Reverse-Engineering und Analyse einer Plug-in-basierten Java-Anwendung

Masterarbeit

B. Sc. Benjamin Harms

30. November 2013

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Arbeitsgruppe Software Engineering

Betreut durch: Prof. Dr. Wilhelm Hasselbring
Dr.-Ing. Jasminka Matevska (Astrium GmbH)
Dipl.-Inform. Reiner Jung
Dipl.-Inf. Jan Waller

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

Zusammenfassung

Langlebige Softwaresysteme sind heutzutage weit verbreitet. Zumeist betrifft dies speziell entwickelte Anwendungen, die sich nur in einem festen Umfeld nutzen lassen. Auch in der Raumfahrt existieren langlebige Systeme, wie etwa die Internationale Raumstation (ISS). Der Betrieb des Columbus-Moduls, das Teil der ISS ist, erfordert die fortlaufende Wartung und Weiterentwicklung eines komplexen Softwaresystems. Häufige Änderungen haben meist auch eine steigende Komplexität zur Folge. Bei Lapap MK II, der umfangreichsten Anwendung im Columbus-Modul, sind zunehmend unergründliche Verhaltensweisen aufgetreten. Sie betreffen das Verhalten beim Öffnen von PDF-Dokumenten sowie den Startvorgang von Lapap MK II. Die Komplexität der Anwendung hat eine eindeutige Identifikation der Ursachen bisher nicht gestattet.

In dieser Arbeit erfolgt eine Untersuchung des Verhaltens von Lapap MK II mittels dynamischer Analyse. Dafür sind zunächst die notwendigen Grundlagen näher erläutert. Nach einer Darstellung von Vorgehensweisen zum Testen von Lapap MK II außerhalb des Columbus-Moduls folgt eine Beschreibung des Aufbaus und der Zusammenhänge der relevanten Komponenten. Im Folgenden sind die Punkte für die Instrumentierung aufgeführt und die zwecks Analyse zu erhebenden Monitoringdaten diskutiert. Neben den Ausführungszeiten der Methoden ist hier auch die Auslastung der Ressourcen von Interesse.

Die Auswertung bezüglich des Verhaltens beim Öffnen von PDF-Dokumenten hat ergeben, dass die seitens Lapap MK II implementierten Methoden nicht als Ursache der Verzögerungen in Betracht kommen. Vielmehr deuten die Ergebnisse darauf hin, dass die Ursachen in zusätzlich verwendeten externen Komponenten zu finden sind. Eine übermäßige Auslastung der Systemressourcen ließ sich an dieser Stelle nicht feststellen. Für den ersten Startvorgang von Lapap MK II belegt die Analyse, dass die Verzögerung im Wesentlichen auf das Nachladen der Configuration-Data-Items zurückzuführen ist. Bereits für das erste CDI sind mehr als drei Viertel der Zeit des gesamten Startvorgangs notwendig. Zudem ist auch für das Instanziieren der Plug-ins während des ersten Startvorgangs mehr Zeit erforderlich als bei allen weiteren. Die Analyse der Ressourcennutzung zeigt, dass in der ersten Hälfte des Startvorgangs eine hohe Prozessorauslastung vorliegt. Eine Performance-Anomalie ist jedoch auch hier nicht zu erkennen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

AOP Aspect-Oriented-Programming

APM Attached-Pressurized-Module

ASS APM-Software-System

CDI Configuration-Data-Item

COBOL Common-Business-Oriented-Language

CSLI Columbus-System-Laptop-Image

DMS Data-Management-System

FTB File-Transfer-Browser

GHz Gigahertz

HTML Hypertext-Markup-Language

HTTP Hypertext-Transfer-Protocol

Hz Hertz

IPV International-Procedure-Viewer

ISS International-Space-Station

KiB Kibibyte

MMU Mass-Memory-Unit

NFS Network-File-System

ODF Operations-Data-File

PDF Portable-Document-Format

SIGAR System-Information-Gatherer

SITE System-Integration-and-Test-Environment

SLA Script-Launcher-Application

SPC Standard-Processor-Computer

SPR System-Problem-Report

UML Unified-Modeling-Language

XML Extensible-Markup-Language

Kapitel 1

Einleitung

Langlebige Softwaresysteme sind heutzutage allgegenwärtig. Hat sich eine Anwendung in einem Bereich erst einmal etabliert, gestaltet es sich meist schwierig, diese zu ersetzen [Sneed 2005]. Neben grundlegender Software, wie beispielsweise Betriebssystemen, umfasst dies vor allem speziell entwickelte Anwendungen, die sich nur in einer festen Umgebung einsetzen lassen. Zumeist erfolgt bereits die Konzeption einer solchen Anwendung unter Berücksichtigung einer Lebensdauer von mehreren Jahren oder Jahrzehnten. Entsprechend zeichnen sich diese Anwendungen häufig durch einen großen Funktionsumfang aus.

Neben dem Beheben von Fehlern sind es insbesondere die Wünsche der Anwender sowie weiterentwickelte Technologien, die eine permanente Wartung und Weiterentwicklung des Softwaresystems erfordern. Häufige Änderungen und Erweiterungen haben jedoch einen fortwährenden Anstieg der Komplexität zur Folge [Nierstrasz und Demeyer 2004], der sich auch im Quellcode widerspiegelt. Dementsprechend gestaltet es sich im Verlauf der Zeit auch für die Entwickler zunehmend schwieriger, Wartungsaufgaben durchzuführen. Durch den Umfang des Quellcodes und häufige Änderungen der internen Strukturen kann sich bereits das Auffinden von einfachen Fehlern als aufwendig erweisen. Weitere Faktoren, wie eine unvollständige Dokumentation und die zunehmende Verwendung von Nebenläufigkeit, können die Suche nach den Ursachen weiter verkomplizieren. Dies kann dazu führen, dass die Wartung bezüglich qualitativer Attribute aufgrund des hohen Aufwands ausbleibt und Abweichungen, etwa bei der Geschwindigkeit, zu dulden sind.

Die Forschung hat gezeigt, dass in solchen Fällen der Einsatz von Techniken aus dem Bereich des Reverse-Engineerings hilfreich sein kann [Ritsch und Sneed 1993; Zaidman u. a. 2006]. Mittels statischer und dynamischer Analyse ist es beispielsweise möglich, die Struktur einer Anwendung zu rekonstruieren und das Verhalten zur Laufzeit zu studieren. Somit können diese Techniken auch Softwareentwickler bei der Suche nach den Ursachen von Fehlern unterstützen.

1.1. Motivation

Das europäische Weltraumlabor Columbus ist seit 2008 ein Teil der Internationalen Raumstation (ISS). Der Betrieb ist derzeit bis mindestens 2020 geplant. Somit ist auch das Columbus-Modul als langlebiges System anzusehen. Für den Betrieb sind umfangreiche Softwaresysteme notwendig, deren Wartung und Weiterentwicklung ebenfalls über diesen Zeitraum hinweg erfolgen muss.

Eine Anwendung, die unter anderem das Bedienen des Columbus-Moduls erlaubt, ist Lapap MK II. Die fortlaufende Weiterentwicklung soll sicherstellen, dass jeder Zeit problemloses und komfortables Arbeiten mit Lapap MK II möglich ist. Bedingt durch den großen Funktionsumfang und kontinuierliche Änderungen hat sich im Laufe der Zeit ein hohes Maß an Komplexität ergeben. Obwohl für Lapap MK II eine besonders umfangreiche Dokumentation vorliegt und für jede Änderung mehrstufige akribische Tests erfolgen [Matevska 2013], war es in einigen Situationen bisher nicht möglich, dass Verhalten von Lapap MK II eindeutig zu ergründen. Zum Teil betrifft dies qualitative Attribute, wie etwa Verzögerungen beim Öffnen von PDF-Dokumenten. Erste Analysen haben hier nur wenige Erkenntnisse geliefert. Bedingt durch den hohen Arbeitsaufwand erfolgte jedoch keine weitere Untersuchung.

Um weiterführende Erkenntnisse über das Verhalten von Lapap MK II gewinnen zu können, die auch über die vorhandene Dokumentation und bisherige Kenntnisse hinaus gehen, ist es an dieser Stelle sinnvoll, eine dynamische Analyse durchzuführen. Mittels zusätzlicher Informationen über das Laufzeitverhalten ist es meist einfacher, die Ursachen für ein ungewolltes oder fehlerhaftes Verhalten zu identifizieren.

1.2. Ziele

Aus dieser Motivation heraus ergeben sich folgende Ziele für diese Arbeit:

Z1. Ursachen für die Verzögerung beim Öffnen von PDF-Dokumenten

Lapap MK II bietet die Möglichkeit, PDF-Dokumente innerhalb der Anwendung zu öffnen. Bei der Verwendung der Funktion ist es vorgekommen, dass für das Öffnen einzelner Dateien mehr als 90 Sekunden nötig waren. Dieses Verhalten trat bisher nur vereinzelt auf und ist nicht reproduzierbar. Es gilt derzeit als nicht-deterministisch.

Das erste Ziel dieser Arbeit besteht darin, die Ursachen für dieses Verhalten seitens Lapap MK II zu identifizieren. Zunächst gilt es daher zu prüfen, ob der Quellcode von Lapap MK II Anhaltspunkte für dieses Verhalten aufweist. Dies soll durch eine dynamische Analyse mit Kieker geschehen. Im weiteren Verlauf ist anhand von Daten über die Auslastung der Systemressourcen zu ergründen, ob es sich an dieser Stelle um eine Performance-Anomalie handelt. Abschließend ist zu prüfen, ob die Architektur von Lapap MK II die identifizierten Ursachen begünstigt. Die vollständige Beschreibung des Verhaltens von Lapap MK II in Bezug auf das Öffnen von PDF-Dokumenten ist in Unterabschnitt 3.1.3 aufgeführt.

Z2. Ursachen für die Verzögerung während des ersten Startvorgangs von Lapap MK II

Der Start von Lapap MK II ist ein komplizierter Vorgang, der aus vielen einzelnen Aufgaben besteht. Daher sind regelmäßig bis zu 120 Sekunden nötig, bevor der erste Startvorgang

abgeschlossen ist. Der Neustart von Lapap MK II benötigt hingegen nur etwa 30 Sekunden. Ein Teil der Verzögerung während des ersten Starts ist auf das Nachladen von Configuration-Data-Items zurückzuführen. Der Umfang des Quellcodes und die große Anzahl an Methoden haben es bisher nicht gestattet, eine detaillierte Analyse des Startvorgangs durchzuführen.

Das zweite Ziel dieser Arbeit besteht darin, die Ursachen für die Verzögerung während des ersten Startvorgangs zu identifizieren und mittels dynamischer Analyse die Ausführungszeiten einzelner Methoden und Abschnitte zu ermitteln. Dies soll vergleichend für den ersten Startvorgang sowie den Neustart von Lapap MK II geschehen. Auch hier gilt es zu ergründen, ob es sich an dieser Stelle um eine Performance-Anomalie handelt und in wieweit die Verhaltensunterschiede auf die Architektur von Lapap MK II zurückzuführen sind. Die vollständige Beschreibung des Verhaltens während des Startvorgangs ist in Unterabschnitt 3.1.3 gegeben.

Z3. Entwicklung von Lösungsansätzen

Das dritte Ziel dieser Arbeit umfasst die Entwicklung von Lösungsansätzen. Sofern die Untersuchungen für Ziel Z1 und Ziel Z2 ergeben haben, dass Ursachen für das Verhalten auf der Seite von Lapap MK II zu finden sind, gilt es hier zu erörtern, wie es möglich ist die Ursache zu beheben.

1.3. Struktur des Dokuments

Diese Arbeit ist wie folgt strukturiert: In Kapitel 2 sind zunächst die Grundlagen aufgeführt. Sie umfassen theoretische Konzepte, genutzte Techniken sowie Lapap MK II, das Objekt der Analyse. In Kapitel 3 ist beschrieben, wie Tests von Lapap MK II außerhalb des Columbus-Moduls erfolgen können. Darauf aufbauend ist das Konzept der Analyse erläutert. Dies schließt auch die Auswahl der Monitoringpunkte und Monitoringdaten ein. Weiterführende Informationen bezüglich der Durchführung der Analyse sowie über aufgetretene Auffälligkeiten sind in Kapitel 4 dargelegt. Kapitel 5 enthält eine ausführliche Auswertung der erhobenen Daten. Dies umfasst sowohl eine Evaluation des Verhaltens beim Öffnen von PDF-Dokumenten als auch eine Auswertung des Verhaltens während des Startvorgangs. In Kapitel 6 sind verwandte Arbeiten aus dem Themengebiet des Reverse-Engineerings aufgeführt. Abschließend erfolgt in Kapitel 7 eine Zusammenfassung der Ergebnisse sowie eine Beschreibung offener und weiterführender Aufgaben.

Kapitel 2

Grundlagen

Die Analyse einer Software in Bezug auf das Verhalten erfordert ein umfassendes Verständnis der Software. Das Anwenden von Techniken, wie der dynamischen Analyse, und die Auswertung der Ergebnisse setzen weitere Kenntnisse über die Technik, deren Anwendung und deren Grenzen voraus.

In diesem Kapitel sind daher zunächst die Grundlagen dieser Arbeit beschrieben. Abschnitt 2.1 enthält Begriffsdefinitionen für die Analyse. Der Bereich des Reverse-Engineerings sowie die statische und dynamische Analyse sind in Abschnitt 2.2 beschrieben. Als weitere theoretische Grundlage ist in Abschnitt 2.3 die aspektorientierte Programmierung aufgeführt. Die weiteren Abschnitte enthalten eine Beschreibung der zu analysierenden Software Lapap MK II in Abschnitt 2.4, des dafür verwendeten Kieker Monitoring Frameworks in Abschnitt 2.5 sowie einer verwendeten Programmbibliothek in Abschnitt 2.6.

2.1. Definitionen

Dieser Abschnitt enthält einige Definitionen, die im weiteren Verlauf der Arbeit von Bedeutung sind. Dies umfasst zum einen verschiedene Formen von Anomalien und zum anderen den Begriff der Reaktionszeit.

2.1.1. Anomalien

Den Begriff *Anomalie* definieren Chandola u. a. [2009] als ein Muster in Daten, das nicht dem normalen oder erwarteten Verhalten entspricht. Darauf basierend lassen sich durch das Betrachten von Daten in verschiedenen Kontexten unterschiedliche Formen von Anomalien definieren. In den folgenden Unterabschnitten sind einige Anomalieformen erläutert.

Punktanomalien

Die einfachste Form, in der eine Anomalie auftreten kann, ist die *Punktanomalie*. Ein einzelnes Datum stellt eine Punkanomalie dar, wenn es in Bezug auf die Übrigen Daten als anormal anzusehen ist. Abbildung 2.1 zeigt ein einfaches Beispiel anhand von Daten in einem zweidimensionalen Koordinatensystem. Die meisten Werte liegen hier in den Regionen N_1 und N_2. Sie beschreiben ein normales oder erwartetes Verhalten und bilden daher sogenannte *Normalregionen*. Die Punkte o_1 und o_2 hingegen liegen außerhalb der

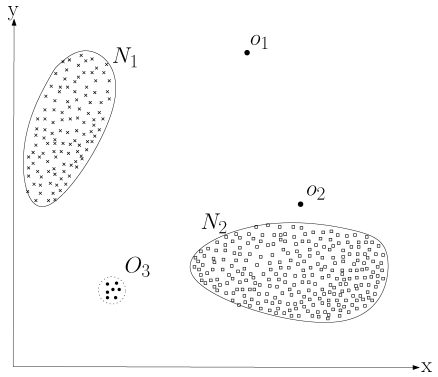

Abbildung 2.1. Punktanomalien in einem zweidimensionalen Koordinatensystem [Chandola u. a. 2009]

Normalregionen N_1 und N_2. Daher handelt es sich bei o_1 und o_2 um Punktanomalien [Chandola u. a. 2009].

Kontextanomalien

Eine weitere Form ist die *Kontextanomalie*. Um ein Datum als Kontextanomalie bezeichnen zu können, reicht es nicht aus, dieses als anormal in Bezug auf die übrigen Daten anzusehen. Dafür sind zwei gesonderte Eigenschaften des Datums zu betrachten. Dabei handelt es sich zum einen um den Wert, etwa eine konkrete Zahl, zum anderen um den Kontext des Wertes. Abbildung 2.2 zeigt den Verlauf der Temperatur an einem festen Ort über mehrere Monate. Zu jedem Punkt der Temperaturkurve sind beide Eigenschaften definiert. Der Wert ist in diesem Fall durch eine Zahl angegeben, die die Höhe der Temperatur beschreibt. Der Kontext eines Wertes ist hier der Zeitpunkt der Erhebung.

Eine Kontextanomalie zeichnet sich dadurch aus, dass nicht der Wert an sich anormal ist, sondern der Wert in Bezug auf seinen Kontext. In Abbildung 2.2 ist zum Zeitpunkt t_1 ein Wert von 35 °F gegeben, was an diesem Ort eine normale Temperatur für einen Wintermonat ist. Tritt der gleiche Wert in einem Sommermonat auf, wie zum Zeitpunkt t_2, ist dies als anormal anzusehen, da der Wert in Bezug auf die Jahreszeit zu niedrig ist. Dieses Beispiel macht deutlich, dass es sich in diesem Fall nicht um eine zuvor definierte Punktanomalie handeln kann, da ansonsten ein Wert von 35 °F generell und damit auch zum Zeitpunkt t_1 als anormal anzusehen wäre.

Performance-Anomalien

Eine *Performance-Anomalie* zeichnet sich dadurch aus, dass sich das beobachtete Verhalten einer Anwendung nicht durch deren Arbeitslast erklären lässt. Dies kann sich zum Beispiel dadurch zeigen, dass die Prozessorauslastung anders (meist höher) ausfällt, als es durch die

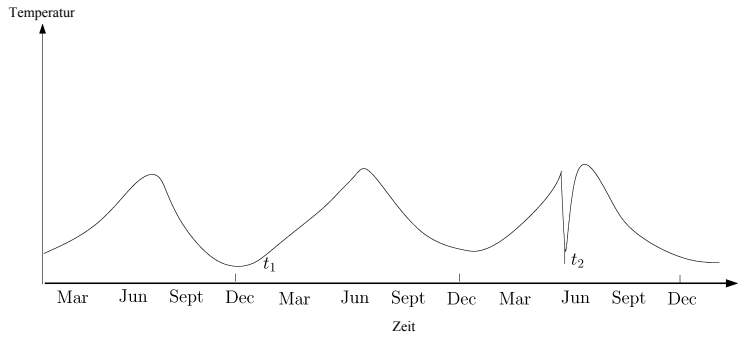

Abbildung 2.2. Kontextanomalie in einer Temperaturkurve [Chandola u. a. 2009]

Arbeitslast der Anwendung zu erwarten wäre [Cherkasova u. a. 2008; Cherkasova u. a. 2009]. Das Verhalten einer Anwendung lässt sich jedoch nicht nur durch die Ressourcenauslastung charakterisieren, sondern auch durch die Zeit, die nötig ist, um eine Aktion durchzuführen.

Die Performance-Anomalie lässt sich somit als ein Spezialfall der zuvor definierten Kontextanomalie einordnen. Die zwei Eigenschaften der Anomalie sind in ihren Definitionsbereichen jedoch beschränkt. Die Werte einer Performance-Anomalie beschreiben generell das Verhalten oder die Leistung, beispielsweise durch die Prozessorauslastung. Der Kontext eines solchen Wertes ist bei einer Performance-Anomalie grundsätzlich durch eine Aussage über die Größe der Arbeitslast gegeben.

2.1.2. Weitere Definitionen

Dieser Abschnitt enthält weitere Definitionen, die zum Verständnis der Ausführungen in Kapitel 3 und Kapitel 4 nötig sind.

Reaktionszeit für den Anwender

Für den Anwender ist die *Reaktionszeit* eines Systems definiert als die Zeit, die nach dem Anstoßen einer Aktivität durch den Benutzer vergeht, bis das System das Ergebnis präsentiert [Shneiderman 1984]. Bezogen auf die Nutzung einer Anwendung wie Lapap MK II ist das Anstoßen einer Aktivität gleichbedeutend mit dem Klicken des Benutzers, etwa auf eine Schaltfläche, oder dem Drücken einer Taste. Die Präsentation des Ergebnisses erfolgt auf einem Ausgabegerät, zumeist einem Bildschirm.

Durchschnitt

Die Begriffe *Durchschnitt* und *Durchschnittswert* bezeichnen in dieser Arbeit das *arithmetische Mittel* $\bar{x}$, das nach Stingl [2004] definiert ist:

$$\bar{x} = \frac{1}{n}(x_1 + x_2 + \cdots + x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Median

Der *Median* $\tilde{x}$ der geordneten Werte $x_1, x_2, \cdots, x_n$ ist in dieser Arbeit nach Stingl [2004] definiert:

$$\tilde{x} = \begin{cases} x_{k+1}, & falls\ n = 2k+1,\ k \in \mathbb{N} \\ \frac{1}{2}(x_k + x_{k+1}), & falls\ n = 2k,\ k \in \mathbb{N} \end{cases}$$

2.2. Reverse-Engineering

Eine grundlegende Begriffsdefinition ist von Chikofsky und Cross [1990] gegeben:

> „*Reverse-Engineering* bezeichnet den Prozess der Analyse eines Systems, um (i) die einzelnen Komponenten des Systems und deren Zusammenhänge zu identifizieren sowie (ii) das System in einer anderen Form oder auf einer höheren Abstraktionsebene darzustellen." [Chikofsky und Cross 1990]

Der Einsatz von Reverse-Engineering kann verschiedene Gründe haben. Die Analyse von Altsystemen ist ein Einsatzgebiet. Wenn ein Softwaresystem mehrere Jahrzehnte lang im Einsatz ist, sind alte Technologien und Programmiersprachen keine Seltenheit. Dies macht es für Entwickler besonders schwer, die genaue Funktionsweise zu rekonstruieren. Besonders bei älteren Programmiersprachen, wie zum Beispiel COBOL, ist eine Modernisierung aufwendig, da die meisten Softwareentwickler heute keine oder nur sehr begrenzte Kenntnisse dieser Sprache besitzen [Knoche u. a. 2012].

Bei der Wartung von Software kann es vorkommen, dass Änderungen am Quellcode erfolgen ohne diese zu dokumentieren. Dadurch kann die Dokumentation bereits nach kurzer Zeit von der Software abweichen. Auch in diesem Fall kann es nützlich sein, die Funktionsweise der Software durch die Anwendung von Reverse-Engineering-Techniken zu rekonstruieren, um die Dokumentation anzupassen. Dieser Vorgang wird auch als *Redukomentation* bezeichnet [Chikofsky und Cross 1990].

Besonders bei langlebigen Softwaresystemen, die einer kontinuierlichen Weiterentwicklung unterliegen, gestaltet sich zudem die Fehlersuche oftmals schwierig. Durch das ständige Anwachsen des Quellcodes und der damit verbundenen Komplexität lassen sich Zusammenhänge nur schwer nachvollziehen. Daher ist das Identifizieren von Ursachen meist mit hohem Aufwand verbunden. Mittels Reverse-Engineering ist es möglich, gezielte Untersuchungen durchzuführen, die das Auffinden der Ursachen erleichtern können. Zwei Techniken, die im Bereich des Reverse-Engineerings häufig zum Einsatz kommen, sind statische und dynamische Analyse [Canfora u. a. 2011].

In dieser Arbeit erfolgt eine dynamische Analyse von Lapap MK II unter Verwendung des Kieker Monitoring Frameworks. Ziel ist es, Informationen über das Verhalten von Lapap MK II zur Laufzeit zu gewinnen. Eine Auswertung dieser Informationen soll Aufschluss über die Ursachen von bekannten Problemen geben und dabei helfen, Lösungsansätze zu erarbeiten.

