispiel wenn die Nursery oder die Old Collection an ihre Speichergrenze gelangen. Der Vollständigkeit halber sind im Diagramm zusätzlich noch die einzelnen Phasen der Garbage Collection definiert, die bei einer Garbage Collection durchlaufen werden. Je nach Algorithmus folgt die Sweep-Phase oder Kopaktierung auf eine Markierungs-Phase.

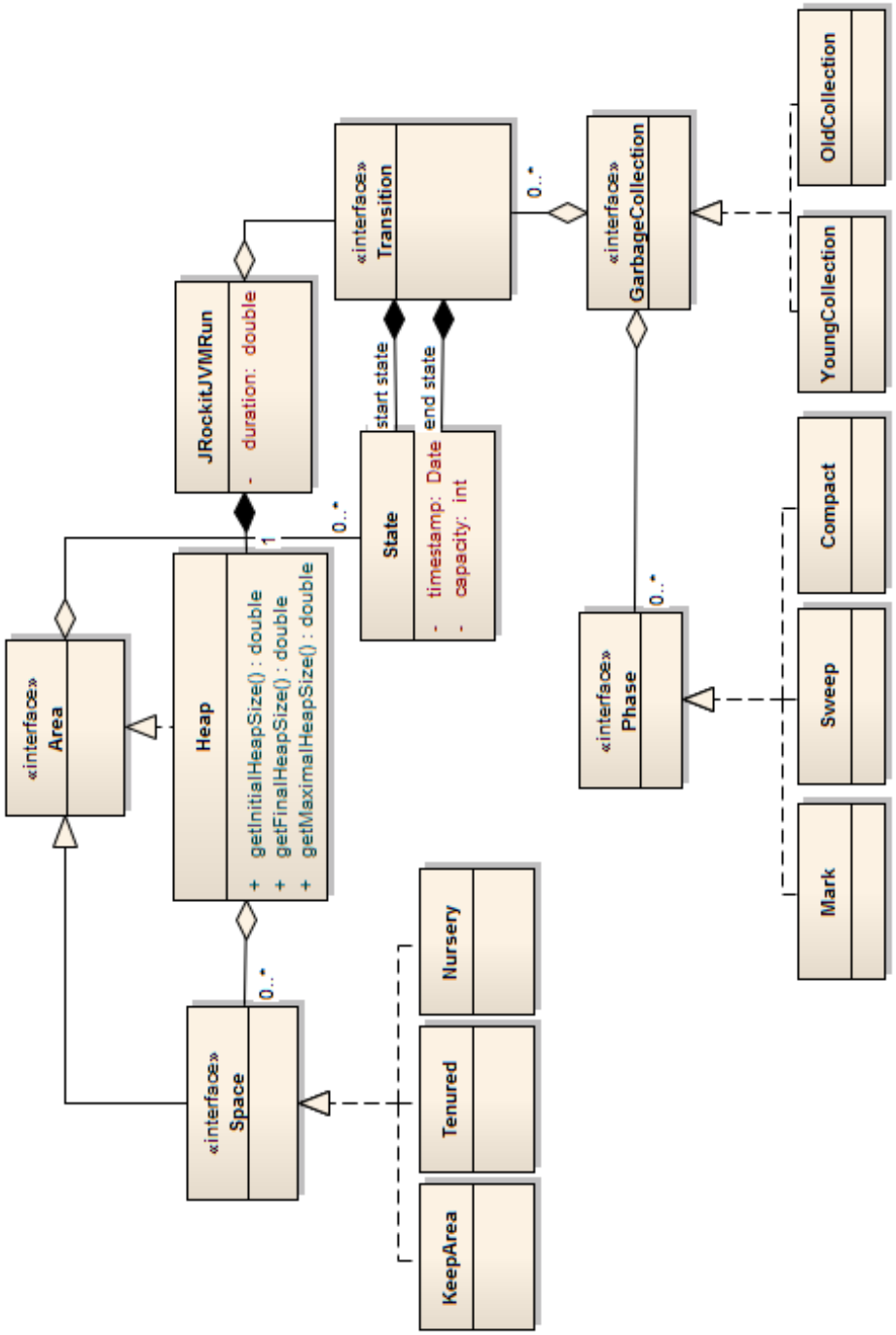



Abbildung 7.5: Domänenmodell: Garbage Collection (JRockit Implementation)