2.2.1. Statische Analyse

Die *statische Analyse* dient der Extraktion von statischen Informationen über eine Software. Diese zeichnen sich dadurch aus, dass sie auch ohne die Ausführung der Software verfügbar sind. Dazu zählen zum Beispiel die Struktur eines Programms, Vererbungshierarchien und Abhängigkeiten durch Methoden- oder Funktionsaufrufe. Als Eingabe sind verschiedene Artefakte denkbar, wie zum Beispiel Dokumentationen, Konfigurationsinformationen oder Tests [Koschke 2009; Canfora u. a. 2011]. Der Quellcode stellt jedoch die am häufigsten verwendete Eingabe dar. Viele Informationen, wie die Einteilung des Quellcodes in Pakete, Klassen oder Module sowie Methoden- oder Funktionsaufrufe, lassen sich meist direkt aus dem Quellcode entnehmen. Eine wichtige Eigenschaft dieser Informationen besteht darin, dass sie für alle möglichen Programmausführungen gültig sind und nicht von bestimmten Zuständen während der Laufzeit des Programms abhängen [van Deursen u. a. 2004].

Die statische Analyse eines Programms kann meist einfach und schnell erfolgen, die Ergebnisse können aber unvollständig sein. Die Verwendung von Pointern oder Polymorphie erschwert die eindeutige Identifikation von Beziehungen oder macht sie zum Teil unmöglich. Dies betrifft alle Eigenschaften, die erst zur Laufzeit eindeutig zu identifizieren sind [Ernst 2003; Canfora u. a. 2011].

2.2.2. Dynamische Analyse

Die *dynamische Analyse* ermöglicht es, Informationen über das Verhalten eines Programms zur Laufzeit zu sammeln. Anders als die statische Analyse setzt die dynamische Analyse ein ausführbares Programm voraus. Dies kann auch aus kompilierbarem Quellcode entstehen. Die extrahierten Informationen können Aufschluss darüber geben, welche Beziehungen zur Laufzeit bestehen, welche Aufrufpfade existieren und welche Werte Variablen oder Pointer zu einem bestimmten Zeitpunkt haben. Weiter ist es möglich, Informationen über das zeitliche Verhalten zu sammeln, beispielsweise über die Zeit, die nötig ist, um eine Methode auszuführen. Dabei ist stets zu beachten, dass diese Informationen nur für die aktuelle Ausführung gültig sind und bei einer erneuten Ausführung abweichen können. Dies ist besonders dann der Fall, wenn die Ausführung von Eingaben oder Benutzerinteraktionen abhängig ist [van Deursen u. a. 2004; Canfora u. a. 2011]. Zusätzlich kann die dynamische Analyse auch statische Informationen extrahieren, etwa zur Struktur des Quellcodes oder Abhängigkeiten.

Die dynamische Analyse eines Programms kann sich schnell als aufwendig erweisen. Das Herausarbeiten von Messpunkten oder die Auswertung von großen Datenmengen

sind besonders zeitintensiv. Die Ergebnisse der dynamischen Analyse können, ebenso, wie die der statischen Analyse, unvollständig sein. Erfolgt die Extraktion der Daten nur an ausgewählten Punkten, können Informationen an anderen ebenfalls zur Laufzeit aktiven Punkten verloren gehen. Auch bei einer Vollinstrumentierung einer Anwendung müssen die Informationen keinesfalls vollständig sein, da nicht alle Teile der Anwendung zur Ausführung kommen müssen [Cornelissen u. a. 2009; Canfora u. a. 2011].

Zusätzlich bedeuten die Beobachtung eines Programms und die Extraktion von Informationen auch einen erhöhten Arbeitsaufwand. Dies kann die Informationen beeinflussen, da sich beispielsweise die Laufzeit eines Programms oder die Prozessorauslastung, bedingt durch das Extrahieren und Speichern der Informationen, erhöhen kann. Cornelissen u. a. [2009] bezeichnen dieses Phänomen als *Beobachtungseffekt*.

Um die oben genannten Nachteile zu minimieren, ist es sinnvoll, statische und dynamische Analyse miteinander zu kombinieren [Ernst 2003; Canfora u. a. 2011].

2.3. Aspektorientierte Programmierung

Ein wichtiger Bestandteil vieler Programmierkonzepte, wie etwa der objektorientierten Programmierung, ist die Modularisierung des Quellcodes. Ziel ist es, einzelne Anforderungen unabhängig von anderen zu realisieren und die Möglichkeit der Wiederverwendung zu erhalten. Bei der Umsetzung von Anforderungen kommt es jedoch vor, dass diese von einander abhängig sind oder einander entgegen wirken. Ein häufig genanntes Beispiel ist die Verwendung eines Logging-Mechanismus, der in der Regel vollkommen unabhängig von der Geschäftslogik ist [Hanenberg u. a. 2009]. Dennoch kann sich die Verwendung eines Loggers durch große Teile der Geschäftslogik hindurchziehen. Derartig querschnittliche Anforderungen werden als *Cross-Cutting Concerns* bezeichnet [Kiczales u. a. 1997]. Eine Realisierung von Cross-Cutting Concerns wirkt sich meist direkt auf andere Anforderungen aus. Weitere Beispiele dafür sind Debugging- und Synchronisationsanforderungen [Walker u. a. 1999; Hanenberg u. a. 2009].

Mit der aspektorientierten Programmierung (AOP) haben Kiczales u. a. [1997] ein Konzept vorgestellt, um Cross-Cutting Concerns unabhängig von der Umsetzung anderer Anforderungen zu realisieren. Es beschreibt eine Erweiterung bereits bekannter Konzepte, wie zum Beispiel die Objektorientierung. Die Umsetzung einfacher Anforderungen erfolgt dabei weiterhin durch bekannte Konstrukte, wie Objekte oder Prozeduren. Für die Realisierung von Cross-Cutting Concerns haben Kiczales u. a. [1997] ein weiteres Konstrukt hinzugefügt, das sie als *Aspekt* bezeichnen. Ein Aspekt beschreibt eine querschnittliche Anforderung, bildet jedoch eine eigene Komponente, sodass diese auf der Quellcodeebene keinen direkten Einfluss auf die Umsetzung anderer Anforderungen hat. Um einen Zusammenhang zwischen Aspekten und anderen Konstrukten herstellen zu können, muss jeder Aspekt sogenannte *Join-Points* definieren. Ein Join-Point beschreibt die Stellen im Quellcode, an denen sich die durch den Aspekt definierte querschnittliche Anforderung auf andere Konstrukte auswirkt. Dabei kann es sich beispielsweise um Methoden oder

Prozeduren handeln. Mit Hilfe eines Zusatzprogramms, dem *Aspect-Weaver*, ist es dann möglich die Anforderung und den Aspekt zusammenzuführen. Der Aspect-Weaver fügt an den Join-Points weiteren Quellcode hinzu, daher wird dieser Vorgang auch als weben bezeichnet.

AspectJ

AspectJ ist eine aspektorientierte Erweiterung für die Programmiersprache Java [AspectJ]. Mit Hilfe von AspectJ ist es möglich, Cross-Cutting Concerns, wie zum Beispiel das Loggen von Ereignissen, einfach umzusetzen. Mit dem Schlüsselwort `aspect` lässt sich in AspectJ ein Cross-Cutting Concern ähnlich wie eine Klasse in Java definieren. Ein Aspekt muss alle *Join-Points* enthalten, an denen der Cross-Cutting Concern zum Tragen kommen soll. Ein Join-Point stellt in AspectJ einen wohldefinierten Punkt in einem Programm dar, etwa einen Methodenaufruf oder einen Speicherzugriff. Die Definition der Join-Points geschieht durch sogenannte *Pointcuts*. Ein Pointcut beschreibt eine Menge von Join-Points, an denen die gleiche Aktion durchzuführen ist. Die Verwendung von Wildcards erleichtert das Aufzählen der Join-Points.

```
1  pointcut startMethod():
2    call(public void MyClass.start());
```

Listing 2.1. Die Definition eines Pointcuts

Listing 2.1 zeigt eine einfache Definition eines Pointcuts. Sie enthält einen Join-Point, der nur auf die öffentliche Methode `start()` ohne Rückgabewert innerhalb der Klasse `MyClass` zutrifft. Ein weiteres Beispiel für die Definition eines Pointcuts ist in Listing 2.2 gegeben. Für den Rückgabetyp und den Methodennamen sind Wildcards angegeben. Daher trifft der Pointcut `pubMethod()` auf alle öffentlichen Methoden der Klasse `MyClass` zu.

```
1  pointcut pubMethod():
2    call(public * MyClass.*(..));
```

Listing 2.2. Die Verwendung von Wildcards in der Definition eines Pointcuts

```
1  before(): pubMethod() {
2    System.out.println("executing " + thisJoinPoint.getSignature().toShortString());
3  }
```

Listing 2.3. Die Anwendung eines Advices

Die Definition der Aktionen, die an den Join-Points durchzuführen sind, geschieht durch sogenannte *Advices*. Die Verwendung von `before()` oder `after()` gibt beispielsweise an, was vor oder nach dem Join-Point auszuführen ist. Listing 2.3 zeigt ein einfaches Beispiel für die Anwendung eines Advices, die dazu führt, dass vor der Ausführung jeder Methode, die durch einen Join-Point des Pointcuts `pubMethod()` beschrieben ist, eine Ausgabe der Signatur auf der Konsole erfolgt. `thisJoinPoint` stell hier eine Referenz auf den aktuellen

Join-Point dar. Auf diese Weise lässt sich bereits ein einfacher Logging-Mechanismus realisieren.

Die Verbindung von Klassen und Aspekten erfolgt durch den *AspectJ-Weaver*. Wie von Kiczales u. a. [1997] beschrieben, webt der AspectJ-Weaver den zusätzlichen Quellcode an den durch die Join-Points definierten Stellen ein. Der AspectJ-Weaver bietet dafür zwei verschiedene Arten des Einwebens an: Beim *Compile-Time-Weaving* erfolgt das Einweben des zusätzlichen Quellcodes bereits beim Kompilieren, beim *Load-Time-Weaving* geschieht dies dynamisch zur Laufzeit.

Der Einsatz von AspectJ erfolgt in dieser Arbeit im Zusammenhang mit Kieker (siehe Abschnitt 2.5). Die Verwendung von Kieker macht es erforderlich, die zu analysierenden Punkte im Quellcode zu markieren, was zu einer Modifizierung des ursprünglichen Quellcodes führt. Das Hinzufügen dieser Instrumentierung stellt jedoch nach Kiczales u. a. [1997] einen Cross-Cutting Concern dar. Daher ist eine aspektorientierte Umsetzung mittels AspectJ sinnvoll, um eine Mischung der Geschäftslogik und der Erfassung der für Kieker erforderlichen Daten zu vermeiden. Die aspektorientierte Umsetzung der Instrumentierung von Lapap MK II ist zudem einfacher und schneller durchführbar als eine manuelle Instrumentierung. Für einige externe Komponenten ist kein Quellcode verfügbar. Daher ist hier die Verwendung einer aspektorientierten Instrumentierung mittels Load-Time-Weaving unerlässlich.

2.4. Lapap MK II

Das Columbus Weltraumlabor ist ein Modul der Internationalen Raumstation (ISS). Es bietet den Astronauten die Möglichkeit, auf verschiedenen Forschungsgebieten Experimente unter den Bedingungen der Schwerelosigkeit durchzuführen. Neben einer umfangreichen Forschungsausrüstung stehen auch drei spezielle *On-Board-System-Laptops* zur Verfügung, die der Besatzung mehrere Anwendungen bereitstellen. Die umfangreichste Anwendung ist *Lapap MK II* [Columbus System Laptop ADD]. Sie besteht aus mehreren Komponenten und stellt Funktionen zur Verfügung, die sich im Wesentlichen in zwei Kategorien einteilen lassen: Zum einen handelt es sich dabei um Funktionen zum Überwachen und Bedienen von Columbus, wie zum Beispiel umfangreiche Statusdisplays, zum anderen um Funktionen, die die Besatzung bei der Durchführung von Experimenten unterstützen [Lapap MK II SDD].

2.4.1. Die Benutzeroberfläche

Die Benutzeroberfläche von Lapap MK II besteht aus vielen einzelnen Komponenten. Die Anordnung der Komponenten ist durch den ausgewählten *Workspace* vorgegeben. Lapap MK II enthält mehrere vordefinierte Workspaces, die ihren Schwerpunkt zum Beispiel auf die Überwachung von Columbus oder das Lesen der Dokumentation gelegt haben. Die vordefinierten Workspaces entsprechen etwa dem Schema in Abbildung 2.3. Des Weiteren

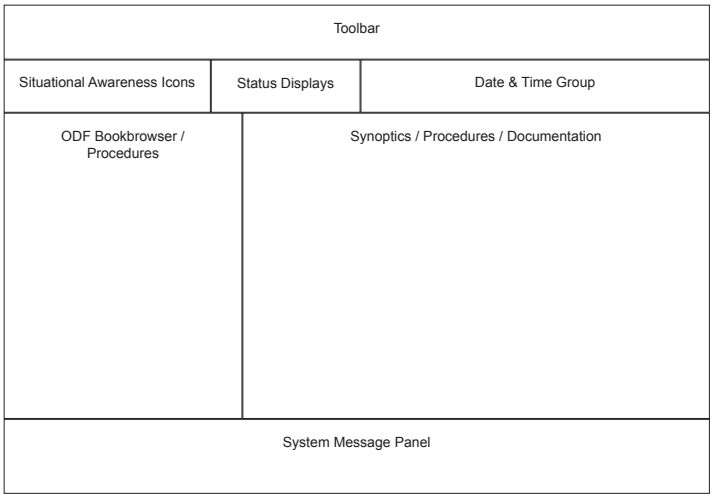

Abbildung 2.3. Schematische Darstellung der vordefinierten Workspaces von Lapap MK II [Lapap MK II SDD]

bietet Lapap MK II die Möglichkeit eigene Workspaces zu erstellen, die nach persönlichen Vorlieben gestaltet sind [Lapap MK II Manual].

Abbildung 2.4 zeigt den *Synoptics-Workspace*, der nach dem Start von Lapap MK II standardmäßig ausgewählt ist. Sein Schwerpunkt liegt auf der Überwachung von Columbus. Sechs *Synoptic-Displays* (blau markiert in Abbildung 2.4) bilden den größten Teil der Benutzeroberfläche. Jedes Synoptic-Display veranschaulicht den Zustand eines einzelnen Systems und zeigt eine Reihe von Sensordaten oder Messwerten, wie zum Beispiel Umdrehungszahlen oder Temperaturen. Um Veränderungen schnell erkennen zu können, sind die Messwerte farbig hinterlegt. Kritische Messwerte sind beispielsweise durch einen orangen Hintergrund gekennzeichnet, während grün signalisiert, dass sich der Wert im normalen Bereich befindet. Die Gestaltung der Displays sowie die Auswahl der Farben entsprechen den Vorgaben des Display and Graphics Commonality Standards [SSP 50313]. Tritt ein kritischer Messwert auf, erscheint außerdem eine Nachricht im *System-Message-Panel* (rot markiert in Abbildung 2.4). Das System-Message-Panel enthält immer die fünf aktuellsten Systemnachrichten. Weiter bekommt das *Situational-Awareness-Icon* des betroffenen Systems ebenfalls einen farbigen Hintergrund (grün markiert in Abbildung 2.4). Im Gegensatz zu den Synoptic-Displays bleibt die Hintergrundfarbe der Situational-Awareness-Icons solange erhalten, bis eine manuelle Bestätigung des Nutzers erfolgt ist, auch wenn alle Werte wieder im normalen Bereich liegen. Somit lässt sich auch das Auftreten kritischer Werte in der Vergangenheit nachvollziehen. Durch das Anzeigen von Warnhinweisen in verschiedenen Komponenten sind kritische Änderungen auch dann sichtbar, wenn ein anderer Workspace ausgewählt ist, der keine Synoptic-Displays beinhaltet. Neben den Situational-Awareness-

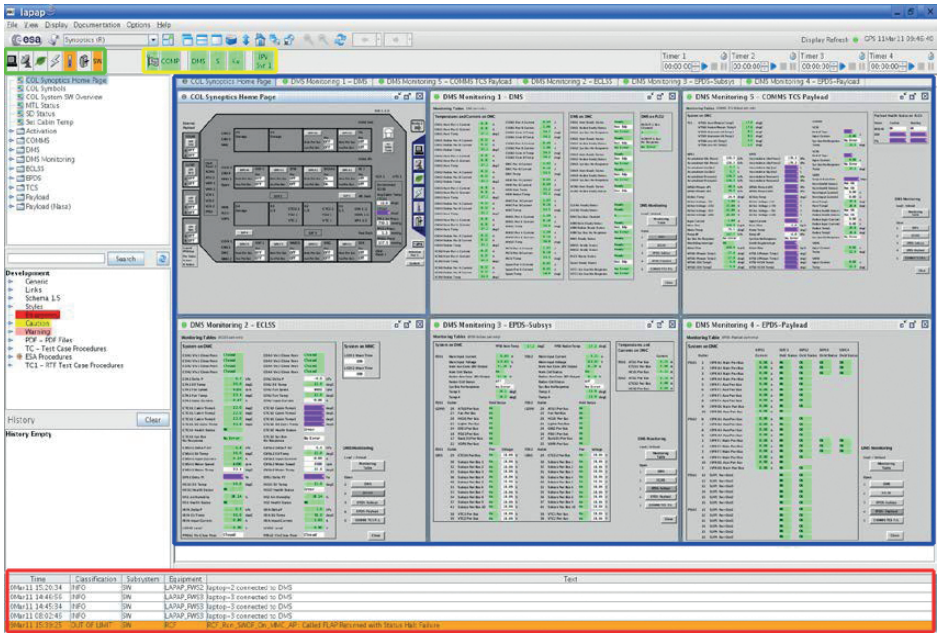

Abbildung 2.4. Der Synoptics-Workspace von Lapap MK II [Columbus System Laptop ADD] mit den hervorgehobenen Komponenten: Synoptic-Displays (blau), System-Message-Panel (rot), Situational-Awareness-Icons (grün) und Status-Display (gelb)

Icons befindet sich das *Status-Display* (gelb markiert in Abbildung 2.4). Es enthält weitere Icons, die den Zustand von anderen Systemen und Verbindungen zu anderen Systemen darstellen [Lapap MK II SDD; Columbus System Laptop ADD].

2.4.2. Architekturübersicht

Die Grundlage für Lapap MK II bildet das *Navigator-Framework*, ebenfalls eine firmeneigene Entwicklung [Navigator Framework Documentation]. Das Navigator-Framework realisiert eine Plug-in-basierte Architektur zum Entwickeln umfangreicher Anwendungen. Es stellt Mechanismen zum Initialisieren und Beenden sowie zum Konfigurieren der Plug-ins bereit. Die Kommunikation der Plug-ins untereinander ist durch einen speziellen Kommunikationsmechanismus möglich, der auf Properties basiert. Der Navigator informiert alle Komponenten, sobald sich der Wert einer Property geändert hat. Dieser Mechanismus ist mit dem Event/Event-Listener-Pattern vergleichbar [Navigator Framework Documentation]. Lapap MK II besteht aus mehreren Komponenten, die als Plug-in für das Navigator-Framework implementiert sind. Die Auswahl und Anordnung der Komponenten ist durch den ausgewählten Workspace vorgegeben (siehe Unterabschnitt 2.4.1).

In Abbildung 2.5 sind einzelne Komponenten und die Abhängigkeiten untereinander

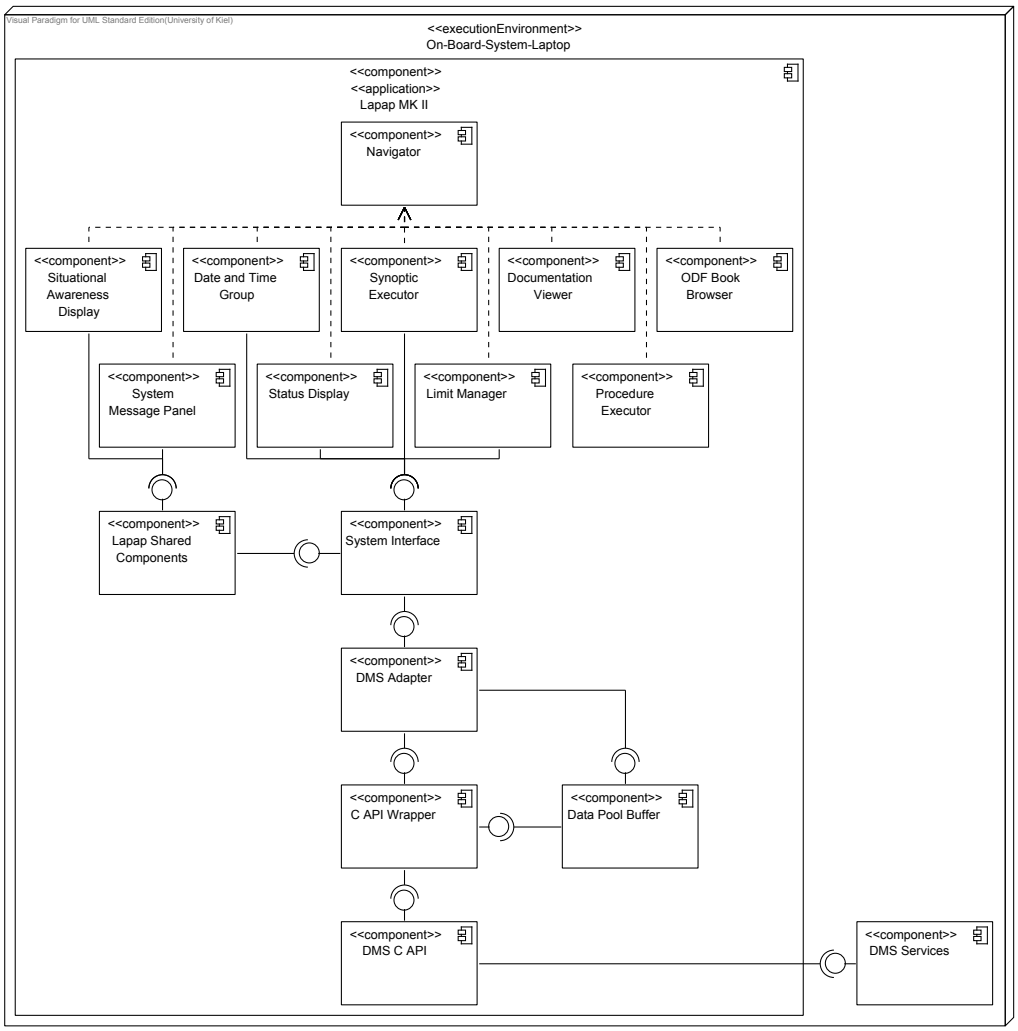

Abbildung 2.5. UML-Komponentendiagramm von Lapap MK II [Lapap MK II SDD]

durch ein UML-Komponentendiagramm dargestellt. Es zeigt die hierarchische Anordnung der Komponenten auf einzelnen Schichten. Der Zugriff auf tiefer liegende Schichten erfolgt streng entlang der Hierarchie, ohne Schichten auszulassen. Es bestehen nur einfache Beziehungen zwischen den Komponenten, Zyklen sind nicht vorhanden. Weiter ist zu beachten, dass keine direkten Verbindungen zwischen den einzelnen Plug-ins bestehen. Die Kommunikation der Plug-ins untereinander erfolgt mittels des Navigator-Frameworks.

Das Komponentendiagramm zeigt im Wesentlichen zwei Richtungen für die Abhängigkeiten. Komponenten, die als Plug-in für das Navigator-Framework implementiert sind,

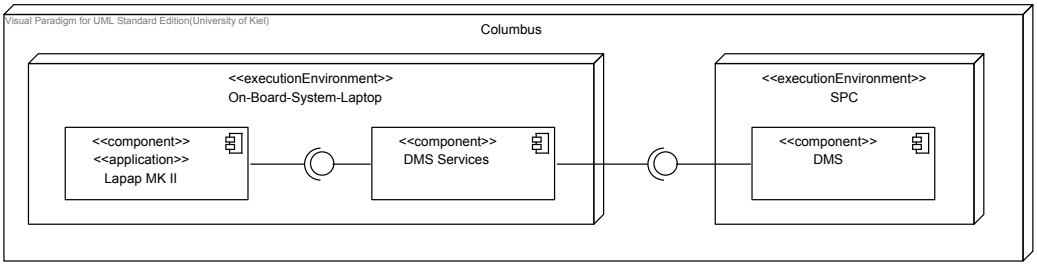

Abbildung 2.6. Zusammenhang der Komponenten für den Zugriff auf das DMS nach [Lapap MK II SDD]

besitzen eine nach oben gerichtete Abhängigkeit. Alle nach unten gerichteten Abhängigkeiten sind funktioneller Art und dienen vor allem dem Zugriff auf Komponenten in tieferen Schichten von Lapap MK II sowie dem Zugriff auf das Data-Management-System [Lapap MK II SDD].

2.4.3. Data-Management-System

Das *Data-Management-System* (DMS) ist eine zentrale Komponente des Columbus-Moduls, dessen Ausführung auf dem Standard-Processor-Computer (SPC) erfolgt [Columbus System Laptop ADD]. Es ermöglicht den Zugriff auf Messwerte aus dem Columbus-Modul, Statusdaten der Internationalen Raumstation und des Columbus-Moduls sowie Systemnachrichten. Das Data-Management-System stellt diese Daten in einem Datenpool bereit. Der Datenpool enthält alle Daten, die für Lapap MK II relevant sind, etwa zum Anzeigen in einem der Displays oder zum internen Auswerten. Die Aktualisierung dieser Daten erfolgt seitens des Data-Management-Systems mit einer Frequenz von 1 Hertz (Hz). Weiter ist das DMS für die Verarbeitung von Befehlen zuständig. Dadurch kann die Besatzung Einstellungen ändern oder Werte wie die Kabinentemperatur anpassen [Columbus System Laptop ADD; Lapap MK II SDD].

Die On-Board-System-Laptops können mittels der *DMS-Services*-Komponente auf das Data-Management-System zugreifen. Die DMS-Services sind Teil des Data-Management-Systems, auch wenn sie auf den On-Board-System-Laptops ausgeführt werden [Columbus System Laptop ADD]. Abbildung 2.6 zeigt einen Ausschnitt aus dem Komponentendiagramm des Columbus-Moduls, in dem der Zusammenhang zwischen Lapap MK II, den DMS-Services und dem Data-Management-System dargestellt ist. Eine Anwendung der Laptops erzeugt lokal sogenannte *Acquisition-Tables*, die alle Daten aus dem Datenpool des Data-Management-System enthalten. Bei einer Änderung von Werten erfolgt eine automatische Aktualisierung der lokalen Acquisition-Tables. Der Zugriff von Lapap MK II auf die Daten der Acquisition-Tables erfolgt ebenfalls mit einer Frequenz von 1 Hz. Dadurch ist sichergestellt, dass die von Lapap MK II angezeigten Werte stets aktuell sind und eine Reaktion auf kritische Änderungen umgehend erfolgen kann [Lapap MK II SDD].

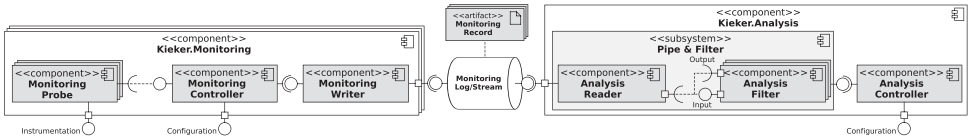

Abbildung 2.7. UML-Komponentendiagramm von Kieker nach [Kieker Project 2013]

2.5. Kieker Monitoring Framework

Kieker [van Hoorn u. a. 2012] ist ein Framework zum Überwachen und dynamischen Analysieren von Anwendungen. Es ist in Java implementiert und erlaubt daher standardmäßig die Analyse von Java-Anwendungen. Adapter zur Unterstützung weiterer Plattformen und Programmiersprachen, wie zum Beispiel .NET, Visual Basic 6 oder COBOL, befinden sich zurzeit in der Entwicklung [van Hoorn u. a. 2011b; Knoche u. a. 2012]. Der Einsatz von Kieker ist in vielen Fällen möglich, im Wesentlichen lassen sich aber zwei Einsatzgebiete herausstellen. Zum einen handelt es sich dabei um *Application-Performance-Monitoring*, eine kontinuierliche Überwachung einer Anwendung zur Laufzeit, zum anderen um *Architecture-Discovery*, die Extraktion von Informationen über die Softwarearchitektur einer Anwendung [van Hoorn u. a. 2009].

Abbildung 2.7 zeigt einen Überblick über die Architektur von Kieker. Die wesentlichen Bestandteile sind die beiden Komponenten `Kieker.Monitoring` und `Kieker.Analysis`. Sie sind in den folgenden beiden Unterabschnitten näher erläutert.

Das Kieker Monitoring Framework dient in dieser Arbeit dazu, eine dynamische Analyse von Lapap MK II (siehe Abschnitt 2.4) durchzuführen. Die Informationen über das Verhalten von Lapap MK II zur Laufzeit sollen dabei helfen, die Ursachen von bekannten Problemen zu finden und zur Entwicklung von Lösungsansätzen beitragen.

2.5.1. Die Kieker.Monitoring-Komponente

Die `Kieker.Monitoring`-Komponente stellt alle Funktionen bereit, die zur Instrumentierung einer Anwendung und dem Sammeln der Monitoring-Daten nötig sind. Der erste Schritt besteht darin, so genannte *Monitoring-Probes* in die zu überwachende Anwendung einzuarbeiten. Sie markieren die Stellen, an denen von Kieker Messdaten zu erheben sind. Gleichzeitig bestimmt der Typ einer Monitoring-Probe die Art der zu erhebenden Daten, wie beispielsweise zeitliche Aspekte oder die Auslastung von Ressourcen. Kieker bietet dafür bereits verschiedene Varianten. Die in den Quellcode eingefügten Monitoring-Probes stehen in der Regel in keinem Zusammenhang mit der Geschäftslogik der Anwendung und stellen daher einen Cross-Cutting Concern dar [Kiczales u. a. 1997]. Daher bietet Kieker auch die Möglichkeit, die Instrumentierung mittels Java-Annotationen und aspektorientierter Programmierung durchzuführen (siehe Abschnitt 2.3).

Monitoring-Records stellen die interne Repräsentation der Messdaten dar. Kieker ermöglicht es, die Monitoring-Records durch *Monitoring-Writer* zu speichern oder sie direkt

an andere Anwendungen zu übermitteln. Dafür stehen verschiedene Monitoring-Writer zur Verfügung, etwa zum Speichern in einer Datei, zum Schreiben in einen Stream oder eine Datenbank [van Hoorn u. a. 2009]. Des Weiteren bietet Kieker die Möglichkeit, eigene Komponenten, beispielsweise Monitoring-Records oder Monitoring-Writer, zu entwickeln.

Die Instrumentierung einer Anwendung hat zur Folge, dass mehr Quellcode zu verarbeiten ist. Daraus resultiert eine erhöhte Laufzeit einer instrumentierten Methode. Umfangreiche Tests haben ergeben, dass sich die durchschnittliche Laufzeit einer mit Kieker instrumentierten Methode um weniger als 10% erhöht [van Hoorn u. a. 2012].

2.5.2. Die Kieker.Analysis-Komponente

Die `Kieker.Analysis`-Komponente stellt ein Grundgerüst bereit, um eine Analyse von gesammelten Messdaten durchzuführen. Den Ausgangspunkt bilden die *Monitoring-Reader*. Sie lesen die von einem Monitoring-Writer geschriebenen Daten aus und erzeugen daraus wieder Monitoring-Records. Die Analyse der Daten erfolgt durch Plug-ins. Die Verwendung einer *Pipe-And-Filter-Architektur* erlaubt es Plug-ins beliebig miteinander zu kombinieren und die Daten im Verlauf der Analyse anzureichern oder zu verändern.

Auch für die `Kieker.Analysis`-Komponente bietet Kieker viele Möglichkeiten, eigene Komponenten zu entwickeln und zu nutzen. Meist sind die vorhandenen Komponenten, wie die Monitoring-Reader, für eine Analyse ausreichend. Die Entwicklung eigener Plug-ins kann jedoch sinnvoll sein, wenn aufwendigere Analyseverfahren oder Filterungen notwendig sind.

Mit dem `Kieker.TraceAnalysis`-Tool bietet Kieker außerdem die Möglichkeit eine schnelle Analyse der aufgezeichneten Daten durchzuführen und diese zu visualisieren. Dadurch lassen sich auf einfache Weise UML-Sequenzdiagramme, Komponentenabhängigkeitsgraphen und andere Darstellungsformen erzeugen [Rohr u. a. 2008; van Hoorn u. a. 2009].

2.6. System-Information-Gatherer

Der *System-Information-Gatherer* (SIGAR) ist eine Programmierschnittstelle zum Erfassen von systemnahen Daten [SIGAR]. SIGAR unterstützt neben Windows und Linux auch Mac OSX und andere Betriebssysteme. Dadurch stellt SIGAR eine einheitliche Programmierschnittstelle zur Verfügung, die unabhängig von dem zugrunde liegenden Betriebssystem ist. Sie erlaubt den Zugriff auf hardwarenahe Daten, wie verfügbaren Arbeitsspeicher und die Prozessorauslastung, Informationen zu einzelnen Prozessen, wie beispielsweise geöffnete Dateien und Informationen über Netzwerkaktivitäten. SIGAR ist in C implementiert, derzeit aber auch für Java, C# und Perl verfügbar.

Die Verwendung von SIGAR erfolgt in dieser Arbeit im Zusammenhang mit Kieker (siehe Abschnitt 2.5). Ziel ist es, während der Ausführung von Lapap MK II Informationen über die Ressourcenauslastung zu sammeln, um diese in Verbindung mit den von Kieker gesammelten Daten über die Ausführung auswerten zu können.

Kapitel 3

Testen und Monitoring von Lapap MK II

Um eine Analyse von Lapap MK II durchführen zu können, ist es zunächst notwendig, auf geeignete Weise Monitoringdaten zu erzeugen. Dies wiederum setzt voraus, dass es möglich ist, Lapap MK II unter realistischen Bedingungen auszuführen und Funktionen gezielt zu testen. Daher enthält Abschnitt 3.1 weitere Informationen darüber, wie es möglich ist Lapap MK II auszuführen und zu testen. Hier ist ebenfalls erläutert, wie diese Tests aufgebaut und formal beschrieben sind, welche Ergebnisse sie liefern können und wo ihnen Grenzen gesetzt sind. Außerdem ist eine ausführliche Beschreibung des Verhaltens von Lapap MK II gegeben, das es im weiteren Verlauf der Arbeit zu analysieren gilt. Abschnitt 3.2 gibt einen Überblick über die Komponenten von Lapap MK II, die dieses Verhalten hervorrufen sowie über deren Aufbau und Zusammenhang. In Abschnitt 3.3 ist aufgeführt, welche konkreten Punkte und Schnittstellen für das Erheben der Monitoringdaten relevant und welche Daten an diesen Punkten im Einzelnen von Interesse sind. Abschließend ist die Entwicklung von speziellen Testprozeduren für das zu untersuchende Verhalten in Abschnitt 3.4 erläutert.

3.1. Testen von Lapap MK II

Das Columbus-Modul und die ISS sind äußerst komplexe Systeme. Viele einzelne Komponenten aus verschiedenen Ländern der Erde wirken dort zusammen und ergeben dadurch ein einzigartiges Gesamtsystem. Der Einsatz von Lapap MK II beschränkt sich zwar auf das Columbus-Modul, dennoch findet seitens Lapap MK II auch Kommunikation mit verschiedenen Subsystemen statt. Dazu zählen beispielsweise das Data-Management-System (siehe auch Unterabschnitt 2.4.3) und der International-Procedure-Viewer (IPV). Bei der Durchführung von Änderungen an Lapap MK II oder der Entwicklung neuer Funktionen ist daher stets sicherzustellen, dass diese keine Beeinträchtigung der besagten Systeme oder der Kommunikation mit diesen zur Folge haben. Daher erfolgt das Testen von Lapap MK II unter Verwendung von verschiedenen Testverfahren und Testumgebungen. Im Folgenden ist ein Teil dieses umfangreichen Prozesses erläutert, der für das Vorgehen in dieser Arbeit relevant ist. Eine Übersicht über den Entwicklungs- und Wartungsprozess der On-Board-Software des Columbus-Moduls ist durch Matevska [2013] gegeben.

3.1.1. Testumgebungen für Lapap MK II

Für Lapap MK II existieren grundlegend verschiedene Testumgebungen, mittels denen die Ausführung von unterschiedlich gearteten Tests erfolgen kann. Die Durchführung der Tests in dieser Arbeit erfolgte unter Verwendung von zwei dieser Umgebungen, die nachfolgend erläutert sind.

Die Standalone-Version

Mit Hilfe der *Standalone*-Version von Lapap MK II ist es möglich einfache Tests durchzuführen. Die Standalone-Version lässt sich auf jedem Java-fähigen Computer ausführen und ist daher besonders für die Entwickler von Nutzen. Die Kommunikation mit anderen Subsystemen oder dem Columbus-Modul ist in diesem Fall jedoch nicht möglich. Eine Simulation ermöglicht Lapap MK II in der Standalone-Version nur grundlegende Aktionen. Sie reicht aber aus, um Lapap MK II fehlerfrei zu starten und die Synoptic-Displays mit simulierten Werten zu füllen. Außerdem erfolgt eine kontinuierliche Änderung der Daten im Datenpool, was zu häufigen Aktualisierungen der Synoptic-Displays seitens Lapap MK II führt. Im Vergleich zu dem realen Verhalten ist die Frequenz der Änderungen in der Simulation deutlich höher, ebenso wie der Anteil der kritischen Werte. Dadurch ergibt sich die Möglichkeit auch Mechanismen zu testen, die nur in Verbindung mit dem Auftreten von Wertänderungen oder kritischen Werten anwendbar sind.

Diese Testumgebung eignet sich nur für Tests, die nicht auf die reale Umgebung oder die Kommunikation mit anderen Komponenten, wie zum Beispiel dem Data-Management-System, angewiesen sind. Dazu zählen einfache Dinge, wie die Gestaltung der Benutzeroberfläche, die Funktion von Datenstrukturen oder Unit-Tests. Die Standalone-Version von Lapap MK II zeigt in einigen Bereichen ein abweichendes Verhalten gegenüber dem realen im Columbus-Modul. Dies betrifft beispielsweise auch das Arbeiten mit ODF-Prozeduren. Zum einen ist in der Standalone-Version nur ein geringer Teil aller ODF-Prozeduren enthalten, zum anderen sind diese bereits lokal verfügbar. Es ist an dieser Stelle kein Nachladen, also keine Kommunikation mit anderen Komponenten nötig.

Die System-Integration-and-Test-Environment

Um das in jeder Hinsicht reale Verhalten von Lapap MK II testen zu können, ist die Standalone-Version keineswegs ausreichend. Das optimale System für Tests wäre die reale Umgebung der ISS, da hier keine umgebungsbedingten Abweichungen zu erwarten sind. Diese Verfahrensweise verbietet sich an dieser Stelle aber, da Beeinträchtigungen einzelner Subsysteme, bei denen es sich auch um die Steuerung lebenserhaltender Maßnahmen handeln könnte, unter allen Umständen zu vermeiden sind.

Um dennoch Tests in einem realistischeren Umfeld durchführen zu können, als es mit der Standalone-Version von Lapap MK II möglich ist, gibt es die *System-Integration-and-Test-Environment* (SITE). Hierbei handelt es sich um ein System, das dem im Columbus-Modul

nachempfunden ist und die wichtigsten Bausteine der Infrastruktur durch spezielle Software simulieren kann [Matevska 2013]. Dazu zählen insbesondere das DMS und die Mass-Memory-Unit (MMU). Insgesamt existieren drei voneinander unabhängige Testumgebungen dieser Art. Mit Hilfe der SITE ergibt sich die Möglichkeit, verschiedene Konfigurationen der Komponenten und ihr Zusammenwirken zu testen. Treten dabei Fehler auf, kann zunächst eine Analyse oder auch eine Wiederholung eines Tests erfolgen. Haben Tests zu schwerwiegenden Fehlern geführt, ist es bei der SITE im Vergleich zum Columbus-Modul einfacher, die Konfiguration einer Komponente anzupassen oder diese neu zu starten. Treten Fehler im Columbus-Modul auf, bietet die SITE die Möglichkeit das Entstehen dieser Fehler zu rekonstruieren und Lösungsstrategien zu entwickeln.

3.1.2. Testszenarios für Lapap MK II

Um eine Anwendung wie Lapap MK II testen zu können, ist ein strukturiertes Vorgehen erforderlich. Daher finden die Tests auf mehreren Ebenen statt. Auf der untersten Ebene kann es sich zum Beispiel um Unit-Tests handeln, die ausschließlich die Funktionsweise einer Datenstruktur überprüfen. Die Entwickler können die Unit-Tests in der Regel selbst entwickeln und durchführen. Auf der obersten Ebene hingegen geht es darum, die Funktionsweise des gesamten Systems zu prüfen. Hierzu zählen nicht nur einzelne Datenstrukturen, sondern alle Komponenten, aus denen die Anwendung besteht oder die auf andere Weise für die Ausführung nötig sind. Viele derartige Tests lassen sich nicht oder nur zum Teil automatisieren, da sie beispielsweise auf Eingaben oder andere Interaktionen des Anwenders angewiesen sind.

Das Testen der Funktionsweise von Lapap MK II auf Systemebene erfolgt durch speziell entwickelte Testszenarios. Sie sind ein Teil der Tests, die bei jeder On-Board-Software-System Qualifikation durchzuführen sind. Jedes dieser Szenarios umfasst Tests für einen kleinen Bereich der Funktionalität. Um eine möglichst umfassende Prüfung gewährleisten zu können, sind zum Teil mehrere Varianten eines Tests in einem Szenario enthalten. Dadurch kann auch die Prüfung einzelner Details mit Hilfe eines Testszenarios erfolgen. Zu jedem Szenario sind die Rahmenbedingungen präzise definiert. Dies umfasst die Konfiguration von Software und Hardware sowie die Versionen der zu verwendenden Software und weitere Details. Auf diese Weise lassen sich Fehlerquellen, wie etwa eine fehlerhafte Ausgangskonfiguration, ausschließen. Der Aufbau eines Testszenarios und die Durchführung sind im folgenden Abschnitt anhand des Testszenarios IX noch einmal detaillierter erläutert.

Das Testszenario IX

Das Testszenario IX [TS IX] umfasst Testprozeduren aus dem Bereich der Crew-Anwendungen. Ziel dieses Testszenarios ist die Präqualifikation der Anforderungen auf Systemebene. Das Testszenario IX besteht aus neun einzelnen Testprozeduren, die jeweils Tests zu einem

oder mehreren Themenbereichen enthalten. Die Testprozeduren sind voneinander unabhängig und in beliebiger Reihenfolge durchführbar. Als Testumgebung ist die SITE vorgesehen. Die Vorbedingungen für alle Testprozeduren sind durch eine genaue Konfiguration der SITE sowie den Zuständen der drei On-Board-System-Laptops vorgegeben. Dies entspricht der zu qualifizierenden On-Board-Software Konfiguration. Die Prozeduren aus Testszenario IX enthalten unter anderem Tests für die Bereiche:

- Generelle Funktionen: Ein- und Ausschalten, Bildschirmaufteilung und Lapap MK II Workspaces
- Speichern und Wiederherstellen des Datenpools
- Event-Handling im System-Message-Panel und Message-Buffer
- Synoptic-Displays
- Documentation- und IPV-Workspace, Suche nach Dokumentation
- Arbeiten mit Dateien in Lapap MK II, File-Transfer-Browser (FTB), Software Updates
- LAN Switch Monitoring
- Geschwindigkeitsanforderungen
- Script-Launcher-Application (SLA): Funktionen und Interface, Anti-Virus

Die Testprozeduren sind aus einzelnen Schritten aufgebaut. Jeder Schritt enthält sowohl eine genaue Beschreibung der durchzuführenden Aktionen als auch detaillierte Angaben über die zu erwartenden Ergebnisse. Das Erscheinungsbild der Prozeduren ist an das Layout einer Checkliste angelehnt. Während der Durchführung ist es so möglich einzelne Schritte mit Hinweisen über Unregelmäßigkeiten oder Fehler zu versehen. Generell erfolgt die Durchführung der Tests stets unter Anwendung des *Mehr-Augen-Prinzips*. Dabei ist mindestens eine Person für das Ausführen der Prozedur zuständig, während mindestens eine weitere Person den Vorgang überwacht und die Ergebnisse protokolliert.

Das Ergebnis einer Testprozedur besteht aus einer Zusammenfassung der einzelnen Teilergebnisse in einem Testreport. Dieser ist für die Software System Qualifikation notwendig [Matevska 2013]. Sind bei der Durchführung Fehler oder Unregelmäßigkeiten aufgefallen, sind dafür System-Problem-Reports (SPRs) zu erstellen, auf deren Grundlage eine weitere Analyse erfolgen kann. Die Verwaltung der SPRs geschieht mittels eines speziellen Problem-Tracking-Tools, der *System-Problem-Report-Database*. Sind alle Fehler behoben und alle Unregelmäßigkeiten diskutiert, soll gegebenenfalls eine Wiederholung der Tests erfolgen.

3.1.3. Unregelmäßigkeiten und unergründliches Verhalten

Bei der Durchführung von Testszenarios kommt es vor, dass sich Unregelmäßigkeiten ereignen. Aber nicht jede Unregelmäßigkeit ist als schwerer Fehler anzusehen. Zum Teil handelt es sich dabei nicht um fehlerhafte Ergebnisse einer Aktion, sondern vielmehr um das Verhalten, das Lapap MK II bei der Durchführung gezeigt hat. In den folgenden zwei Abschnitten sind Beispiele für derartige Verhaltensunregelmäßigkeiten aufgeführt, die es im weiteren Verlauf dieser Arbeit zu analysieren gilt.

Öffnen von PDF-Dokumenten

Bei der Durchführung der Testprozedur 5 aus dem Testszenario IX [TS IX] trat in Schritt 50 eine erhöhte Ladezeit für ein PDF-Dokument auf [SPR-23806]. Es waren mehr als 60 Sekunden nötig, bevor das Dokument *Laptop_SUM_COL-RIBRE-MA-0096.pdf* vollständig geladen und im Documentation-Workspace von Lapap MK II sichtbar war. Basierend darauf, dass Lapap MK II während dieser Zeit keinerlei Reaktion innerhalb des Documentation-Workspaces zeigte, war die Annahme zunächst, dass das Laden des Dokuments fehlgeschlagen sei. Weitere Versuche haben aber ergeben, dass es möglich ist dieses und auch andere PDF-Dokumente im Documentation-Workspace von Lapap MK II anzuzeigen. Die Ladezeit betrug bei allen Versuchen weniger als 20 Sekunden.

Bei einer erneuten Durchführung der Testprozedur 5 zu einem späteren Zeitpunkt ließ sich abermals eine erhöhte Ladezeit beobachten. Auch in diesem Fall ließen die äußeren Umstände keinen Rückschluss auf die Ursache für dieses Verhalten zu. Durch die Auswertung der Log-Dateien war es möglich den Zeitpunkt zu bestimmen, an dem der Aufruf zum Laden des Dokuments stattfand. Hinweise auf mögliche Ursachen, wie etwa Fehlermeldungen, waren aber nicht enthalten. Weitere Fälle dieser Art sind bisher nicht bekannt. Dieses Verhalten trat ausschließlich beim Durchführen von Tests an der SITE auf, jedoch nicht an Bord der ISS.