### 7.5.2 Garbage Collection Log Datei parsen (FRQ-05)

Die Garbage Collection Logs der JRockit Virtual Machine bestehen aus Einträgen unterschiedlicher Log Module. Genauer genommen können die Ausgaben betreffend Garbage Collection und Memory Allokation per Kommandozeile eingeschaltet werden (siehe Abschnit 4.3.2 Log Module). Für die Garbage Collection Analyse sind nur ein Teil dieser Einträge interessant - es werden also niemals alle dieser Einträge von der Analyse-Software verstanden. Für die Analyse der Einträge ist der *JRockitAnalyzer* zuständig.Die Einträge werden durch einen Prozessor, der nach dem Chain-of-Responsibility Pattern[10] aufgebaut ist, prozessiert. Die wichtigsten Einträge des Garbage Collection Logs sind die des Memory Modules und werden vom *MemoryModuleProzessor* geparst. Die gelesenen Daten werden interpretiert und innerhalb des Domänenmodells abgelegt. Das Chain-of-Responsibility Pattern ermöglicht es an dieser Stelle, neue Funktionalität in Form von neuen oder erweiterten Prozessoren zu definieren.

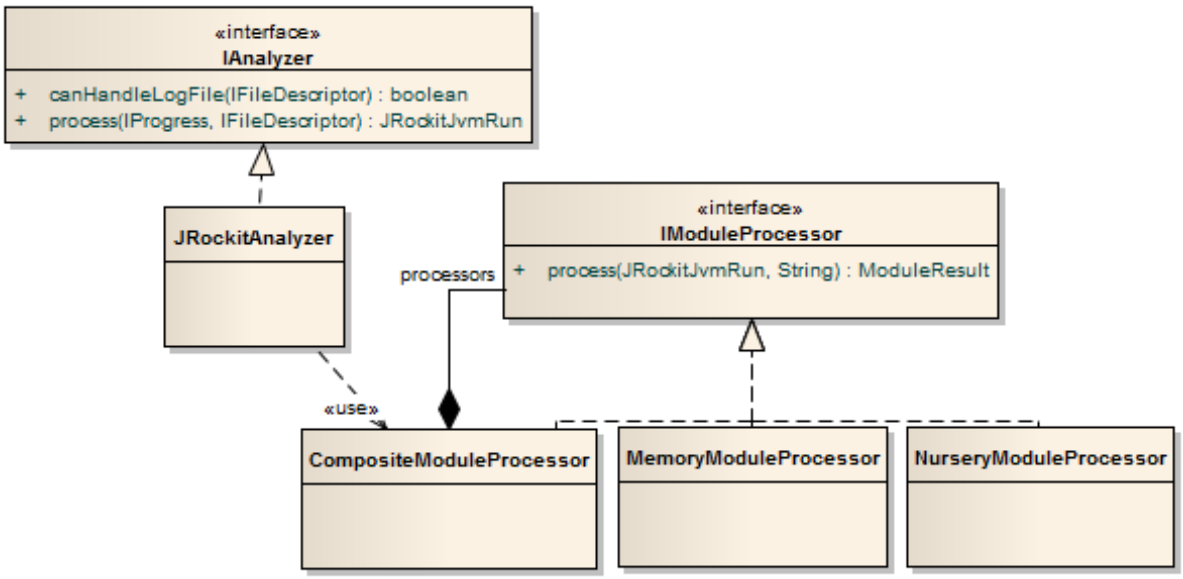



Abbildung 7.6: JRockit Analyseprozess

### 7.5.3 Standardauswertung anzeigen (FRQ-06)

#### Anzeige Übersicht Garbage Collection (FRQ-06.1)

Der initiale Tab der Analyseseite zeigt verschiedene zusammenfassende Daten des Garbage Collection Logs. Diese werden folgendermassen tabellarisch dargestellt:

- Heap Kapazität

  - Initiale Kapazität (Nursery, Tenured, Heap)
  - Maximale Kapazität (Heap)
  - Speicherbedarf Peak (Heap)
  - Kapazität Durchschnittlich (Heap)
  - Speicherbedarf Durchschnittlich (Heap)
- Garbage Collection Aktivität (Young und Old Collection)
  - Letzter Zyklus
  - Anzahl Zyklen
  - Durchschnittlicher Interval in Sekunden
  - Durchschnittliche Dauer in Sekunden
- Gesamtstatistik
  - Dauer der Messung in Sekunden
  - Anzahl Garbage Collection Zyklen
  - Total Zeit der Garbage Collection
  - Prozentuale Zeit der Garbage Collection
  - Totale Zeit der Old Garbage Collection Zyklen
  - Prozentuale Zeit der Old Garbage Collection Zyklen

**Chart Anzeige Heap Benutzung (FRQ-06.2)**

Die Heap-Analyse zeichnet den Verlauf des benutzten Speichers im Heap über die Zeit auf. Die einzelnen Garbage Collection Zyklen inklusive der Farbe als Kennzeichnung Young, Old Collection werden als Punkte im Chart markiert.

**Chart Anzeige Dauer Garbage Collection (FRQ-06.3)**

Die Dauer der einzelnen Garbage Collection Zyklen wird gegenüber der verstrichenen Laufzeit dargestellt. Die einzelnen Garbage Collection Zyklen inklusive der Farbe als Kennzeichnung Young, Old Collection werden werden als Punkte im Chart markiert.

## 7.6 Nichtfunktionale Anforderungen

### 7.6.1 Testabdeckung (QRQ-S-02)

In Nicht-Plugin-Projekten legt man den Test-Code in einem zusätzlichen Quelltext-Ordner an (Beispielsweise src/main/java und src/main/test). Damit ist der Zugriff auf package-private[16] Felder und Methoden ebenfalls möglicht, zusätzlich

[16]Felder und Methoden ohne Deklaration der Sichtbarkeit (private, protected, public) sind in Java implizit package-private - sie sind also für alle Subklassen und innerhalb des selben Ordners (Package) sichtbar.

bleibt der Test-Code von der Implementation getrennt. Dieses vorgehen wird auch vom Build-Werkzeug Maven Tycho unterstützt. Wenn man das entsprechende Test-Plugin als Eclipse-Test-Plugin konfiguriert[17], können die Tests auch während dem Build automatisiert durchgeführt werden.

Bei Eclipse-Plugin-Projekten erstellt man für den Test-Code ein separates Projekt. In das zu verteilende Softwarepacket (Feature) werden nur die Nicht-Test-Plugins getan. Der Zugriff vom Test-Projekt auf die Implementationen ist per se nicht möglich[18], zu diesem Zweck wird das Test-Projekt als Fragment definiert.

### 7.6.2 Internationalisierung (QRQ-S-03))

Basierend auf dem Lokalisierungssystem der Java Virtual Machine kann man in Eclipse alle Spracheressourcen in Properties-Dateien auslagern (Eclipse bietet dafür sogar einen Wizard). Normalerweise liefert man die Default-Sprache innerhalb des Projektes für welches die Ressource-Bundles definiert wurden, erweiterte Sprachpackete werden dann als Fragmente ausgeliefert. Den Sprachwechsel der Eclipse Entwicklungsumgebung muss man allerdings in der Eclipse-Konfigurationsdatei eclipse.ini vollziehen - dieser benötigt auch immer einen Neustart der Entwicklungsumgebunt.

### 7.6.3 Performance (QRQ-S-04))

### 7.6.4 Korrektheit (QRQ-S-05))

## 7.7 Infrastruktur

### 7.7.1 Build-Automation

Die Automatisierung des Software-Builds ist hinsichtlich der Integration in ein Continuous Integration System Voraussetzung. Es hat zusätzlich aber andere Vorteile:

- Tasks wie das Kompilieren, die Packetierung und das Deployment der Software müssen nicht mehr manuell gemacht werden.
- Zur Verhinderung von Regression und entsprechend zur Gewährleistung der Qualität können vor jedem Release automatisch die Tests durchgeführt werden.

Als Werkzeug zum automatisierten Build der Software wird Maven Tycho[19] verwendet. Um Tycho führt mittlerweile kein Weg drum herum, es hat im Vergleich zum PDE Build einige Vorteile:

[17] Dies kann innerhalb der Maven-Konfigurationsdatei pom.xml mit dem Element packaging gemacht werden.