Der Startvorgang von Lapap MK II

Das Starten von Lapap MK II ist ein komplizierter Vorgang. Der erste Start erfolgt automatisch, sobald das Betriebssystem vollständig geladen ist. Dieser erste Startvorgang ist besonders aufwendig, da es erforderlich ist, zusätzliche Daten über das Netzwerk von der MMU zu laden und lokal zu speichern. Auf Grund von unterschiedlichen Network-File-Systems auf dem Laptop und dem SPC ist hier die Verwendung der slow_copy-Funktion notwendig. Sie verlangsamt den Kopiervorgang und verhindert dadurch eine Blockade der MMU. Dies führt dazu, dass für den ersten Startvorgang im Durchschnitt bis zu 120 Sekunden nötig sind. Bei jedem weiteren Start von Lapap MK II entfällt das Nachladen von Daten aus dem Netzwerk, sofern zuvor kein Neustart des Laptops stattgefunden hat und lokale Kopien der Daten vorhanden sind. Diese Startvorgänge sind durchschnittlich nach etwa 30 bis 35 Sekunden abgeschlossen.

Erste Analysen des Startvorgangs haben bisher keine vollständigen Informationen über den genauen Ablauf ergeben. Auch die Auswertung der Log-Dateien war diesbezüglich wenig nutzbringend, da diese in der Regel auf eine Größe von zwei Megabyte beschränkt sind. Während des Startvorgangs erzeugt Lapap MK II jedoch deutlich mehr als zwei Megabyte Log-Daten. Intern nutzt Lapap MK II Apache Log4j zum Loggen von Ereignissen. Die Verwendung eines *Rolling-File-Appenders* sorgt in diesem Zusammenhang dafür, dass ein Überschreiben der Log-Datei erfolgt, sobald die maximale Größe erreicht ist [Log4j]. Dies führt dazu, dass nur die letzten Einträge des Startvorgangs vorhanden sind. Daher war eine vollständige Auswertung auf diese Weise bisher nicht möglich.

3.2. Aufbau und Zusammenhang der Komponenten

Um das Verhalten von Lapap MK II analysieren zu können, ist zunächst ein detailliertes Verständnis der Funktionsweise einzelner Komponenten und deren Zusammenhänge nötig. Der folgende Abschnitt enthält daher eine Beschreibung des Aufbaus der Komponenten, die an der Verarbeitung von PDF-Dokumenten im Documentation-Workspace beteiligt sind sowie deren Zusammenhänge. In einem weiteren Abschnitt ist der Startvorgang von Lapap MK II genauer dargestellt und eine Übersicht über die wichtigsten Komponenten und Schritte gegeben.

3.2.1. Der Documentation-Viewer

Der *Documentation-Viewer* ist die Komponente von Lapap MK II, die für das Anzeigen der Dokumentation zuständig ist. Abbildung 2.5 zeigt die Integration des Documentation-Viewers in Lapap MK II. Hier ist auch ersichtlich, dass keine direkten Abhängigkeiten zu anderen Komponenten von Lapap MK II bestehen. In Abbildung 3.1 ist der Aufbau

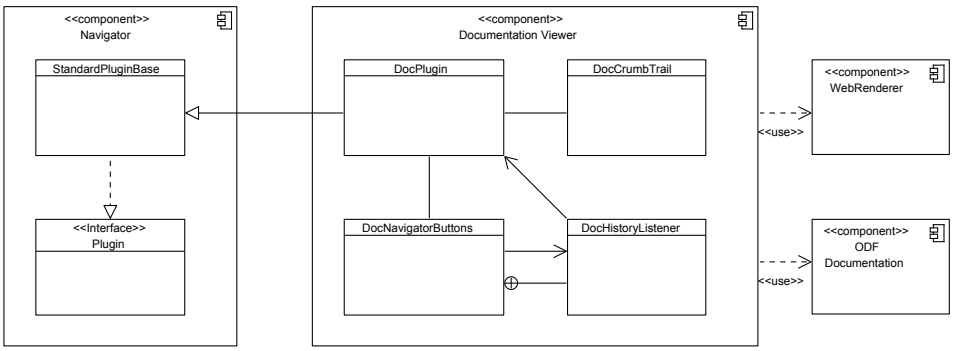

Abbildung 3.1. Auszug aus dem Klassendiagramm des Documentation-Viewers nach [Lapap MK II SDD]

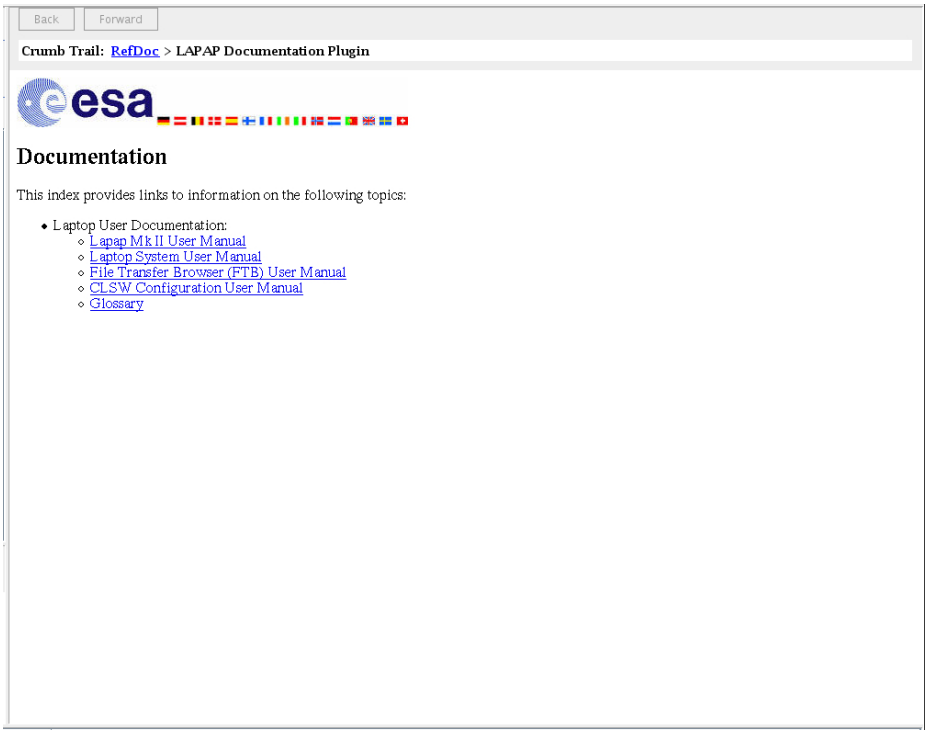

Abbildung 3.2. Ausschnitt aus dem Documentation-Workspace von Lapap MK II mit den Navigationsschaltflächen, der Crumb-Trail-Komponente und der Browser-Komponente mit Startseite [Lapap MK II Manual]

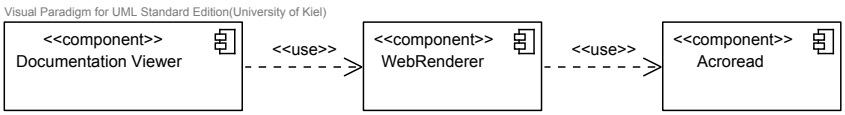

Abbildung 3.3. Zusammenhang der Komponenten für das Öffnen von PDF-Dokumenten

des Documentation-Viewers anhand eines Klassendiagramms dargestellt. Der wesentliche Bestandteil ist die Klasse `DocPlugin`. Sie ist als Plug-in für das Navigator-Framework implementiert und stellt die visuellen Komponenten zur Integration in die Benutzeroberfläche von Lapap MK II bereit. Eine dieser Komponenten zeigt den Pfad zum derzeit ausgewählten Dokument an und ist durch die Klasse `DocCrumbTrail` realisiert. Bei einer weiteren Komponente handelt es sich um die Vor- und Zurück-Schaltflächen, die es ermöglichen durch zuvor geöffnete Dokumente zu navigieren, wie es von Browsern bekannt ist. Sie sind mittels der Klasse `DocNavigatorButtons` umgesetzt. Abbildung 3.2 zeigt die Anordnung der durch die Klasse `DocPlugin` bereitgestellten Komponenten in einem Aus-

schnitt aus dem Documentation-Workspace von Lapap MK II. Zusätzlich verwendet der Documentation-Viewer den `WebRenderer` [WebRenderer], eine externe Komponente. Der `WebRenderer` stellt verschiedene Browser-Komponenten bereit, die sich in andere Anwendungen integrieren lassen. Die Klasse `DocPlugin` verwendet eine Mozilla-Browser-Komponente des `WebRenderers`, die ebenfalls in den Documentation-Viewer integriert ist. Mittels dieser Komponente erfolgt letztendlich das Anzeigen von Dokumenten, die in HTML- oder PDF-Format vorliegen. Der `WebRenderer` wiederum nutzt zum Darstellen von PDF-Dokumenten den auf Betriebssystemebene installierten Adobe Reader. Abbildung 3.3 zeigt diese Zusammenhänge.

Innerhalb des Documentation-Workspaces können, insbesondere bei der Verwendung der Suchfunktion, auch Ziele verlinkt sein, bei denen es sich nicht um PDF- oder HTML-Dokumente handelt. Zum einen kann dies ODF-Prozeduren betreffen, die zum Teil auch als XML-Dokumente vorliegen, zum anderen auch Displays. Das Anzeigen dieser Ziele geschieht nicht durch den Documentation-Viewer, sondern mittels des ODF-Procedure-Viewers beziehungsweise durch Öffnen des Displays im Synoptics-Workspace.

3.2.2. Der Startvorgang von Lapap MK II

Das Navigator-Framework bildet die Grundlage für Lapap MK II, daher erfolgt auch der Start aus dem Navigator-Framework heraus. Zunächst ist dafür die Initialisierung des Navigators und seiner Komponenten erforderlich. Viele dieser Komponenten sind ebenso wie die Komponenten von Lapap MK II als Plug-ins realisiert. Dazu zählen unter anderem das `NavigatorCorePlugin`, das `SystemPlugin` und das `UIManagerPlugin`. Der zentrale Einstiegspunkt befindet sich in der Klasse `NavigatorLauncher`. Dort beginnt die Initialisierung und erfolgt im weiteren Verlauf auch unter Verwendung der Klassen `Launcher` und `Navigator`. Hier findet auch das Laden und Initialisieren der Plug-ins von Lapap MK II statt.

Das Navigator-Framework verwendet außerdem weitere Komponenten, sowohl externe als auch firmeneigene Entwicklungen. Besonders wichtig für die Analyse des Startvorgangs ist die Komponente `SystemInterface`. Sie enthält die Klasse `SystemFacadeImpl` sowie die abstrakte Klasse `SystemAdapter`. In Verbindung mit der Klasse `DMSAdapter`, die den `SystemAdapter` erweitert, erfolgt hier der Aufbau der Verbindung und die Kommunikation mit dem Data-Management-System. Die Initialisierung des `DMSAdapters` umfasst auch das Initialisieren und Konfigurieren weiterer Komponenten, die für die Kommunikation mit dem Data-Management-System (DMS) notwendig sind.

Eine dieser Komponenten ist der Datenpool von Lapap MK II (siehe auch Unterabschnitt 2.4.3). Der Datenpool besteht aus mehreren Klassen, von denen für die Analyse im Wesentlichen die Klassen `DataPoolCtrlImpl`, `DmsDataPoolImpl` und `DmsAcquisitionServerImpl` von Interesse sind. Mittels der Klasse `DmsDataPoolPollingThread` erfolgt das Aktualisieren der Daten im Datenpool. Eine Übersicht über die Zusammenhänge ist in Abbildung 3.4 durch UML-Sequenzdiagramme gegeben. Abbildung 3.4 (a) zeigt die Sequenz für das Initialisieren des `DMSAdapters`. Der Abschnitt für das Initialisieren des Datenpools ist in Abbildung 3.4 (b) gesondert dargestellt.

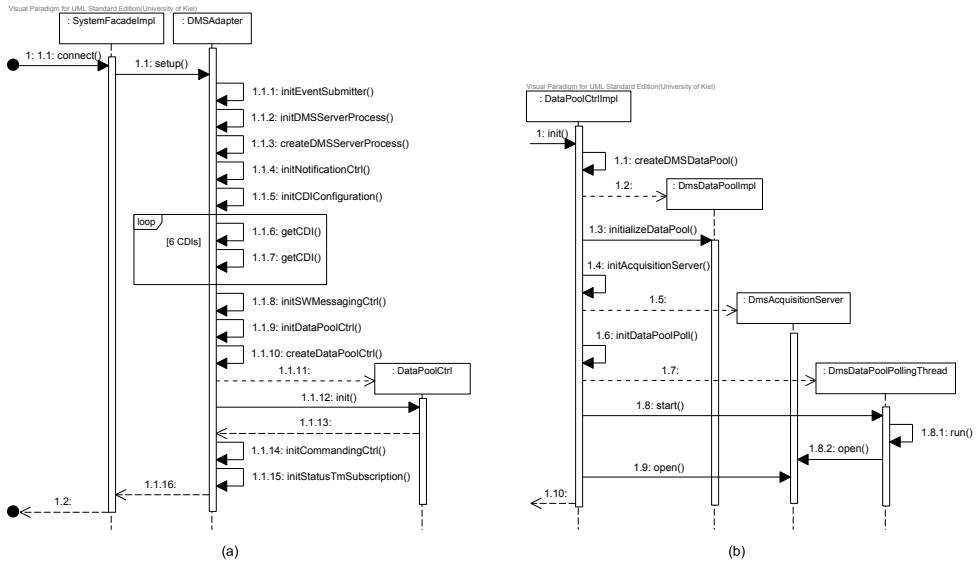

Abbildung 3.4. Sequenzdiagramm für das Initialisieren des (a) `DMSAdapters` und (b) des Datenpools

3.3. Erfassen der Monitoringdaten

Um das in Unterabschnitt 3.1.3 beschriebene Verhalten von Lapap MK II detaillierter untersuchen zu können, ist eine dynamische Analyse mittels Kieker vorgesehen. Im Vorfeld der Analyse sind zwei Faktoren zu bestimmen. Als Erstes ist es wichtig, sinnvolle Punkte zum Erfassen der Monitoringdaten auszuwählen. Von dieser Auswahl hängt später ab, ob es möglich ist mit Hilfe der gesammelten Daten eine Aussage über das untersuchte Verhalten zu treffen oder nicht. Ebenso wichtig ist die Entscheidung, welche Daten für die Verhaltensanalyse von Bedeutung und entsprechend an den zuvor ausgewählten Punkten zu erheben sind. Daher sind diese beiden Faktoren in den folgenden Abschnitten noch einmal ausführlich bezüglich der Analyse von Lapap MK II erläutert.

3.3.1. Auswahl der Monitoringpunkte

Im Folgenden ist zunächst die Auswahl der Monitoringpunkte für das Verhalten beim Öffnen eines PDF-Dokuments sowie beim Startvorgang von Lapap MK II dargelegt.

Monitoringpunkte im Documentation-Viewer

Wie bereits in Unterabschnitt 3.2.1 beschrieben ist, verwendet der Documentation-Viewer zum Anzeigen von PDF-Dokumenten eine Browser-Komponente des `WebRenderers`. Das Öffnen eines Dokuments erfolgt in der Regel durch das Aufrufen eines Links, wie sie

beispielsweise auf der Startseite des Documentation-Workspaces enthalten sind (siehe Abbildung 3.2). Da es sich hierbei aber um ein HTML-Dokument handelt, welches den Link enthält, findet bereits diese erste Aktion im `WebRenderer` statt. Auch die weitere Bearbeitung des Vorgangs bis zum Anzeigen des Dokuments führt fast ausschließlich der `WebRenderer` aus. Innerhalb von Lapap MK II ist es nur möglich, sich mittels Event-Handlern an vordefinierten Punkten in den Vorgang einzuhängen. Der `WebRenderer` bietet eine ganze Reihe solcher Punkte an [WebRenderer API], von denen Lapap MK II insgesamt drei verwendet.

Dieser Zusammenhang macht bereits deutlich, dass es für eine genaue Analyse notwendig ist, nicht nur Lapap MK II dynamisch zu analysieren, sondern auch den `WebRenderer`. Trotz zahlreicher Versuche war es bisher nicht möglich, sinnvolle Monitoringpunkte innerhalb des `WebRenderers` zu identifizieren und diese durch das Einweben von Monitoring-Probes mittels AOP in die Analyse mit einzubeziehen. In einigen Fällen führten die Tests dazu, dass vereinzelt auch Methodenaufrufe innerhalb des `WebRenderers` in den Daten von Kieker enthalten waren. Hierbei handelte es sich jedoch um Methodenaufrufe, die von anderen losgelöst und ohne jeglichen Zusammenhang vorkamen. Derartige Einträge sind in ihrer Aussagekraft stark begrenzt und tragen nicht oder nur unwesentlich zur weiteren Analyse bei, weshalb keine weitere Betrachtung dieser Einträge stattfindet. In anderen Fällen führte allein der Versuch die Monitoring-Probes einzuweben zu schweren Fehlern und letztendlich zum vorzeitigen Beenden von Lapap MK II. Aus diesen Gründen erfolgt die Analyse des Verhaltens während des Ladevorgangs eines PDF-Dokuments ausschließlich anhand der Punkte, die direkt im Quellcode von Lapap MK II implementiert sind. Im Folgenden sind die einzelnen Schnittstellen zwischen dem `WebRenderer` und Lapap MK II genauer beschrieben.

Der BrowserAdapter Mit dem `BrowserAdapter` stellt der `WebRenderer` eine abstrakte Klasse bereit, um auf Ereignisse im Browser reagieren zu können. Dazu zählen Ereignisse wie zum Beispiel das Ändern des Seitentitels oder der Adresse im Browser [WebRenderer API]. Listing 3.1 zeigt einen Auszug aus der Klasse `DocPlugin` mit der Implementation des `BrowserAdapters`. Hier ist ausschließlich die Methode `onLoadIntercept()` konkretisiert. Der Aufruf dieser Methode erfolgt seitens des `WebRenderers` unmittelbar bevor das Laden eines Dokuments beginnt. Der Documentation-Viewer nutzt dieses Ereignis, um den Typ des verlinkten Dokuments zu prüfen. Wie bereits in Unterabschnitt 3.2.1 beschrieben, ist mittels der Klasse `DocPlugin` ausschließlich das Anzeigen von HTML- oder PDF-Dokumenten möglich. Handelt es sich jedoch um eine Prozedur oder ein Display, erfolgt an dieser Stelle eine Unterbrechung des Ladevorgangs des `WebRenderers` und ein Anstoß zum Darstellen der Prozedur im Procedure-Viewer beziehungsweise zum Öffnen des Displays im Synoptics-Workspace.

In Tabelle 3.1 sind alle Methoden aufgelistet, die zwecks Überwachung der `BrowserAdapter`-Schnittstelle mit einer `OperationExecutionMonitoringProbe`-Annotation versehen sind. In diesem Fall betrifft es ausschließlich Methoden der Klasse `DocPlugin`.

```
140  private final BrowserAdapter fBrowserAdapter = new BrowserAdapter() {
141
142    @Override
143    @OperationExecutionMonitoringProbe
144    public void onLoadIntercept(final BrowserEvent browserEvent) {
145
146      final String requestedURL = browserEvent.getURL();
147      final String convertedURL = replaceVarInURL(requestedURL);
148
149      if (testAndLaunchOpsData(convertedURL)) {
150        browserEvent.blockLoad();
151
152      } else if (!requestedURL.equals(convertedURL)) {
153        browserEvent.blockLoad();
154        fBrowser.loadURL(convertedURL);
155
156      } else if (!requestedURL.startsWith("javascript:")) {
157
158        try {
159          URLHelper.check(new URL(requestedURL));
160        } catch (MalformedURLException err) {
161          LOG.warn(requestedURL + " does not denote a valid URL: "
162            + err.getMessage() + ". Unable to check for validity, allowing to load though");
163        } catch (IOException e) {
164          browserEvent.blockLoad();
165          update(MESSAGE_SENDING_REQUEST,
166            SystemLogging.createDataLinkBrokenMessage(SystemLogging.DocumentType.document,
167              getDisplayURL(requestedURL)));
168        }
169      }
170      fBrowser.getCanvas().requestFocus();
171    }
172  };
```

Listing 3.1. Implementation des `BrowserAdapters` in der Klasse `DocPlugin`

Tabelle 3.1. Annotierte Methoden zur Überwachung der `BrowserAdapter`-Schnittstelle

Klasse	Methode
DocPlugin	onLoadIntercept() replaceVarInURL() testAndLaunchOpsData() getType()

Der HistoryListener Der `HistoryListener` dient dazu eine Liste bereits aufgerufener Seiten zu verwalten und eine History-Funktion bereitzustellen. Die Methode `onURLChange()` ist der einzige Event-Handler eines `HistoryListeners`. Mit der Klasse `DocHistoryListener` ist eine eigene Variante implementiert, um einen Fehler in der Pflege der History zu vermeiden, der in der Standardimplementation des `WebRenderers` existiert. Die dafür umgesetzte Neuimplementation der Methode `onURLChange()` ist in Listing 3.2 dargestellt.

Tabelle 3.2 zeigt alle Methoden, die zum Überwachen der `HistoryListener`-Schnittstelle mit einer `OperationExecutionMonitoringProbe`-Annotation versehen sind. In diesem Fall betrifft dies ausschließlich Methoden der Klasse `DocHistoryListener`.

```
210  @Override
211  @OperationExecutionMonitoringProbe
212  public void onURLChange(BrowserEvent e) {
213    if ((!e.getURL().equals("about:blank")) && (!fNavigatorButtonAction)) {
214      addToHistory(e.getURL());
215    }
216    fNavigatorButtonAction = false;
217  }
```

Listing 3.2. Implementation der Methode `onURLChange()` in der Klasse `DocHistoryListener`

Tabelle 3.2. Annotierte Methoden zur Überwachung der `HistoryListener`-Schnittstelle

Klasse	Methode
DocHistoryListener	onURLChange() addToHistory()

Der NetworkAdapter Mit dem `NetworkAdapter` stellt der `WebRenderer` eine abstrakte Klasse bereit, um auf netzwerkbezogene Ereignisse reagieren zu können. Dazu zählen unter anderem HTTP-Ereignisse oder Netzwerkfehler [WebRenderer API]. Listing 3.3 zeigt einen Ausschnitt aus der Klasse `DocPlugin` mit der Implementation eines `NetworkAdapters`. Konkret implementiert ist hier ausschließlich die Methode `onDocumentComplete()`. Der Aufruf seitens des `WebRenderers` erfolgt, sobald der Ladevorgang im Browser abgeschlossen ist. Die Methode dient dazu, den Pfad zum aktuellen Dokument anzupassen und den Wert der Eigenschaft *MESSAGE_SENDING_REQUEST* zu aktualisieren, um den Ladevorgang als abgeschlossen zu markieren.

In Tabelle 3.3 sind alle Methoden aufgelistet, die zwecks Überwachung der Schnittstelle des `NetworkAdapters` mit einer `OperationExecutionMonitoringProbe`-Annotation versehen sind. Neben den Klassen `DocHistoryListener` und `DocPlugin` sind in diesem Fall auch die Klassen `DocNavigatorButtons` und `DocCrumbTrail` relevant.

```
174  private final NetworkAdapter fPageLoad = new NetworkAdapter() {
175    @Override
176    @OperationExecutionMonitoringProbe
177    public void onDocumentComplete(final NetworkEvent event) {
178
179      final String requestedURL = event.getURL();
180      update(MESSAGE_SENDING_REQUEST,
181        SystemLogging.createUserActionMessage(SystemLogging.Operation.Opened,
182                            SystemLogging.DocumentType.document,
183                            getDisplayURL(requestedURL)));
184      final IDocument doc = fBrowser.getDocument();
185
186      final String docTitle = doc.getTitle() == null ? "ODF Page" : doc.getTitle();
187      final String[] urlParts = requestedURL.split("#");
188
189      fCurrentURL = urlParts[0];
190      fDocNavigatorButtons.setButtons();
191      fDocCrumbTrail.updateCrumbTrail(event.getURL(), docTitle);
192    }
193  };
```

Listing 3.3. Implementation des `NetworkAdapters` in der Klasse `DocPlugin`

Tabelle 3.3. Annotierte Methoden zur Überwachung der `NetworkAdapter`-Schnittstelle

Klasse	Methode
DocPlugin	onDocumentComplete()
	getDisplayURL()
	getDocBase()
DocNavigatorButtons	setButtons()
DocHistoryListener	addToHistory()
	canGoForward()
	canGoBack()
DocCrumbTrail	updateCrumbTrail()
	createCrumbHyperText()

Der MouseAdapter Der `MouseAdapter` war ursprünglich kein Bestandteil der Implementation des `DocPlugins` von Lapap MK II. Um das in Unterabschnitt 3.1.3 beschriebene Verhalten beim Öffnen eines PDF-Dokuments untersuchen zu können, ist es zunächst erforderlich, eine feste Zeitspanne für den Ladevorgang zu definieren. Die Grundlage hierfür bildet die in Unterabschnitt 2.1.2 definierte Reaktionszeit. Da das Laden eines Dokuments fast ausschließlich seitens des `WebRenderers` erfolgt, eine Instrumentierung des `WebRenderers` bisher

```
316  fBrowser.addMouseListener(new MouseAdapter() {
317    @Override
318    @OperationExecutionMonitoringProbe
319    public void onMouseDown(MouseEvent mouseEvent) {
320      //nothing to do here
321    }
322  });
```

Listing 3.4. Implementation des `MouseAdapters` in der Klasse `DocPlugin`

aber nicht möglich war, ist es nötig, die Zeitspanne auf andere Weise zu ermitteln. Auch hier findet der seitens Lapap MK II implementierte Event-Handler `onDocumentComplete()` erneut Verwendung. Dieser Aufruf kennzeichnet das Ende des Ladevorgangs. Für den Beginn des Ladevorgangs war ursprünglich kein passender Punkt verfügbar, da der Ladevorgang beim Aufruf von `onURLChange()` oder `onLoadIntercept()` bereits begonnen hat. Den frühest möglichen Zeitpunkt, an dem der Ladevorgang beginnen kann, stellt der Aufruf des Dokuments, in diesem Fall also das Klicken auf den Link dar. Der `WebRenderer` stellt mit dem `MouseAdapter` auch hierfür einen geeigneten Ansatzpunkt zur Verfügung. Durch Implementation der Methode `onMouseDown()` ist es nun möglich, den Beginn des Ladevorgangs und damit eine feste Zeitspanne zu definieren. Um den Einfluss auf den Vorgang möglichst gering zu halten, enthält der Event-Handler keinerlei weitere Logik. Entscheidend für die Berechnung der Zeitspanne ist ausschließlich der Zeitpunkt des Aufrufs, daher ist auch die Methode `onMouseDown()` zwecks Monitoring mit einer `OperationExecutionMonitoringProbe`-Annotation versehen. Listing 3.4 zeigt die Implementation.

Monitoringpunkte für den Startvorgang

Der Startvorgang von Lapap MK II lässt sich in mehrere Abschnitte unterteilen. Für die Analyse des Verhaltens während des Startvorgangs sind im Wesentlichen drei Abschnitte relevant:

1. Die Initialisierung des Navigators und seiner Komponenten sowie die Initialisierung der Plug-ins von Lapap MK II
2. Das Initialisieren des `DMSAdapters` und der Aufbau der Verbindung zum Data-Management-System
3. Das Erzeugen und Befüllen des Datenpools

Initialisierung des Navigators Den zentralen Einstiegspunkt zum Starten von Lapap MK II bildet die Methode `main()` in der Klasse `NavigatorLauncher`. Hier erfolgt zunächst die grundlegende Initialisierung des Navigators. Dazu zählt unter anderem das Festlegen der Arbeitsverzeichnisse sowie das Laden von Konfigurationsinformationen. In einem weiteren

Tabelle 3.4. Methoden zum Überwachen der Navigator-Initialisierung

Klasse	Methode
Launcher	Launcher()
	main()
	run()
Navigator	Navigator()
	main()
	init()
	initLookAndFeel()
	loadPlugins()
	initNavigatorPluginIntegration()
	performPluginInitialization()
	activatePlugins()
	activatePlugin()
	connectPlugins()

Schritt erfolgt der Aufruf der Methode `main()` der Klasse `Launcher`, die weitere Einstellungen vornimmt. Der Aufruf der Methode `Navigator.main()` stößt im weiteren Verlauf des Startvorgangs die übrige Initialisierung an. Die Methode `initLookAndFeel()` legt hier zunächst das äußere Erscheinungsbild des Navigators fest. Im weiteren Verlauf erfolgt dann unter anderem durch das Aufrufen der Methoden `loadPlugins()`, `initNavigatorPluginIntegration()` sowie des Konstruktors der Klasse `Navigator` das Instanziieren und Initialisieren aller Plug-ins. Eine Übersicht über die für diesen Teil des Startvorgangs überwachten Methoden sowie der zugehörigen Klassen ist in Tabelle 3.4 gegeben. Tabelle 3.5 zeigt alle Plug-ins, für deren Initialisierung es ausschließlich der Ausführung des Konstruktors bedarf. Einige Plug-ins erfordern neben dem Aufruf des Konstruktors zusätzlich das Ausführen der Methode `performInitialization()`. Diese sind in Tabelle 3.6 aufgelistet.

Initialisierung des DMSAdapters Der `DMSAdapter` stellt die Verbindung zwischen Lapap MK II und dem Data-Management-System dar (siehe Abbildung 2.5) und gewährt anderen Komponenten Zugriff auf die Daten. Mittels der Methode `setup()` erfolgt die Initialisierung des `DMSAdapters` sowie der intern benötigten Objekte. Die Methode `initDMSServerProcess()` führt zunächst den Anmeldevorgang bei dem DMS-Prozess des Laptops durch und stellt dadurch die Verbindung zwischen Lapap MK II und dem DMS her. Im nächsten Schritt erfolgt das Nachladen von weiteren Konfigurationsdateien durch Ausführen der Methode `initCDIConfiguration()` sowie das Instanziieren eines `DataPoolCtrls`-Objekts, das den Datenpool verwaltet. Alle zum Analysieren des `DMSAdapters` überwachten Methoden sind in Tabelle 3.7 aufgelistet.

Initialisierung des Datenpools Alle Daten, die Lapap MK II anzeigt oder verarbeitet, stammen aus dem Datenpool des Data-Management-Systems. Diese sind zusätzlich in zwei

Tabelle 3.5. Plug-ins ohne zusätzliche Initialisierung durch den Navigator

Klasse
DMSMsgLoggerPlugin
HotKeyHelpPlugin
MessageProviderPlugin
ODFBookBrowserPlugin
ODFEventCodePlugin
ODFProcPlugin
ResetFunctionPlugin
SearchPlugin
ServerConnectionHandlingPlugin
SMPPlugin
StandardDisplayNavigationCommandPlugin
StatusPlugin
SystemInterfacePlugin
UserLoggerPlugin
UncaughtExceptionHandlingPlugin

Tabelle 3.6. Plug-ins mit zusätzlicher Initialisierung durch die Methode `performInitialization()`

Klasse
DataPoolLoggerPlugin
DisplayHierarchyPlugin
DisplayPerformanceIndicatorPlugin
DocPlugin
DTGPlugin
ToolTipConfigurationPlugin
USSPlugin

Acquisition-Tables lokal gespeichert (siehe auch Unterabschnitt 2.4.3). An der Verwaltung des Datenpools sind mehrere Objekte beteiligt. Der `DMSAdapter` erzeugt bereits bei der Initialisierung ein `DataPoolCtrl`-Objekt, aus dem drei weitere Schlüsselobjekte hervorgehen. Dabei handelt es sich im Einzelnen um ein `DMSDataPoolImpl`-Objekt, das die eigentliche Implementation des DMS-Datenpools enthält, ein Objekt der Klasse `DmsDataPoolPollingThread`, das die aktuellen Werte des Data-Management-System kontinuierlich abfragt und den lokalen Datenpool gegebenenfalls aktualisiert sowie ein `DmsAcquisitionServerImpl`-Objekt, das die Kommunikation mit dem DMS übernimmt und die Synchronisation mit der Aktualisierungsfrequenz des DMS von 1 Hz ermöglicht. Tabelle 3.8 enthält alle in diesem Bereich überwachten Methoden.

Tabelle 3.7. Methoden zum Überwachen der `DMSAdapter`-Initialisierung

Klasse	Methode
SystemFacadeImpl	connect()
DMSAdapter	setup()
	initDMSServerProcess()
	createDMSServerProcess()
	initCDIConfiguration()
	getCDI(3)
	getCDI(2)
	initDataPoolCtrl()
	initEventSubmitter()
	createEventSubmitter()
	initNotificationCtrl()
	createNotificationCtrl()
	initSWMessagingCtrl()
	initStatusTmSubscription()
	initCommandingCtrl()

Tabelle 3.8. Methoden zum Überwachen des Datenpools und dessen Pflege

Klasse	Methode
DataPoolCtrlImpl	init()
	createDMSDataPool()
	initDataPoolPoll()
	initAcquisitionServer()
DmsDataPoolImpl	initializeDataPool()
DmsDataPoolPollingThread	start()
	run()
DmsAcquisitionServerImpl	open()

3.3.2. Auswahl der Monitoringdaten

Neben der Auswahl von sinnvollen Monitoringpunkten ist auch die Zusammenstellung von geeigneten Daten von großer Bedeutung für die Analyse des Verhaltens einer Anwendung. Das Ausführen einer aufwendigen Rechenoperation kann beispielsweise viel Zeit erfordern, wenn der Prozessor keine ausreichende Rechenleistung zur Verfügung stellen kann. Dies zeigt sich meist durch eine hohe Auslastung des Prozessors oder einzelner Prozessorkerne. Ebenso kann es sich mit speicherintensiven Anwendungen verhalten, wenn kein ausreichender Arbeitsspeicher vorhanden ist. Die Analyse von Lapap MK II soll daher nicht nur auf der Auswertung der Ausführungszeiten einzelner Methoden beruhen, sondern auch

andere Faktoren, wie etwa die Auslastung von Ressourcen berücksichtigen. Im Einzelnen sind folgende Werte für die Analyse vorgesehen:

- Die Zeit für die Ausführung einzelner Methoden
- Die Auslastung des Prozessors im Ganzen sowie der durch Lapap MK II verursachte Anteil
- Die Auslastung des Arbeitsspeichers im Ganzen sowie der durch Lapap MK II verursachte Anteil
- Die Nutzung von Swap-Speicher
- Die Anzahl der aktiven Prozesse
- Die Anzahl der Threads allgemein sowie die Anzahl der Threads von Lapap MK II
- Die Anzahl der Datei-Deskriptoren von Lapap MK II

Für die Analyse der Ausführungszeiten von Methoden stellt Kieker bereits ausreichende Mechanismen zur Verfügung. Die `OperationExecutionRecords` enthalten je einen Zeitstempel für den Beginn und das Ende der Ausführung einer Methode. Diese Zeitstempel reichen aus, um Rückschlüsse auf die Ausführungszeit einer Methode zu ziehen. Zudem enthalten die `OperationExecutionRecords` weitere Daten, die eine Rekonstruktion der Aufrufpfade und damit zum Beispiel das Erzeugen von Sequenzdiagrammen ermöglichen. Für das Erzeugen der `OperationExecutionRecords` mittels aspektorientierter Programmierung stellt Kieker bereits Möglichkeiten bereit. In diesem Fall geschieht dies durch Anwendung des vordefinierten Aspekts `OperationExecutionAspectAnnotation`. Die Auswahl der Monitoringpunkte, die mit einer entsprechenden Java-Annotation gekennzeichnet sind, ist in Unterabschnitt 3.3.1 beschrieben.

Für die übrigen der oben aufgeführten Werte stellt Kieker keine ausreichenden Record-Typen zur Verfügung. Mittels der `CPUUtilizationRecords` ist zwar eine Aufzeichnung der Prozessorauslastung möglich, diese lässt jedoch keinen Rückschluss auf den Anteil einzelner Prozesse zu. Ähnlich verhält es sich mit den `MemSwapUsageRecords`. Auch hier sind nur Daten über die allgemeine Nutzung von Arbeits- und Swap-Speicher enthalten, jedoch keine Details über einzelne Prozesse. Über die Anzahl der aktiven Prozesse und Threads auf einem System stellt Kieker bisher keine Informationen zur Verfügung. Um die Daten dennoch erheben und auswerten zu können ist hier die Implementation spezieller Records notwendig. Diese sind im folgenden Unterabschnitt näher erläutert.

Um das Verhalten von Lapap MK II während der Ausführung bezüglich der Ressourcennutzung beurteilen zu können, reicht eine stichprobenartige Erhebung der Daten, beispielsweise an Stellen, an denen der Aufruf einer annotierten Methode erfolgt, nicht aus. Hier ist eine regelmäßige Erhebung nötig, um auch Tendenzen der Ressourcenauslastung erkennen zu können. Kieker stellt für diese Art des Monitorings das Interface *ISampler* bereit. Die Implementation dieses Interfaces sowie anderer notwendiger Klassen sind in weiteren Unterabschnitten beschrieben.

Entwicklung spezieller IMonitoringRecords

Die Entwicklung von *IMonitoringRecords* gestaltet sich recht einfach. Kieker stellt mit der abstrakten Klasse `AbstractMonitoringRecord` bereits eine Grundlage dafür bereit. Das Erweitern dieser Klasse und das Hinzufügen der zu speichernden Werte reicht aus, um spezielle Record-Typen zu erzeugen. Alternativ ist auch die vollständige Implementation des Interfaces *IMonitoringRecord* möglich [Kieker Project 2013]. Im Folgenden sind die neu entwickelten Record-Typen und deren Werte aufgeführt.

Das SystemInformationRecord Die Klasse `SystemInformationRecord` enthält Daten über die Systemressourcen. Die Daten beschreiben ausschließlich die Auslastung und Nutzung der Ressourcen durch das System im Ganzen. Nicht enthalten sind Informationen über die Anteile einzelner Prozesse. Im Einzelnen umfasst dies folgende Daten:

- Den aktuellen Zeitstempel – Dieser gibt Aufschluss über den genauen Zeitpunkt der Aufzeichnung der Daten. Dies ist für die spätere Auswertung von großer Bedeutung. Der bereits im `AbstractMonitoringRecord` enthaltene Logging-Zeitstempel ist hierfür nicht geeignet, da er lediglich den Zeitpunkt des Wegschreibens darstellt.
- Die Auslastung des Prozessors – Dieser Wert beschreibt die prozentuale Auslastung des Prozessors im Ganzen, nicht jedoch einzelner Kerne.
- Der aktuell freie und belegte Arbeitsspeicher – Diese Werte geben an, wie viel Arbeitsspeicher aktuell noch auf dem System zur Verfügung steht beziehungsweise bereits belegt ist.
- Die Gesamtgröße sowie der aktuell freie und belegte Swap-Speicher – Diese Werte umschreiben, in wie weit das System auf den Swap-Speicher angewiesen ist und davon Gebrauch machen muss.
- Die Anzahl der ausgeführten Prozesse – Dieser Wert beschreibt die Anzahl der Prozesse, die derzeit existieren.
- Die Anzahl der Threads – Dieser Wert umfasst alle derzeit existierenden Threads, lässt jedoch keinen Rückschluss auf die Anzahl der Threads eines einzelnen Prozesses zu.

Das SystemMetaDataRecord Die Klasse `SystemMetaDataRecord` enthält weitere Daten über das aktuelle System. Dabei handelt es sich jedoch ausschließlich um Daten, die sich während der Laufzeit des Systems nicht verändern. Sie sind als Ergänzung der mit Hilfe des `SystemInformationRecords` gesammelten Daten anzusehen und können später bei der Auswertung als Kontextinformationen dienen. Im Einzelnen umfasst dies folgende Daten:

- Weitere Angaben über den Prozessor – Dazu zählen Informationen über den Hersteller des Prozessors, die Bezeichnung des Modells, die Geschwindigkeit in Hz und die Größe

des Cache sowie die Anzahl der physikalischen Sockets, der Prozessorkerne und der Prozessorkerne pro Socket. Die Anzahl der Prozessorkerne ist außerdem erforderlich, um die Prozessorauslastung eines Prozesses korrekt berechnen zu können.

- Angaben über die Gesamtgröße des Arbeitsspeichers – Dies umfasst die Größe des Arbeitsspeichers, auf den das System zugreifen kann.
- Angaben über das Betriebssystem – Dazu zählen der Hersteller des Betriebssystems, der Name, die Beschreibung und die Version.

Das ProcessSpecificInformationRecord Ein `ProcessSpecificInformationRecord` ist eine Erweiterung des `SystemInformationRecords`. Daher enthält es zum einen die Daten über die Auslastung der Systemressourcen, wie sie zuvor beschrieben sind, zum anderen aber auch Daten über die Auslastung und Nutzung der Ressourcen durch einen bestimmten Prozess. Die Entscheidung für die Zusammenfassung dieser Daten in einem Record-Typen beruht darauf, dass es im Verlauf der Analyse einfacher ist, wenn die Daten möglichst zusammenhängend vorliegen. Außerdem ist es wichtig, dass die Erhebung der Daten möglichst zeitgleich geschieht, um bei der Analyse auch Aussagen über gemeinsame Tendenzen treffen zu können. Folgende Daten sind neben den zuvor genannten des `SystemInformationRecords` in einem `ProcessSpecificInformationRecord` enthalten:

- Unterteilung der Prozessorauslastung – Diese Werte geben an, zu wie viel Prozent der Prozessor durch Benutzeraktionen oder Systemprozesse ausgelastet ist.
- Der Arbeitsspeicher des Prozesses – Dieser Wert beschreibt, wie viel Arbeitsspeicher für einen bestimmten Prozess verfügbar ist.
- Die Prozessorauslastung durch den Prozess – Dieser Wert gibt an, zu wie viel Prozent der überwachte Prozess den Prozessor auslastet.
- Die Anzahl der Datei-Deskriptoren – Dieser Wert sagt aus, auf wie viele Dateien der Prozess Zugriff hat.
- Die Anzahl der Threads des Prozesses – Dazu zählen alle Threads, die der überwachte Prozess abgespalten hat.

Das ProcessMetaDataRecord Die Klasse `ProcessMetaDataRecord` enthält ergänzende Informationen zu dem überwachten Prozess. Ähnlich wie bei dem `SystemMetaDataRecord` sind auch in diesem Fall nur Daten enthalten, die sich nicht zur Laufzeit ändern. Dies umfasst:

- Weitere Informationen zum Prozess – Dazu zählen die während der Ausführung auf dem System eindeutige Prozess-ID, der Name des Prozesses sowie die Kommandozeilenparameter.

- Angaben zur Umgebung des Prozesses – Dies umfasst zum einen das aktuelle Arbeitsverzeichnis, zum anderen den Namen der ausführbaren Datei.
- Informationen über den Benutzer – Zu diesen Daten zählen der Name des aktuell auf Betriebssystemebene angemeldeten Benutzers sowie dessen Benutzergruppe.

Entwicklung spezieller ISampler

Das Interface *ISampler* schreibt einzig die Methode `sample()` vor. Der `MonitoringController` ruft diese Methode periodisch in festen Zeitabständen auf. Im Folgenden sind Varianten zum Erheben von system- und prozessspezifischen Informationen gegeben.

Der SystemInformationPercSampler Der `SystemInformationPercSampler` erhebt in der Methode `sample()` unter Verwendung von SIGAR Informationen über die Auslastung der Systemressourcen und erzeugt daraus ein `SystemInformationRecord`. Außerdem erzeugt der `SystemInformationPercSampler` bereits beim Ausführen des Konstruktors ein zusätzliches `SystemMetaDataRecord`. Da sich die im `SystemMetaDataRecord` enthaltenen Daten zur Laufzeit nicht ändern, ist nur eine einzige Erhebung dieser Daten vorgesehen. Dies verhindert eine redundante Speicherung der Informationen und verringert die Menge an Daten, die bei einem Aufruf der Methode `sample()` zu verarbeiten sind.

Der ProcessSpecificPercSampler Der `ProcessSpecificPercSampler` erhebt in der Methode `sample()` ebenfalls unter Verwendung von SIGAR Daten über die Auslastung der Systemressourcen. In diesem Fall sind jedoch auch Informationen über die anteilige Ressourcennutzung von Lapap MK II enthalten. Daher erzeugt der `ProcessSpecificPercSampler` aus diesen Daten `ProcessSpecificInformationRecords`. Auch der `ProcessSpecificPercSampler` erzeugt zusätzlich Records, die ergänzende Informationen enthalten. Neben einem `SystemMetaDataRecord` handelt es sich in diesem Fall auch um ein `ProcessMetaDataRecord`. Beides erfolgt ebenfalls nur ein einziges Mal während der Ausführung des Konstruktors.

Entwicklung des ApplicationStartupAspects

Für die Verwendung der zuvor beschriebenen *ISampler*-Implementationen ist es erforderlich, diese unter Verwendung der Methode `schedulePeriodicSampler()` der Klasse `MonitoringController` zu registrieren [Kieker Project 2013]. Besonders für die Analyse des Startvorgangs von Lapap MK II ist es wichtig, die periodische Erhebung möglichst frühzeitig zu beginnen. Der Startvorgang erfolgt jedoch durch Ausführen einer `main()`-Methode aus dem Navigator-Framework heraus. Da der Quellcode des Navigator-Frameworks zum Zeitpunkt der Analyse nicht vorlag und somit keine direkte Implementation erfolgen konnte, war auch hierfür die Entwicklung eines aspektorientierten Ansatzes nötig.

Der `ApplicationStartupAspect` ist ein speziell entwickelter Aspekt, um das Registrieren eines *ISamplers* unmittelbar nach dem Start von Lapap MK II durchzuführen. Als Pointcut

diente ursprünglich direkt die Methode `NavigatorLauncher.main()`. Bei der Durchführung von ersten Tests hat sich jedoch herausgestellt, dass es dabei zu Wechselwirkungen zwischen Kieker und Log4j kommt, da zu dem Zeitpunkt seitens Lapap MK II noch keine Konfiguration von Log4j stattfinden konnte. Daraus resultierte, dass Lapap MK II keine Log-Dateien mehr erzeugen und Nachrichten hineinschreiben konnte. Ein Einfluss auf das übrige Verhalten von Lapap MK II war jedoch nicht erkennbar. Um beim Testen, insbesondere des Startvorgangs, auch die Log-Dateien auswerten zu können, war es unerlässlich, eine korrekte Konfiguration von Log4j zu ermöglichen. Aus diesem Grund erfolgte eine Änderung des Pointcuts auf `NavigatorLauncher.configureLog4j()`, was dazu führte, dass das periodische Erheben der Daten erst beginnt, wenn die Konfiguration von Log4j abgeschlossen ist.

3.4. Entwicklung der Testprozeduren

Um das in Unterabschnitt 3.1.3 beschriebene Verhalten von Lapap MK II analysieren zu können, ist die Entwicklung eigener Testprozeduren notwendig. Die erhöhte Ladezeit für ein PDF-Dokument ließ sich bei der Durchführung der Testprozedur 5 aus dem Testszenario IX (siehe Unterabschnitt 3.1.2) beobachten. Diese Testprozedur ist mit fast 80 Schritten jedoch zu umfangreich, um das Öffnen von PDF-Dokumenten gezielt untersuchen zu können. Zudem sind viele Schritte enthalten, die sich nicht auf die Nutzung des Documentation-Workspaces oder das Arbeiten mit PDF-Dokumenten beziehen.

Der Startvorgang von Lapap MK II ist eine Anforderung, die bereits gegeben sein muss, bevor Tests auf Systemebene erfolgen können. Ein Test der einzelnen Schritte des Startvorgangs wäre somit auf Produkt-Ebene durchzuführen. Dementsprechend war eine Testprozedur für den Startvorgang auf Systemebene bisher nicht erforderlich.