[18] Jedes Plugin muss definieren, welche Packages für die anderen Plugins sichtbar sind.

[19] Im Bereich der Eclipse Rich Client Entwicklung kann entweder PDE Build, ein auf Apache Ant basiertes Build-System für Eclipse RCP Applikationen[9] oder die Maven-Integration Tycho (http://tycho.sonatype.org) verwendet werden.

- Maven Builds lassen sich ohne grossen Aufwand in Continuous Integration Systeme integrieren.
- Maven hat eine gute Integration in alle gängigen Entwicklungsumgebungen[20] und ist sehr verbreitet

### 7.7.2 Continous Integration

Als Continuous Integration Server wird Jenkins[21] verwendet. Jenkins bringt nebst seiner Benutzerfreundlichkeit als Build Server einige zum Projekt nötigen Voraussetzungen mit:

- Jenkins ist für alle denkbaren Betriebssysteme vorhanden.
- Jenkins ist kompatibel mit allen gängigen Systemen zur Versionskontrolle: Git, Subversion, CVS, etc.
- Maven-Projekte lassen sich ohne grossen Aufwand als Build-Projekte konfigurieren.

### 7.7.3 Versionskontrolle

Zur Versionskontrolle kommen mehrere Werkzeuge in Frage. Git[22] ist ein verteiltes Sourcecode Management System und ist konzeptionell und hinsichtlich Benutzerfreundlichkeit besser als Subversion und CVS[23]. Auf der Plattform Github[24] kann man öffentliche Projekte gratis "hosten".

### 7.7.4 Issue Tracker

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

[20] Projekt-Dateien müssen nicht mehr in die Versionskontrolle eingecheckt werden.
[21] http://jenkins-ci.org - Jenkins ist der ehemalige Hudson CI Server, welcher nach dem Kauf von Sun durch Oracle als Branch entstanden ist.
[22] http://git-scm.com
[23] Git kann offline verwendet werden, das Verschieben von Verzeichnissen führt nicht zu Problemen, etc.
[24] http://github.com

# Kapitel 8

# Implementation

# Kapitel 9

# Review

# Glossar

Tabelle 9.1: Glossar

| **Wort** | **Beschreibung** | **Herkunft** |
|---|---|---|
| Continuous Integration | | |
| 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. Test-Projekte werden oft auch als Fragmente definiert, da ihnen der volle Zugriff auf das Host-Bundle gewährleistet wird. | Eclipse |
| Garbage Collection | | |
| Garbage Collection Algorithmus, Strategie | | |

| | | |
|---|---|---|
| Plugin (Eclipse) | Ein Plugin ist eine technische Trennung von gewissen logischen Softwareteilen. Dabei definiert man die Schnittstelle zu anderen Plugins mittels einer Manifest.mf respektive einer plugin.xml Datei. Diese definiert die Abhängigkeiten (Import, Required Bundles inklusive den jeweiligen Versionen), die Exportierten Klassen und die Extension Points | Eclipse |
| Update Site | Eine Update Site ist eine per Eclipse-Konvention aufgebaute Webseite die via das Http-Protokoll zugänglich ist und Software-Features enthält. | Eclipse |
| Virtual Machine (JRockit, HotSpot) | | |

# Abkürzungen

Tabelle 9.2: Glossar

| **Abkürzung** | **Begriff** | **Beschreibung** |
|---|---|---|
| GC | Garbage Collection | |
| GUI | Graphical User Interface | |
| Unit under Test | Unit under Test | Von Unit under Test spricht man bei der zu testenden Klasse. |

# Literaturverzeichnis


[1] E. Gamma. Design patterns: elements of reusable object-oriented software. Addison-Wesley professional computing series. Addison-Wesley, 1995.

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

[3] A. Langer and K. Kreft. Generational garbage collection. Java Magazin, 3:26–30, 2010.

[4] A. Langer and K. Kreft. Mark-and-compact. Java Magazin, 7:20–24, 2010.

[5] A. Langer and K. Kreft. Young generation garbage collection. Java Magazin, 5:24–30, 2010.

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

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

[8] K. Pohl and C. Rupp. Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering– Foundation Level. Dpunkt.Verlag GmbH, 2010.

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

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

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

# Listings

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Teil III

# Anhang