3.4.1. Testprozedur für das Öffnen von PDF-Dokumenten

Die hier entwickelte Testprozedur enthält einzelne Schritte der Testprozedur 5 aus dem Testszenario IX [TS IX]. Das beschriebene Verhalten trat bei der Durchführung der Testprozedur 5 in Schritt 50 auf, daher sind insbesondere die zuvor ausgeführten Schritte von Interesse. Einige dieser Schritte behandeln unter anderem das Ausführen einer Checklisten-Prozedur, was jedoch keinen Zusammenhang mit der Verarbeitung von PDF-Dokumenten aufweist. Daher hat das Suchen und Öffnen einer PDF-Prozedur diese Schritte ersetzt. Um die Nutzung des Documentation-Workspaces zu intensivieren, sind weitere Schritte hinzugekommen, die das Öffnen verschiedener PDF-Dokumente vorsehen. Einige dieser Schritte waren in der Testprozedur 5 nicht oder erst zu einem späteren Zeitpunkt vorgesehen. Folgende Bereiche sind im Einzelnen durch die neue Testprozedur abgedeckt:

- Verwenden der Suchfunktion – Die Suchfunktion ist in mehreren Workspaces anwendbar. Das Anzeigen der Ergebnisse erfolgt jedoch immer innerhalb des Documentation-

Workspaces. Als Ziel der Suche dienen hier der Name eines Displays sowie die Bezeichnung einer PDF-Prozedur.

- Öffnen von PDF-Dokumenten – Als zentraler Punkt der Analyse ist das Öffnen von verschiedenen PDF-Dokumenten sowie das Anzeigen einer PDF-Prozedur ein wesentlicher Bestandteil der Testprozedur.
- Weitere PDF-bezogene Aktionen – Dazu zählen Aktionen, die die Dokumente direkt betreffen, wie das Scrollen oder das Skalieren eines Dokuments.
- Navigieren im `WebRenderer` – Dies betrifft vor allem die Verwendung der Zurück-Schaltfläche sowie die Links auf der Startseite des Documentation-Workspaces und des RefDoc-Links (siehe Abbildung 3.2).

Die Testprozedur besteht aus zwei Teilen, denen unterschiedliche Konfigurationen von Lapap MK II zugrunde liegen. Die durchzuführenden Schritte sind jeweils identisch. Der erste Teil sieht vor die Standardeinstellungen von Lapap MK II zu verwenden. Der zweite Teil hingegen legt seinen Schwerpunkt auf die Analyse der Ressourcennutzung seitens Lapap MK II, insbesondere auf den Speicherbedarf. Zu diesem Zweck erfolgt eine Anpassung der Konfiguration dahingehend, dass der Wert des maximal von Lapap MK II nutzbaren Heap-Speichers 64 Megabyte beträgt, was den minimalen Anforderungen entspricht. Die einzelnen Schritte der Prozedur sowie Anmerkungen zu den zu erwarteten Ergebnissen sind in Abschnitt A.1 enthalten.

3.4.2. Testprozedur für den Startvorgang von Lapap MK II

Aus Sicht des Anwenders handelt es sich beim Starten von Lapap MK II nur um einen einzelnen Schritt, weshalb hier der Eindruck entstehen kann, dass es an dieser Stelle keiner formalen Testprozedur bedarf. Neben den einzelnen Schritten sind aber auch zu erwartende Ergebnisse oder Zustände in der Prozedur aufgeführt. Für den Startvorgang von Lapap MK II ist es an dieser Stelle besonders wichtig zu definieren, wann dieser als abgeschlossen anzusehen ist. Außerdem beschreibt die Prozedur weitere nachbereitende Schritte und listet alle Dateien auf, die nach einem Durchgang zu sichern sind.

Die Testprozedur für den Startvorgang besteht ebenfalls aus zwei Teilen. In diesem Fall ist jedoch nicht die Konfiguration ausschlaggebend für die Differenzierung, sondern das Verhalten von Lapap MK II während des Startvorgangs. Wie bereits in Unterabschnitt 3.1.3 erläutert, unterscheidet sich der erste Startvorgang von Lapap MK II dahingehend von allen weiteren, das zusätzlich das Laden von Daten aus dem Netzwerk notwendig ist. Dies führt dazu, dass der erste Startvorgang deutlich mehr Zeit in Anspruch nimmt als alle folgenden. Der erste Teil der Testprozedur umfasst daher ausschließlich den Neustart von Lapap MK II. Der erste Start im Anschluss an den Bootvorgang des Laptops ist hier nicht relevant. Der zweite Teil der Prozedur dient der Analyse der Unterschiede im Verhalten zwischen dem ersten und allen folgenden Startvorgängen. Dementsprechend ist in diesem

Fall ausschließlich der erste Startvorgang relevant. Dies hat zur Folge, dass für jeden Durchgang der Prozedur ein Neustart des Laptops notwendig ist. Die einzelnen Schritte der Prozedur sowie Anmerkungen zu den zu erwarteten Ergebnissen sind in Abschnitt A.2 aufgeführt.

Kapitel 4

Durchführung der Analyse

Dieses Kapitel enthält eine Zusammenfassung der Durchführung der Testprozeduren und gibt einen Überblick über Auffälligkeiten, die dabei aufgetreten sind. Abschnitt 4.1 beschreibt die Software- und Hardware-Ausstattung des verwendeten Laptops sowie die Konfiguration der SITE, an der die Durchführung der Tests erfolgte. In Abschnitt 4.2 ist die Erhebung von Referenzdaten für die Auslastung der Ressourcen erläutert. Abschließend sind die einzelnen Durchführungen der Testprozeduren sowie dabei aufgetretene Auffälligkeiten in Abschnitt 4.3 beschrieben.

4.1. Konfiguration der Testumgebung

Die Durchführung der Tests erfolgte unter Verwendung eines Lenovo T61p, dessen Ausstattung den aktuell im Columbus-Modul verwendeten On-Board-System-Laptops entspricht. Der Laptop verfügt über einen Intel Core™ 2 Duo T7700 Doppelkern-Prozessor mit einer Taktfrequenz von 2,4 GHz sowie 4 Gigabyte Arbeitsspeicher.

Vor Beginn der Tests erfolgte eine Erneuerung jeglicher Software des Laptops durch die Installation des Columbus-System-Laptop-Image (CSLI) V5.1.1. Das Image enthält als Betriebssystem SuSE Linux Enterprise Desktop 10 sowie alle übrigen Anwendungen eines On-Board-System-Laptops einschließlich einer Version von Lapap MK II. Um während der Tests die in Unterabschnitt 3.3.2 aufgeführten Monitoringdaten erheben zu können, war es an dieser Stelle notwendig, die bereits vorhandene Version von Lapap MK II zu ersetzen. Bei der modifizierten Version handelt es sich um eine Variante von Lapap MK II V2.1.2pre mit einer Integration des Kieker Monitoring Frameworks. Der Laptop war als dritter On-Board-System-Laptop (laptop-3) konfiguriert, der Benutzername lautete entsprechend *astro-3*.

Die Durchführung der Testprozeduren erfolgte an der SITE unter Verwendung der Testanlage 1. Die Konfiguration der SITE entsprach den Vorgaben für Software Cycle 14. Es handelte sich dabei um die Version ASS 7.1.1, wie sie auch im Columbus-Modul zum Einsatz kommen soll.

4.2. Erzeugen von Referenzdaten

Die Analyse des Verhaltens von Lapap MK II erfolgt unter anderem durch die Auswertung von Informationen über die Nutzung der Systemressourcen. Um auch die Auslastung durch Lapap MK II auswerten zu können, enthalten die `ProcessSpecificInformationRecords` (siehe auch Unterabschnitt 3.3.2) bereits Informationen über die anteilige Ressourcennutzung seitens Lapap MK II. Zusätzlich ist es an dieser Stelle sinnvoll, das Verhalten des Systems zu analysieren, um die Ressourcennutzung seitens des Systems gegebenenfalls mit der von Lapap MK II vergleichen zu können. Zum Erheben dieser Daten erfolgte die Ausführung zweier weiterer Monitoring-Applikationen vor der Durchführung der Testprozeduren, die diese Referenzdaten erzeugen. Beide Anwendungen erheben ebenfalls Monitoringdaten mittels Kieker.

Die erste Anwendung setzt voraus, dass das Betriebssystem vollständig geladen ist. Um nur die Ressourcennutzung seitens des Systems ermitteln zu können, ist es hier notwendig, Lapap MK II vor Beginn der Ausführung zu schließen. Diese Anwendung erzeugt ausschließlich `SystemInformationRecords` mit einer Frequenz von vier Hz über einen Zeitraum von zehn Minuten. Auf diese Weise ist ausschließlich die Ressourcennutzung seitens des Systems und der generell ausgeführten Hintergrundprozesse erfasst.

Die zweite Anwendung setzt das Ausführen von Lapap MK II voraus. Der Startvorgang muss vollständig abgeschlossen und die Verbindung zum Data-Management-System hergestellt sein. Dieser Zustand gilt als erreicht, sobald das DMS-Icon im Status-Display mit einer grünen Hintergrundfarbe versehen ist und die Synoptic-Displays mit Werten gefüllt sind. Die Anwendung erzeugt in diesem Fall `ProcessSpecificInformationRecords` über einen Zeitraum von zehn Minuten mit einer Frequenz von vier Hz. Während der Ausführung der Anwendung findet keinerlei Interaktion mit Lapap MK II statt. Die erzeugten Monitoringdaten zeigen ausschließlich die Ressourcenauslastung durch das Betriebssystem, Lapap MK II sowie aller generell ausgeführten Hintergrundprozesse.

4.3. Durchführung der Testprozeduren

In diesem Abschnitt ist die Durchführung der einzelnen Testprozeduren beschrieben. Die Durchführung jeder Prozedur erfolgte mehrfach. In den folgenden beiden Abschnitten sind daher Auffälligkeiten aufgeführt, die bei der Durchführung der in Abschnitt 3.4 entwickelten Testprozeduren aufgetreten sind.

4.3.1. Testprozedur für das Öffnen von PDF-Dokumenten

Dieser Abschnitt beschreibt die Durchführung der in Unterabschnitt 3.4.1 definierten Testprozedur für das Öffnen von PDF-Dokumenten. Dies umfasst sowohl die Ausführung unter Verwendung der Standardkonfigurations als auch mit reduziertem Heap-Speicher.

Tabelle 4.1. Anmerkungen zu Auffälligkeiten bei einzelnen Durchführungen der Prozedur für das Laden eines PDF-Dokuments mit Standardkonfiguration

Durchgang	Schritt	Anmerkung
2	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
3	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
6	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
9	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
10	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
12	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
13	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
15	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
16	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
18	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
19	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
20	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
23	10	Das Dokument war sofort sichtbar
	11	Stark verzögerte Reaktion
25	10	Das Dokument war sofort sichtbar
	11	Stark verzögerte Reaktion

Standardkonfiguration von Lapap MK II

Unter Verwendung dieser Konfiguration erfolgten insgesamt 25 Durchführungen der Prozedur. In mehreren Durchgängen trat hierbei eine erhöhte Ladezeit für ein PDF-Dokument auf. Die Ladezeit gilt als erhöht, wenn mehr als 30 Sekunden vergangen sind, bevor das Dokument sichtbar war. In Tabelle 4.1 sind Anmerkungen zu einzelnen Durchgängen und Schritten dieses Tests aufgeführt. In vier Fällen kam es vor, dass Lapap MK II direkt nach dem Start keine Verbindung zum Data-Management-System aufbauen konnte. In diesen Fällen war ein Neustart des Laptops notwendig, um die Verbindung wiederherstellen und mit dem Test fortfahren zu können. Daher erfolgte beim Auftreten dieses Fehlers eine Wiederholung des Durchgangs.

Reduzierter Heap-Speicher

Für diesen Test fand eine Anpassung der Konfiguration von Lapap MK II statt, die den maximal zulässigen Heap-Speicher auf 64 Megabyte begrenzt. Auch mit dieser Konfiguration erfolgten insgesamt 25 Durchläufe der Testprozedur. In allen Fällen war hier eine erhöhte Ladezeit für das PDF-Dokument in Schritt zehn notwendig. Weitere Auffälligkeiten sind in Tabelle 4.2 zusammengefasst. Auch während der Durchführung dieser Testprozedur kam es in vier Fällen vor, dass es nicht mehr möglich war, nach dem Neustart von Lapap MK II eine Verbindung zum Data-Management-System aufzubauen. Hier erfolgte jeweils ein

Tabelle 4.2. Anmerkungen zu Auffälligkeiten bei einzelnen Durchführungen der Prozedur für das Laden eines PDF-Dokuments mit reduziertem Heap-Speicher

Durchgang	Schritt	Anmerkung
1	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
2	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
3	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
4	2	Erhöhte Bearbeitungszeit für den Suchvorgang
	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
5	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
6	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
7	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
8	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
9	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
10	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
11	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
12	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
13	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
14	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
15	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
	11	Stark verzögerte Reaktion
16	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
17	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
18	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
	11	Stark verzögerte Reaktion
19	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
	11	Stark verzögerte Reaktion
20	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
21	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
	11	Stark verzögerte Reaktion
22	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
23	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
24	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)
25	10	Erhöhte Ladezeit für das Dokument (> 30 Sekunden)

Neustart des Laptops und eine Wiederholung des Durchgangs.

Das Arbeiten mit Lapap MK II hat sich generell als zeitaufwendiger erwiesen, wenn der Heap-Speicher auf 64 Megabyte reduziert ist. Auch wenn dieser Wert als minimale Voraussetzung für das Ausführen von Lapap MK II angegeben ist, hat sich gezeigt, dass unter diesen Bedingungen vermehrt ein *OutOfMemoryError* auftritt. Um dies zu vermeiden, war es notwendig mehr Zeit zwischen der Ausführung einzelner Schritte verstreichen zu lassen, als es mit der Standardkonfiguration der Fall war. Beim Auftreten eines `OutOfMemoryErrors` erfolgte ebenfalls eine Wiederholung des Durchgangs.

4.3.2. Testprozedur für den Startvorgang von Lapap MK II

Dieser Abschnitt beschreibt die Durchführung der in Unterabschnitt 3.4.2 definierten Testprozedur für den Startvorgang von Lapap MK II. Dies umfasst sowohl den ersten Startvorgang als auch den Neustart.

Neustart von Lapap MK II

Zwecks Analyse des Startvorgangs erfolgten insgesamt 30 Durchläufe der Testprozedur. Der Neustart von Lapap MK II lief durchweg zügig und ohne größere Probleme ab. Lediglich das bereits zuvor beobachtete Phänomen, dass es nicht möglich war, nach einem Neustart von Lapap MK II erneut eine Verbindung zum Data-Management-System aufzubauen, trat auch hier mehrfach auf. Wie auch zuvor war hier ein Neustart des Laptops erforderlich und es fand eine Wiederholung des Durchgangs statt. Eine Zusammenfassung dieser Ereignisse ist in Tabelle 4.3 gegeben.

Tabelle 4.3. Anmerkungen zu Auffälligkeiten bei einzelnen Durchführungen der Prozedur für den Neustart von Lapap MK II

Durchgang	Schritt	Anmerkung
8	2	Kein Aufbau der Verbindung zum DMS möglich
15	2	Kein Aufbau der Verbindung zum DMS möglich
22	2	Kein Aufbau der Verbindung zum DMS möglich
28	2	Kein Aufbau der Verbindung zum DMS möglich

Neustart des Laptops

Die Durchführung dieser Testprozedur und damit der Neustart des Laptops erfolgte ebenfalls insgesamt 30 mal. Dieser Teil der Tests war besonders langwierig. Neben dem Starten des Betriebssystems war es auch der erste automatische Startvorgang von Lapap MK II, der besonders viel Zeit in Anspruch nahm. Bedingt durch das Nachladen der Configuration-Data-Items waren hierfür in der Regel etwa 120 Sekunden nötig. Auffälligkeiten sind bei diesem Test jedoch keine aufgetreten.

Kapitel 5

Auswertung

Bei der Durchführung der Prozeduren sind Monitoringdaten entstanden, deren Aufbereitung und Auswertung in diesem Kapitel erfolgt. Abschnitt 5.1 beschreibt die Analyse der Daten bezüglich der Verzögerung beim Öffnen von PDF-Dokumenten. Dies umfasst sowohl die Durchführung der Prozedur mit Standardkonfiguration als auch mit reduziertem Heap-Speicher. In Abschnitt 5.2 erfolgt die Auswertung der Daten bezüglich des Startvorgangs von Lapap MK II. Hier ist neben dem Neustart von Lapap MK II auch der erste Start im Anschluss an den Bootvorgang beschrieben. Abschließend sind in Abschnitt 5.3 weitere Ergebnisse aufgeführt, die das Problem des Verbindungsaufbaus zum Data-Management-System betreffen, das während der Durchführung mehrfach aufgetreten ist.

5.1. Verhalten beim Öffnen eines PDF-Dokuments

Dieser Abschnitt beschreibt die Auswertung der Monitoringdaten über das Verhalten von Lapap MK II beim Öffnen eines PDF-Dokuments. Die Auswertung erfolgt in Hinsicht auf zwei Aspekte. Zum einen handelt es sich dabei um die Ausführungszeiten der Methoden sowie die Reaktionszeit von Lapap MK II, zum anderen um die Auslastung der Ressourcen. Diese Aspekte sind in den folgenden beiden Abschnitten für je eine der zwei Konfigurationsvarianten beschrieben.

Die Testprozedur aus Unterabschnitt 3.4.1 lädt bei jeder Durchführung mehrere HTML- und PDF-Dokumente. Jeder Durchgang gibt Aufschluss über zwei Ladevorgänge eines HTML-Dokuments sowie vier Ladevorgänge eines PDF-Dokuments. Die einzelnen Schritte der Prozedur sind in Abschnitt A.1 aufgeführt. Bedingt durch weitere Schritte, wie etwa die Navigation mittels der Zurück-Schaltfläche, sind in den Daten weitere Ladevorgänge enthalten. In diesen Fällen erfolgte jedoch kein Aufruf der Methode `onMouseDown()`. Für die Auswertung der Ausführungszeiten ergibt sich daher folgende Grundlage: Der Aufruf der Methoden `onURLChange()` und `onDocumentComplete()` erfolgte in jedem Durchlauf insgesamt 13 mal, der Aufruf der Methode `onLoadIntercept()` 14 mal. Für die Methode `onMouseDown()` ergeben sich sechs Aufrufe. Die im Folgenden aufgeführten statistischen Daten, wie etwa das Maximum, beziehen sich auf 25 Durchführungen der Prozedur und entsprechend auf 325, 350 beziehungsweise 150 Methodenaufrufe.

5.1.1. Standardkonfiguration von Lapap MK II

Hier ist die Auswertung der Daten dargelegt, deren Erhebung unter Verwendung der Standardkonfiguration von Lapap MK II erfolgte.

Ausführungszeiten der Methoden und Reaktionszeiten

Bei der Durchführung der Testprozedur aus Unterabschnitt 3.4.1 sind in einigen Durchgängen in Schritt zehn mehr als 30 Sekunden vergangen, bevor das PDF-Dokument sichtbar war (siehe Tabelle 4.1). Dies führt zunächst zu der Annahme, dass sich dieses Verhalten in den Ausführungszeiten der aufgerufenen Methoden widerspiegelt. Tabelle 5.1 zeigt statistische Werte für die Ausführungszeiten der überwachten und in Lapap MK II implementierten Methoden in Millisekunden. Hier ist ersichtlich, dass die Ausführungszeit der Methode `onMouseDown()` bei allen Durchführungen weniger als eine Millisekunde betrug. Dieses Verhalten entspricht an dieser Stelle den Erwartungen. Die Methode `onMouseDown()` führt keine weiteren Aktionen aus und verursacht entsprechend kaum Arbeitsaufwand. Ein ähnliches Bild zeigen die Ausführungszeiten der Methode `onURLChange()`. Bis auf den Maximalwert von 1,18 Millisekunden liegen die Ausführungszeiten hier ebenfalls bei weniger als einer Millisekunde.

Anders verhalten sich die Methoden `onLoadIntercept()` und `onDocumentComplete()`. Zwar zeigt `onLoadIntercept()` einen minimalen Wert von deutlich weniger als einer Millisekunde, der Durchschnitt und der Median sagen aber aus, dass an dieser Stelle in der Regel 0,6 bis 0,9 Millisekunden nötig sind. Die Werte für die Methode `onDocumentComplete()` weisen ausnahmslos höhere Ausführungszeiten von etwa 30 Millisekunden auf. Damit ist dies die aufwendigste der vier Methoden. Der Maximalwert von 157,93 Millisekunden zeigt, dass die Ausführungszeit an dieser Stelle stark nach oben abweichen kann.

Mittels der Werte in Tabelle 5.1 ist es möglich, einen pessimistischen Durchgang zu konstruieren, dessen Ausführungszeit aus der Summe der Maximalwerte der vier Methoden folgt. Ein solcher Durchgang benötigt demnach 172,09 Millisekunden für die Ausführung aller vier Methoden. Bezogen auf eine Reaktionszeit von 30 Sekunden macht dies jedoch einen Anteil von lediglich 0,57% aus. Daraus folgt, dass die Ausführungszeiten der einzelnen Methoden allein keinesfalls ausreichend sind, um Reaktionszeiten von mehr als 30

Tabelle 5.1. Statistische Werte für die Ausführungszeiten der Methoden beim Öffnen von HTML- und PDF-Dokumenten in Millisekunden unter Verwendung der Standardkonfiguration

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
DocPlugin	onMouseDown()	0,00	0,07	0,01	0,01
	onLoadIntercept()	0,19	12,91	0,93	0,63
	onDocumentComplete()	21,97	157,93	30,04	28,62
DocHistoryListener	onURLChange()	0,00	1,18	0,08	0,04

Tabelle 5.2. Statistische Werte für die Reaktionszeiten beim Öffnen von HTML- und PDF-Dokumenten in Millisekunden unter Verwendung der Standardkonfiguration

Dokumententyp	Schritt	Dateigröße	Min	Max	$\bar{x}$	$\tilde{x}$
HTML	7	11,06 KiB	126,25	443,55	306,36	323,81
	8	4,94 KiB	90,33	425,31	238,62	242,41
PDF	10	3125,78 KiB	60,82	1841,39	264,59	159,31
	15	360,65 KiB	80,86	235,89	142,74	148,47
	17	661,43 KiB	57,45	276,88	157,80	148,79
	19	95,93 KiB	71,80	229,65	144,91	147,82

Sekunden zu erklären.

Tabelle 5.2 zeigt statistische Werte für die gemessenen Reaktionszeiten von Lapap MK II. Alle sechs Ladevorgänge sind hier entsprechend der Schritte der Testprozedur gesondert aufgeführt. Gemäß den Erläuterungen in Unterabschnitt 3.3.1 ist die Reaktionszeit von Lapap MK II definiert als die Zeitspanne von Beginn der Ausführung der Methode `onMouseDown()` bis zum Ende der Ausführung der Methode `onDocumentComplete()`. Im Vergleich zu den Ausführungszeiten der einzelnen Methoden in Tabelle 5.1 sind die durchschnittlichen Reaktionszeiten um ein Vielfaches höher. Dies ist darauf zurückzuführen, dass der `WebRenderer` beim Öffnen eines Dokuments nicht nur die genannten Methoden innerhalb von Lapap MK II ausführt, sondern auch interne Aktionen. Wie bereits erläutert, war eine detaillierte Überwachung der Abläufe innerhalb des `WebRenderers` bisher nicht möglich.

Das Öffnen der HTML-Dokumente in Schritt sieben und acht der Prozedur verlief ohne Auffälligkeiten. In allen 25 Durchführungen war in etwa die gleiche Zeit nötig, bis Lapap MK II das Dokument angezeigt hat. Der Durchschnitt und der Median für Schritt sieben und acht bestätigen diesen Eindruck. Zudem sind bisher keinerlei Probleme mit Verzögerungen im Zusammenhang mit dem Öffnen von HTML-Dokumenten bekannt. Daher war dieses Verhalten zu erwarten.

Für das Öffnen von PDF-Dokumenten ergibt sich ein anderes Bild. Bezüglich der Anmerkungen in Tabelle 4.1 wären für die Reaktionszeit von Lapap MK II in Schritt zehn deutlich höhere Werte zu erwarten gewesen, als es bei den HTML-Dokumenten zuvor der Fall war. Wie Tabelle 5.2 zeigt, trifft dies jedoch nicht zu. Die durchschnittliche Reaktionszeit in Schritt zehn betrug gerade einmal 264,59 Millisekunden. Die übrigen Werte für die Schritte 15, 17 und 19 bestätigen diese Größenordnung. Besonders zu beachten ist an dieser Stelle, dass die Werte für den Durchschnitt und den Median für PDF-Dokumente in beinahe allen Fällen kleiner sind als entsprechende Werte für HTML-Dokumente. Dies widerspricht jedoch den bei der Durchführung wahrgenommenen Verhalten.

Der Maximalwert von 1841,39 Millisekunden, der in Durchgang 25 aufgetreten ist, stellt bezüglich der übrigen in Schritt zehn gemessenen Werte eine Punktanomalie dar, ebenso wie der Wert von 740,73 Millisekunden aus Durchgang 15. Die verbleibenden Werte waren

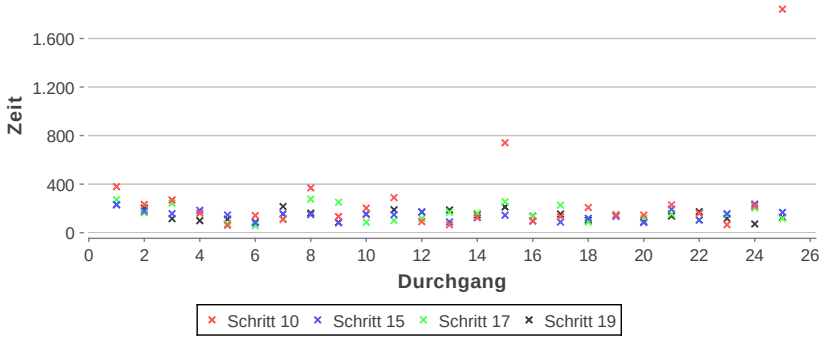

Abbildung 5.1. Reaktionszeiten für das Öffnen der PDF-Dokumente in den Schritten 10, 15, 17 und 19 der Prozedur mit Standardkonfiguration in den einzelnen Durchführungen

stets kleiner als 400 Millisekunden. Abbildung 5.1 zeigt die Reaktionszeiten der einzelnen Schritte. Sowohl in Durchgang 15 als auch in Durchgang 25 waren die Reaktionszeiten der übrigen Ladevorgänge unauffällig. Tabelle 4.1 zufolge trat in Durchgang 15 in Schritt 10 eine erhöhte Ladezeit von mehr als 30 Sekunden auf. Die Ausführungszeiten der überwachten Methoden in diesem Durchgang entsprachen aber etwa dem Durchschnitt. Einzig die Methode `onLoadIntercept()` weist mit 1,68 Millisekunden einen überdurchschnittlich hohen Wert auf. Bezogen auf eine Reaktionszeit von 30 Sekunden machen die vier Methoden mit zusammen 27,18 Millisekunden aber nur einen Anteil von 0,09% aus.

Für die maximale Reaktionszeit aus Durchgang 25 wäre zu erwarten, dass ebenfalls eine erhöhte Ladezeit festgehalten ist. Wie in Tabelle 4.1 aufgeführt war aber das Gegenteil der Fall, da Schritt zehn in diesem Durchgang als besonders schnell aufgefallen ist. Eine Verzögerung ist hier jedoch für Schritt 11 aufgeführt. In diesem Fall sind die Ausführungszeiten der Methoden `onLoadIntercept()` mit 1,18 Millisekunden und `onURLChange()` mit 0,14 Millisekunden überdurchschnittlich hoch. Mit zusammen 27,78 Millisekunden für alle vier Methoden entspricht dies ebenfalls einem Anteil von 0,09% und liegt damit in der gleichen Größenordnung wie in Durchgang 15. Daraus folgt, dass die Ursache der Verzögerungen in diesen Durchgängen nicht durch die überwachten Methoden gegeben ist.

Überraschend ist an dieser Stelle auch das Verhältnis der Median- und Durchschnittswerte zueinander bezüglich der Dateigröße. Obwohl sich die Dokumente zum Teil deutlich in ihrer Größe unterscheiden, spiegelt sich dieses Verhältnis nicht in den Reaktionszeiten wider. Insgesamt zeigen die Reaktionszeiten für PDF-Dokumente in Tabelle 5.2, dass es in dem überwachten Abschnitt sowohl von Lapap MK II als auch seitens des `WebRenderers` möglich ist, den Ladevorgang zügig auszuführen. Sie zeigen aber auch, dass die intern gemessene Zeit nicht der tatsächlich durch den Anwender wahrgenommenen Reaktionszeit entspricht. In Abbildung 5.2 ist die Aufteilung der Zeit vom Auslösen eines Klick-Ereignisses bis zum Anzeigen des Dokuments dargestellt. Zwecks Übersichtlichkeit zeigt Abbildung 5.2 jedoch

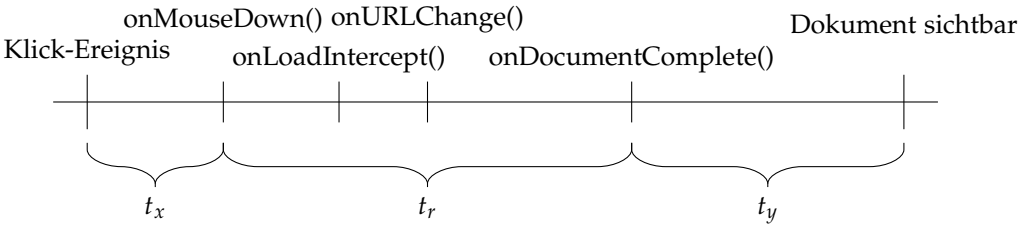

Abbildung 5.2. Aufteilung der Zeitspanne für das Laden eines Dokuments (nicht proportional)

keine echten Proportionen. Die mittels Kieker gemessene Zeit entspricht hier dem Abschnitt t_r. Die Abschnitte t_x und t_y sind durch die aufgezeichneten Messwerte nicht bestimmbar. Zusammen machen sie jedoch, bezogen auf eine Reaktionszeit von 30 Sekunden, einen Anteil von mehr als 90% aus.

Bezüglich Ziel Z1 lässt sich an dieser Stelle bereits festhalten, dass die innerhalb von Lapap MK II implementierten Methoden nicht als Ursache für die Verzögerungen in Betracht kommen. Zwar hat sich gezeigt, dass im Falle einer vom Anwender wahrgenommenen Verzögerung auch überdurchschnittlich hohe Ausführungszeiten der einzelnen Methoden auftreten können, diese aber dennoch nur einen geringen Anteil der wahrgenommenen Reaktionszeit ausmachen. Zudem hat sich ergeben, dass sich eine maximale Reaktionszeit wie in Durchgang 25 nicht zwangsläufig auf die Wahrnehmung des Anwenders auswirken muss.

Auslastung der Ressourcen

Das Muster der Ressourcenauslastung ist in allen Durchgängen nahezu identisch. Für den Prozessor lassen sich stets drei Bereiche identifizieren. Eine hohe Auslastung zu Beginn der Durchführung mit Spitzenwerten von über 90%, gefolgt von einer ruhigen Phase, in der nur vereinzelt Werte über 40% auftreten. Im letzten Drittel zeigen sich wieder vermehrt Werte von mehr als 40% für die Gesamtauslastung. Dieses Muster ist in Abbildung 5.3 an einem Auszug aus dem neunten Durchgang dargestellt. Neben der Auslastung des Prozessors im Ganzen ist auch die Auslastung durch Lapap MK II abgebildet. Hier ist ersichtlich, dass Lapap MK II den größten Teil der Prozessorauslastung verursacht. Nur an wenigen Punkten weicht die Auslastung durch Lapap MK II deutlich von der gesamten Auslastung ab. Diese Punkte befinden sich zumeist im ersten sowie im letzten Drittel.

Die Auslastung des Arbeitsspeichers im Ganzen sowie durch Lapap MK II ist ebenfalls in Abbildung 5.3 dargestellt. Hier zeigen sich jedoch keinerlei Auffälligkeiten. Zu Beginn der Prozedur steigt die Auslastung an. Im weiteren Verlauf bleibt sie aber weitgehend konstant. In allen Durchgängen lagen die Werte für Lapap MK II bei etwa 10%. Die Werte der Gesamtauslastung liegen meist nur wenige Prozentpunkte über denen von Lapap MK II und durchgehend unter 15%. Einzig im letzten Drittel ist bei einem Großteil der

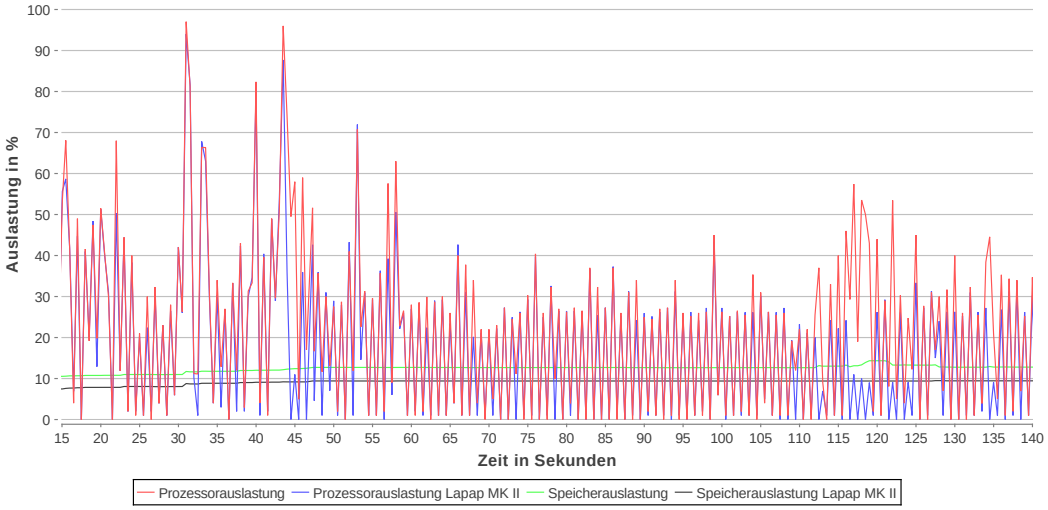

Abbildung 5.3. Auslastung des Prozessors und des Arbeitsspeichers im Ganzen sowie anteilig von Lapap MK II während der neunten Durchführung der Prozedur mit Standardkonfiguration

Durchführungen ein kurzzeitiger Anstieg erkennbar, der sich nicht in der Auslastung durch Lapap MK II widerspiegelt. Dieser fällt zumeist mit dem zuvor beschriebenen Anstieg der Prozessorauslastung zusammen. Abbildung 5.3 zeigt dieses Phänomen im Bereich von 110 bis 130 Sekunden.

Abbildung 5.4 zeigt zwei Ausschnitte aus dem Verlauf der Ressourcenauslastung während der neunten Durchführung der Prozedur. Zusätzlich sind hier die Bereiche grau markiert, die der gemessenen Reaktionszeit von Lapap MK II entsprechen. Sie beginnen zum Zeitpunkt des Aufrufs der Methode `onMouseDown()` und enden zum Zeitpunkt der Rückkehr aus dem Aufruf der Methode `onDocumentComplete()`. Zum Zeitpunkt des ersten Aufrufs der Methode `onMouseDown()`, bei etwa 57 Sekunden, zeigt sich eine Auslastung des Prozessors von mehr als 55%. Dies lässt zunächst auf einen Zusammenhang schließen. Zum Zeitpunkt des zweiten Aufrufs liegt jedoch nur eine Auslastung von 30% vor, die hier keine Auffälligkeit darstellt. In Durchgang neun ist in Schritt zehn eine erhöhte Ladezeit aufgetreten. Der zehnte Schritt der Prozedur entspricht dem dritten Ladevorgang in Abbildung 5.4 bei etwa 68 Sekunden. An dieser Stelle lässt sich aber lediglich ein Wert von etwa 35% für die Prozessorauslastung feststellen. Auch unter Berücksichtigung der Verzögerung von mehr als 30 Sekunden, bis das Dokument sichtbar war, zeigen sich nur wenige höhere Werte. Sie sind jedoch weiterhin kleiner als 50% und daher nicht als unregelmäßig anzusehen. Für die letzten drei Ladevorgänge lässt sich hier ein charakteristisches Verhalten identifizieren. Jeweils im Anschluss an den Aufruf der Methode `onDocumentComplete()` steigt die Auslastung des Prozessors an, ohne dass sich dieses Verhalten in der Auslastung

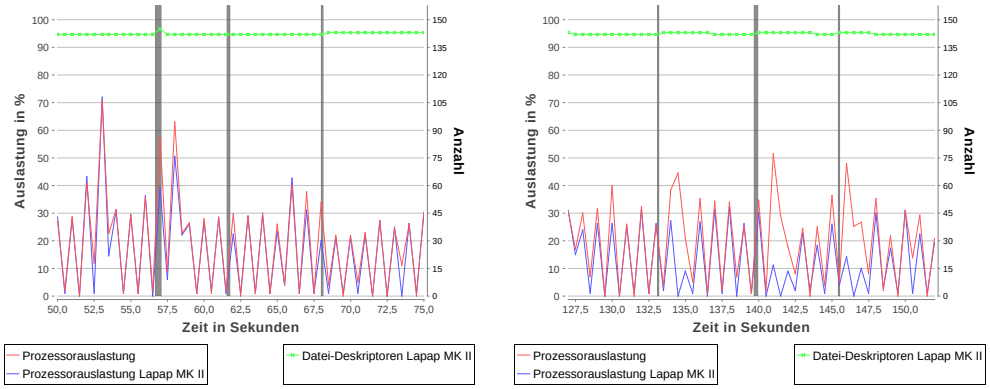

Abbildung 5.4. Ressourcenauslastung in den Bereichen vom Aufruf der Methode `onMouseDown()` bis zur Rückkehr aus dem Aufruf der Methode `onDocumentComplete()` während der neunten Durchführung mit Standardkonfiguration

durch Lapap MK II widerspiegelt. Da hier keine Verzögerungen aufgetreten sind, liegt die Vermutung nahe, dass die `Acroread`-Komponente diese Auslastungen beim Anzeigen der Dokumente verursacht hat.

Der Vergleich mit anderen Durchgängen zeigt, dass auch das Öffnen von HTML-Dokumenten während der ersten beiden Ladevorgänge nicht zu einer erhöhten Auslastung des Prozessors führen muss. In Abbildung 5.5 sind zum Vergleich die entsprechenden Ausschnitte aus Durchlauf fünf dargestellt. Keiner der drei Aufrufe in Abbildung 5.5 (a) hat hier eine erhöhte Auslastung verursacht. Nach den letzten drei Ladevorgängen in Abbildung 5.5 (b) kommt es auch jeweils zu einem Anstieg der Auslastung, der von der Auslastung durch Lapap MK II abweicht. Da sich dieses Phänomen auch in den übrigen Durchgängen zeigt, erhärtet sich der Verdacht, dass es sich an diesen Stellen um die `acroread`-Komponente handelt, die das Dokument anzeigt. Ein ähnliches Abweichen der Kurven ist auch in Abbildung 5.5 (a) bei etwa 125 Sekunden ersichtlich. In Durchgang fünf trat an dieser Stelle keine erhöhte Ladezeit auf. Daher könnte das Anzeigen des Dokuments zu diesem Zeitpunkt erfolgt sein. Für den neunten Durchlauf zeigt sich in Abbildung 5.3 ein ähnliches Phänomen bei etwa 112 Sekunden. Auch an dieser Stelle könnte es sich um das Anzeigen des Dokuments handeln, das bedingt durch die Verzögerung erst nach mehr als 40 Sekunden erfolgte.

Die erzeugten Referenzdaten (siehe Anhang C) zeigen, dass Lapap MK II in regelmäßigen Abständen von etwa einer Sekunde eine Auslastung des Prozessors erzeugt. Dies lässt sich durch das kontinuierliche Prüfen der Daten im Datenpool begründen, wie es in Unterabschnitt 2.4.3 erläutert ist. Dieses Verhalten zeigte sich in allen Durchgängen. Im mittleren Bereich schwanken die Werte zumeist zwischen 0% und etwa 30%.

Insgesamt hat der Test gezeigt, dass die Auslastung des Prozessors durch Lapap MK II weitestgehend unauffällig ist. Zwar treten vereinzelt Werte auf, die auch die 70%-

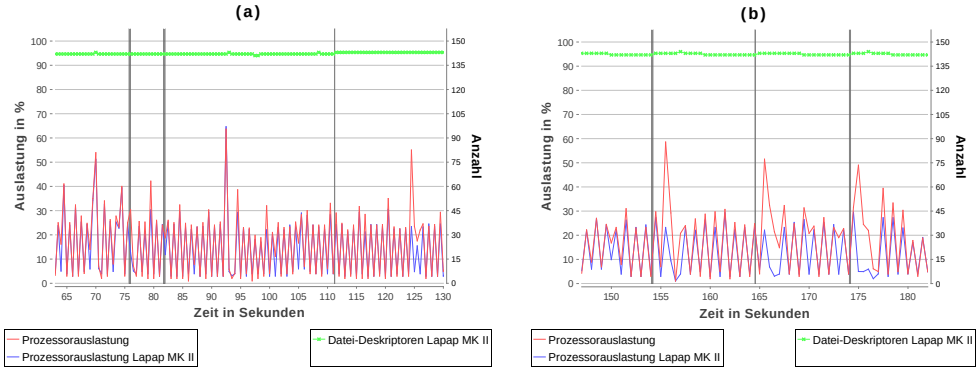

Abbildung 5.5. Prozessorauslastung, Anzahl der Datei-Deskriptoren sowie die Bereiche vom Aufruf der Methode `onMouseDown()` bis zur Rückkehr aus dem Aufruf der Methode `onDocumentComplete()` für Durchgang fünf

Marke überschreiten können, jedoch ist zu keinem Zeitpunkt eine anhaltende Auslastung über 90% festzustellen. Eine Überlastung ist daher nicht ersichtlich. Für das Verhalten beim Öffnen von PDF-Dokumenten hat sich ergeben, dass das Ausführen der innerhalb von Lapap MK II implementierten Methoden nicht zu einer erhöhten Auslastung führt. Ein sich häufig anschließender Anstieg der Auslastung könnte jedoch auf die `acroread`-Komponente zurückzuführen sein. Die Nutzung des Arbeitsspeichers war durchweg unauffällig. Bezüglich Ziel Z1 lässt sich feststellen, dass es sich während der Ausführung der überwachten Methoden von Lapap MK II nicht um eine Performance-Anomalie handelt.

5.1.2. Reduzierter Heap-Speicher

Dieser Abschnitt beschreibt die Auswertung der Daten, die während der Ausführung von Lapap MK II mit auf 64 Megabyte beschränktem Heap-Speicher entstanden sind.

Ausführungszeiten der Methoden und Reaktionszeiten

Bereits während der Durchführung der Tests hat sich gezeigt, dass Lapap MK II unter diesen Bedingungen deutlich langsamer reagiert und entsprechend mehr Zeit für die Durchführung nötig war. Davon betroffen waren alle Schritte der Testprozedur. Dementsprechend ist es nicht frappierend, dass sich dieses Verhalten auch in der Reaktionszeit für das Öffnen von Dokumenten widerspiegelt. In allen 25 Durchführungen unter diesen Bedingungen trat in Schritt zehn eine erhöhte Ladezeit von mehr als 30 Sekunden auf.

Tabelle 5.3 zeigt statistische Werte für die Ausführungszeiten der Methoden in diesem Szenario. Hier ist ersichtlich, dass die Maximalwerte höher sind als die entsprechenden Werte aus Tabelle 5.1, die unter Verwendung der Standardkonfiguration entstanden sind. Die Auswirkungen auf die einzelnen Methoden fallen jedoch unterschiedlich aus. Für

Tabelle 5.3. Statistische Werte für die Ausführungszeiten der Methoden beim Öffnen von HTML- und PDF-Dokumenten in Millisekunden mit reduziertem Heap-Speicher

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
DocPlugin	onMouseDown()	0,00	2,45	0,02	0,00
	onLoadIntercept()	0,18	379,03	7,22	0,30
	onDocumentComplete()	21,73	877,01	57,65	25,99
DocHistoryListener	onURLChange()	0,00	257,64	0,84	0,04

die Methode `onMouseDown()` sind die Ausführungszeiten nahezu identisch. Einzig der Maximalwert ist um ein Vielfaches erhöht. Mit Ausnahme des Maximalwertes von 257,64 Millisekunden gilt dies auch für die Methode `onURLChange()`, deren Median weiterhin 0,04 Millisekunden beträgt. Die Ausführungszeiten der Methode `onLoadIntercept()` weisen deutlichere Unterschiede gegenüber den Werten aus Tabelle 5.1 auf. Der Maximalwert ist mit 379,03 Millisekunden um mehr als das 29-fache größer. Mit 0,30 Millisekunden ist der Median hingegen nur etwa halb so groß wie zuvor unter Verwendung der Standardkonfiguration. Ein ähnliches Bild zeigen die Werte der Methode `onDocumentComplete()`. Der Maximalwert ist um mehr als das 5-fache größer, während der Median mit 25,99 Millisekunden um gut 9% kleiner ist. Die Minimalwerte der vier Methoden zeigen keine nennenswerten Unterschiede. An dieser Stelle wäre ein stärkerer Einfluss auf die Ausführungszeiten zu erwarten gewesen. Zwar sind die Maximalwerte um ein Vielfaches höher, dem Median zufolge war in mehr als der Hälfte der Aufrufe aber weniger Zeit erforderlich.

Die statistischen Werte für die Reaktionszeiten in diesem Szenario sind in Tabelle 5.4 dargestellt. Hier ist ersichtlich, dass sich das Reduzieren des Heap-Speichers nicht gleichermaßen auf alle Reaktionszeiten ausgewirkt hat. Der Ladevorgang des HTML-Dokuments in Schritt sieben benötigte dem Durchschnitt und dem Median zufolge unter diesen Bedingungen weniger Zeit als unter Verwendung der Standardkonfiguration. Besonders beachtenswert ist, dass dies auch für den Minimalwert gilt. Der Maximalwert hingegen entspricht mit 902,67 Millisekunden mehr als dem Doppelten des Wertes in Tabelle 5.2. Dies zeigt, dass hier Abweichungen nach oben möglich sind. Die Reaktionszeiten für den achten Schritt der Prozedur zeigen dies deutlicher. Der Median und der Durchschnitt sind hier etwa um das 1,4-fache größer als die Werte aus Tabelle 5.2. Der Maximalwert von 893,59 Millisekunden, der in Durchgang 25 aufgetreten ist, entspricht sogar etwas mehr als dem Doppelten. Einzig der Minimalwert war an dieser Stelle kleiner als unter Verwendung der Standardkonfiguration.

Für das Öffnen des PDF-Dokuments in Schritt zehn sind in allen 25 Durchführungen erhöhte Ladezeiten aufgetreten. Die Werte in Tabelle 5.4 bestätigen diesen Eindruck jedoch nicht. Mit Ausnahme des Durchschnitts, der leicht gestiegen ist, sind die übrigen Werte kleiner als die entsprechenden Werte in Tabelle 5.2. Bei den anderen Ladevorgängen in den Schritten 15, 17 und 19 zeigt sich ebenfalls ein gemischtes Bild. Die Maximalwerte sind hier zwar um fast das Vierfache größer, der Median hingegen ist nur in Schritt 15 leicht

Tabelle 5.4. Statistische Werte für die Reaktionszeiten beim Öffnen von HTML- und PDF-Dokumenten in Millisekunden mit reduziertem Heap-Speicher

Dokumententyp	Schritt	Dateigröße	Min	Max	$\bar{x}$	$\tilde{x}$
HTML	7	11,06 KiB	119,86	902,67	295,20	208,00
	8	4,94 KiB	83,89	893,59	351,48	344,11
PDF	10	3125,78 KiB	52,66	1048,70	283,61	145,83
	15	360,65 KiB	51,41	904,86	258,09	161,56
	17	661,43 KiB	52,40	1082,03	281,44	127,75
	19	95,93 KiB	50,07	909,52	243,25	121,30

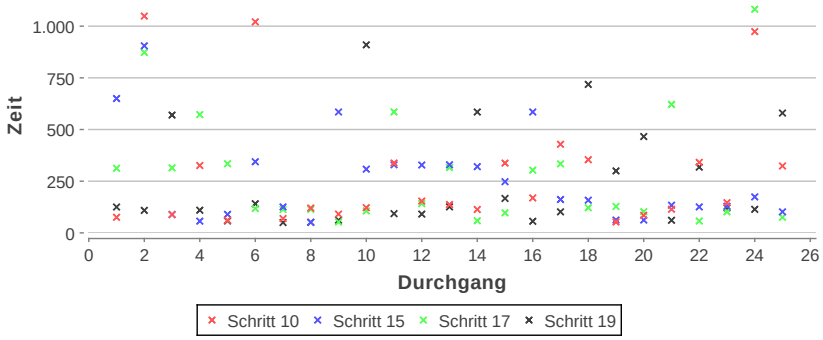

Abbildung 5.6. Reaktionszeiten für das Öffnen der PDF-Dokumente in den Schritt 10, 15, 17 und 19 der Prozedur in den einzelnen Durchführungen

gestiegen. Für die Schritte 17 und 19 ist dieser sogar gesunken. Wie Abbildung 5.6 belegt, lassen sich die Maximalwerte für die Reaktionszeit beim Öffnen eines PDF-Dokuments in diesem Fall aber nicht ohne Weiteres als Anomalien identifizieren. Der überwiegende Teil der Durchführungen benötigte ähnlich viel Zeit wie unter Verwendung der Standardkonfiguration. Abweichungen nach oben zeigen sich in diesem Szenario jedoch stärker und häufiger als zuvor. Ein besonders frappierendes Bild zeigen die Minimalwerte der Reaktionszeiten, da sie entgegen allen Erwartungen kleiner sind als unter Verwendung der Standardkonfiguration.

Dieser Test belegt, dass sich die Größe des verfügbaren Heap-Speichers auf die Geschwindigkeit von Lapap MK II auswirkt. Bereits während der Durchführung der Tests hat sich gezeigt, dass Lapap MK II generell mehr Zeit benötigt, um einzelne Schritte auszuführen. Zudem trat in allen Durchgängen eine erhöhte Ladezeit in Schritt zehn auf. Die Auswertung der Ausführungszeiten in Tabelle 5.3 belegt aber, dass es mit reduziertem Heap-Speicher nicht zwingend zu einer Verzögerung der vier Methoden kommen muss und eine zügige Ausführung weiterhin möglich ist. Die Werte in Tabelle 5.4 verdeutlichen, dass dies auch für die Reaktionszeiten gilt. Insgesamt bestätigen diese Werte das Ergebnis

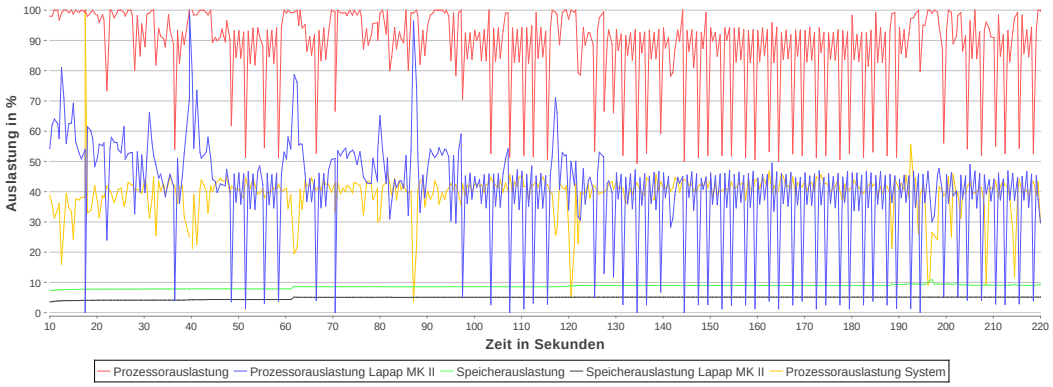

Abbildung 5.7. Auslastung des Prozessors und des Arbeitsspeichers im Ganzen sowie anteilig von Lapap MK II und die Auslastung des Prozessors durch das System für den vierten Durchgang der Prozedur mit reduziertem Heap-Speicher

aus dem Abschnitt zuvor, wonach eine erhöhte Ladezeit von mehr als 30 Sekunden in dem überwachten Abschnitt von Lapap MK II nicht auf ein Speicherproblem zurückzuführen ist.

Auslastung der Ressourcen

Für die Auslastung der Ressourcen zeigen sich in diesem Szenario zwei verschiedenartige Muster. Abbildung 5.7 zeigt den Verlauf der Prozessor- und Speicherauslastung für den vierten Durchgang. Besonders auffällig ist die hohe Auslastung des Prozessors, die durchgehend oberhalb von 50% liegt und regelmäßig Spitzenwerte von mehr als 90% aufweist. Die Auslastung durch Lapap MK II liegt hingegen überwiegend unterhalb von 55% und zeigt nur vereinzelt Spitzenwerte von mehr als 60%. Solch eine beachtliche Differenz ist zuvor nicht aufgetreten. Die in Abbildung 5.7 ebenfalls dargestellte Auslastung des Prozessors durch das System weist fast durchgehend Werte um die 40% auf. Unter Beachtung des reduzierten Heap-Speichers deutet dies darauf hin, dass dieses Verhalten auf vermehrtes *Swapping* zurückzuführen ist. Offenbar ist hier deutlich häufiger ein Austausch der Daten zwischen der Festplatte und dem Arbeitsspeicher notwendig als unter Verwendung der Standardkonfiguration. Die ebenfalls aufgezeichneten Werte über den Swap-Speicher geben diesbezüglich keine weiteren Anhaltspunkte. Die Auslastung des Speichers verläuft ebenso unauffällig wie unter Verwendung der Standardkonfiguration. Mit etwa 10% insgesamt und gut 5% für Lapap MK II liegen die Werte etwas niedriger als zuvor.

Aufgrund des reduzierten Heap-Speichers war durchaus vermehrtes Swapping und entsprechend eine höhere Auslastung des Prozessors zu erwarten. Gleichwohl zeigt sich dieses Muster nur in fünf Durchgängen der Prozedur. In den übrigen Durchläufen gleicht es etwa dem Muster, wie es in Abbildung 5.8 anhand des sechsten Durchgangs dargestellt ist. Die Auslastung des Prozessors verläuft ähnlich wie es in Abbildung 5.3 unter Verwen-

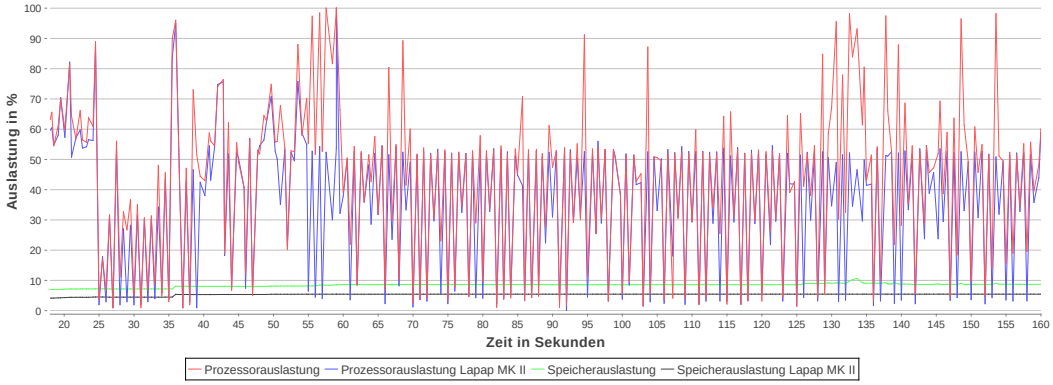

Abbildung 5.8. Auslastung von Prozessor und Arbeitsspeicher im Ganzen sowie anteilig von Lapap MK II für Durchgang sechs der Prozedur mit reduziertem Heap-Speicher

dung der Standardkonfiguration der Fall ist. Es lassen sich hier ebenfalls drei Bereiche identifizieren: Eine hohe Auslastung zu Beginn mit Spitzenwerten von mehr als 90%, ein ruhiger Abschnitt in der Mitte sowie vermehrtes Auftreten von Spitzenwerten im letzten Drittel. Ein Unterschied zeigt sich jedoch in der Höhe der Auslastung. Während sich die Werte in Abbildung 5.3 im mittleren Abschnitt zwischen 0% und 30% bewegen, schwanken sie in Abbildung 5.8 etwa zwischen 5% und 55%. Der Anstieg der durchschnittlichen Auslastung ist hier jedoch nicht überraschend und lässt sich durch vermehrtes Swapping begründen.

Für die Reaktionszeiten ergibt sich unter diesen Bedingungen ein ähnliches Bild wie unter Verwendung der Standardkofiguration. In Abbildung 5.9 sind die sechs Ladevorgänge aus dem sechsten Durchgang dargestellt. Für die ersten drei Bereiche zeigt sich, dass der Ladevorgang hier durchaus mit einem Anstieg der Prozessorauslastung einhergehen kann. Ein Vergleich mit anderen Durchgängen belegt aber auch hier, dass dies nicht grundsätzlich der Fall ist. Für die letzten drei Ladevorgänge ist das gleiche Phänomen ersichtlich wie auch zuvor unter Verwendung der Standardkonfiguration. Auch hier folgt im Anschluss an das Ausführen der Methode `onDocumentComplete()` ein Anstieg der Prozessorauslastung, der sich nicht in der Auslastung durch Lapap MK II widerspiegelt. Dies bestärkt die Vermutung, dass es sich dabei um das Anzeigen der Dokumente mittels der `acroread`-Komponente handelt. Für den dritten Ladevorgang lässt sich dieses Phänomen in diesem Szenario jedoch zumeist nicht eindeutig identifizieren wie Abbildung 5.8 verdeutlicht.

Dieser Test zeigt, dass sich das Reduzieren des Heap-Speichers auf die Auslastung des Prozessors auswirkt. In allen Durchgängen lagen die Werte der Prozessorauslastung über denen, die unter Verwendung der Standardkonfiguration entstanden sind. Auch wenn die Werte bei insgesamt fünf Durchführungen mit kontinuierlichen Spitzenwerten von über 95% besonders hoch waren, lässt sich an dieser Stelle eine Performance-Anomalie

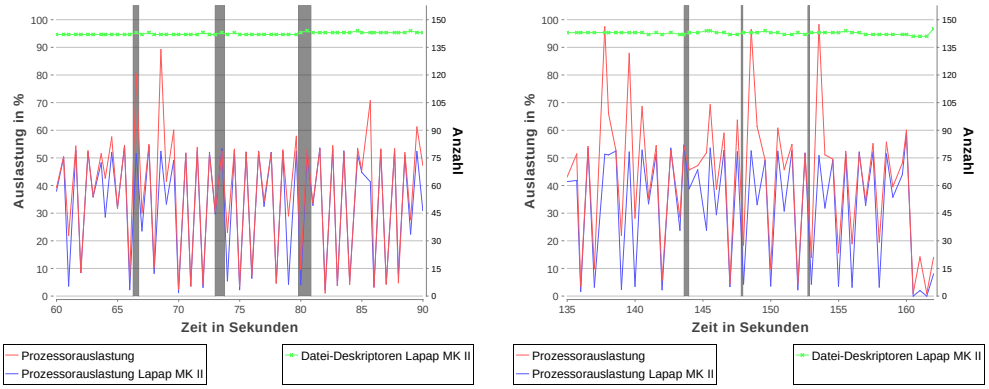

Abbildung 5.9. Auslastung des Prozessors im Ganzen sowie anteilig von Lapap MK II, Anzahl der Datei-Deskriptoren und die Bereiche vom Aufruf der Methode `onMouseDown()` bis zur Rückkehr aus dem Aufruf der Methode `onDocumentComplete()` für den sechsten Durchgang mit reduziertem Heap-Speicher

als Ursache für die Verzögerung beim Öffnen eines PDF-Dokuments ausschließen. Das Reduzieren des Heap-Speichers führt dazu, dass Lapap MK II im Allgemeinen langsamer reagiert. Verzögerungen von mehr als 30 Sekunden lassen sich dadurch jedoch nicht erklären, da die Auslastung durch Lapap MK II weiterhin nicht im kritischen Bereich liegt.

5.2. Verhalten während des Startvorgangs

Dieser Abschnitt enthält die Auswertung der Daten bezüglich des Verhaltens von Lapap MK II während des Startvorgangs. Die Auswertung erfolgt in Hinsicht auf zwei Aspekte. Dies betrifft zum einen die Ausführungszeiten einzelner Methoden, zum anderen die Auslastung der Ressourcen. In den folgenden beiden Abschnitten sind diese Aspekte jeweils für den Neustart von Lapap MK II sowie den ersten Start im Zuge des Startvorgangs des Laptops erläutert. Die aufgeführten statistischen Werte der Ausführungszeiten beruhen auf jeweils 30 Durchführungen der Prozedur. Bis auf wenige Ausnahmen, die gegebenenfalls gesondert genannt sind, erfolgt je Durchgang jeweils ein Aufruf jeder Methode.

5.2.1. Neustart von Lapap MK II

Die Auswertung für den Neustart von Lapap MK II ist im Folgenden in zwei Unterabschnitten beschrieben. Ein Neustart von Lapap MK II benötigt etwa 35 Sekunden. Der Vorgang gilt als abgeschossen, sobald das DMS-Icon im Status-Display durch seine grüne Hintergrundfarbe den erfolgreichen Aufbau der Verbindung zum DMS signalisiert und die Synoptic-Displays mit den aktuellen Werten aus dem Datenpool gefüllt sind.

Tabelle 5.5. Statistische Werte für die Ausführungszeiten der Methoden beim Initialisieren des Navigators in Millisekunden während des Neustarts von Lapap MK II

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
Launcher	main()	6614,35	7221,07	6977,82	6985,67
	run()	6612,81	7219,79	6976,42	6984,40
Navigator	main()	5588,07	6032,26	5820,49	5822,55
	init()	5587,88	6032,12	5819,83	5822,43
	loadPlugins()	4730,82	5083,23	4910,23	4921,85
	initNavigatorPluginIntegration()	499,28	639,48	582,99	582,29
	activatePlugins()	15,88	67,43	33,12	33,32
	performPluginInitialization()	397,09	539,42	486,36	490,62
	initLookAndFeel()	185,97	326,34	283,87	298,54

Ausführungszeiten der Methoden

Der erste wichtige Abschnitt des Startvorgangs besteht aus der Initialisierung des Navigators und aller verfügbaren Plug-ins. Abbildung 5.10 zeigt einen Ausschnitt aus dem Baum der Methodenaufrufe für die Klassen `Launcher` und `Navigator`. Hier ist ersichtlich, dass die Initialisierung in mehreren Schritten erfolgt. Das Laden der Plug-ins durch die Klasse `Navigator` bildet den letzten Schritt. Tabelle 5.5 zeigt statistische Werte für die Ausführungszeiten der Methoden. Die gesamte Ausführungszeit in diesem Ausschnitt ist durch den Wert für die Methode `run()` gegeben. Sie liegt im Schnitt bei etwa sieben Sekunden. Der größte Anteil dieser Zeit, etwa 5,8 Sekunden, ist für die Ausführung der Methode `main()` in der Klasse `Navigator` nötig. Im weiteren Verlauf zeigt sich, dass `loadPlugins()` mit durchschnittlich etwa fünf Sekunden die aufwendigste Methode in diesem Bereich ist. An dieser Stelle erfolgt das Instanziieren aller Plug-ins. Da Lapap MK II aus mehr als 20 einzelnen Plug-ins besteht ist dieser Wert hier nicht frappierend. Ein auffallend hoher Wert von 1362,11 Millisekunden ist jedoch für das Instanziieren des `ODFProcPlugins` ersichtlich. Eine vollständige Aufstellung der Zeiten für das Instanziieren und Initialisieren aller Plug-ins ist in Abschnitt B.1 gegeben.

Eine weitere aufwendige Methode ist `performPluginInitialization()`. Alle Plug-ins, die das Interface *InitializationHandling* implementieren, bedürfen einer weiteren Initialisierung mittels der Methode `performInitialization()`. Der Aufruf dieser Methode seitens des Navigators erfolgt an dieser Stelle. Für diese weitere Initialisierung sind etwa 490 Millisekunden nötig.

Die 30 Durchführungen zeigen eine hohe Beständigkeit für diesen Abschnitt des Startvorgangs von Lapap MK II. Aus den Minima und Maxima folgt, dass insgesamt Abweichungen von bis zu 600 Millisekunden auftreten können. Durchschnitt und Median liegen aber in allen Fällen dicht beisammen, sodass kein übermäßiger Einfluss einzelner Werte festzustellen ist. Ein anormales Verhalten lässt sich an dieser Stelle ausschließen.

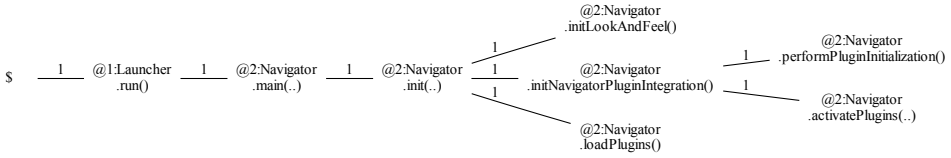

Abbildung 5.10. Aufrufbaum für die Methoden der Klassen `Launcher` und `Navigator`

Tabelle 5.6. Statistische Werte für die Ausführungszeiten der Methoden beim Initialisieren des `DMSAdapters` in Millisekunden während des Neustarts von Lapap MK II

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
SystemFacadeImpl	connect()	5339,97	7065,29	5993,29	6007,45
DMSAdapter	setup()	5334,55	7001,68	5957,85	5929,48
	initSWMessagingCtrl()	4,57	111,44	14,87	4,75
	initDMSServerProcess()	3054,74	4029,82	3211,25	3094,47
	initNotificationCtrl()	6,89	102,02	13,14	8,10
	initCDIConfiguration()	1420,47	2097,27	1773,67	1767,20
	getCDI(3)	61,96	402,98	124,49	92,87
	getCDI(2)	61,78	402,72	124,19	91,97
	initDataPoolCtrl()	413,06	1193,79	816,26	819,54
	initStatusTmSubscription()	3,67	393,43	83,02	59,12

Der zweite Abschnitt umfasst die Initialisierung des `DMSAdapters`. Der Zusammenhang der überwachten Methoden ist in Abbildung 5.11 dargestellt. Wie bereits erläutert ist der `DMSAdapter` für den Aufbau der Verbindung zum Data-Management-System sowie das Initialisieren des Datenpools zuständig. Der Aufruf der `setup()`-Methode erfolgt aus dem `SystemInterface` heraus. Tabelle 5.6 zeigt statistische Werte für die Ausführungszeiten der Methoden. Hier ist ersichtlich, dass die gesamte Ausführungszeit in diesem Abschnitt durchschnittlich etwa 6 Sekunden beträgt. Die Ausführungszeiten der Methoden `connect()` und `setup()` weichen nur unwesentlich voneinander ab. Mit durchschnittlich 3,2 Sekunden ist `initDMSServerProcess()` die aufwendigste Methode in diesem Abschnitt. Minimum und Maximum zufolge sind an dieser Stelle Abweichungen von etwa einer Sekunde möglich. Dies lässt sich vermutlich bereits durch die nötige Netzwerkkommunikation erklären, die Verzögerungen hervorrufen kann. Als weitere aufwendige Methode ist `initCDIConfiguration()` mit einem Durchschnittswert von 1,7 Sekunden erkennbar. Sie umfasst das Laden von insgesamt sechs Configuration-Data-Items (CDIs) durch je einen Aufruf der Methode `getCDI(3)`. Entsprechend basieren die statistischen Werte der Methoden `getCDI(3)` und `getCDI(2)` auf jeweils 180 Aufrufen. Da es sich hier lediglich um

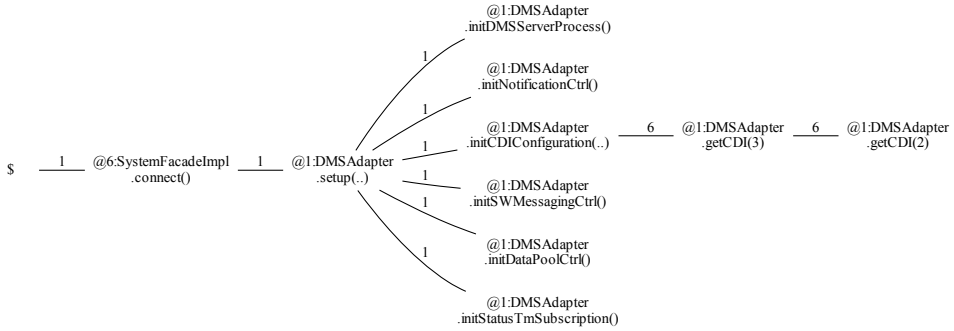

Abbildung 5.11. Aufrufbaum für die Methoden der Klasse `DMSAdapter`

einen Neustart von Lapap MK II handelt, entfällt das vollständige Nachladen der Dateien aus dem Netzwerk, sofern sie lokal vorhanden sind. Die für die Initialisierung der CDI-Konfiguration nötige Zeit ist überwiegend konstant. Kleinere Abweichungen lassen sich auch hier durch Netzwerkkommunikation erklären. Die Methoden `initSWMessagingCtrl()`, `initNotificationCtrl()` und `initStatudTmSubscription()` erfordern dem Median zufolge meist deutlich weniger Zeit als die zuvor genannten Methoden. Die Maxima zeigen, dass in Einzelfällen deutlich mehr Zeit nötig sein kann. Diese Differenzen haben aber nur geringen Einfluss auf die gesamte Ausführungszeit für diesen Abschnitt. Die Ausführungszeit der Methode `initDataPoolCtrl()` ist im Folgenden näher erläutert.

Das Initialisieren des Datenpools stellt den dritten überwachten Abschnitt des Startvorgangs dar. Sie erfolgt mittels der Methode `initDataPoolCtrl()` des `DMSAdapters`. Abbildung 5.12 zeigt die Zusammenhänge der Methoden in diesem Bereich. Hier ist ersichtlich, dass neben der Klasse `DataPoolCtrlImpl` drei weitere Klassen beteiligt sind. Statistische Werte für die einzelnen Methoden sind in Tabelle 5.7 dargestellt. Demnach benötigt die Methode `init()` durchschnittlich etwa 690 Millisekunden. Diese Zeit teilt sich auf drei weitere Methoden auf. Mit etwa 30 bis 40 Millisekunden ist `createDMSDataPool()` die einfachste dieser Methoden. Sie initialisiert im weiteren Verlauf ein `DmsDataPoolImpl`-Objekt. Obwohl die Methode `initializeDataPool()` dem Median zufolge mit 13,01 Millisekunden wenig Zeit benötigt, sind an dieser Stelle bereits Abweichungen von mehr als 100 Millisekunden möglich. Werte oberhalb von 40 Millisekunden sind aber nur vereinzelt aufgetreten. Mehr Zeit benötigt die Methode `initAcquisitionServer()`, deren Durchschnitt fast 140 Millisekunden beträgt. Mittels der Methode `open()` erfolgt hier der Aufbau der Verbindung zum Data-Management-System. Für die statistischen Werte der Methode `open()` ist hier zu beachten, dass diese auf zwei Aufrufen pro Durchgang basieren, wie es auch in Abbildung 5.12 ersichtlich ist. Die Betrachtung der Aufrufe im Einzelnen zeigt, dass hier zwei verschiedene Muster vorliegen. Der erste Aufruf, der durch die Methode

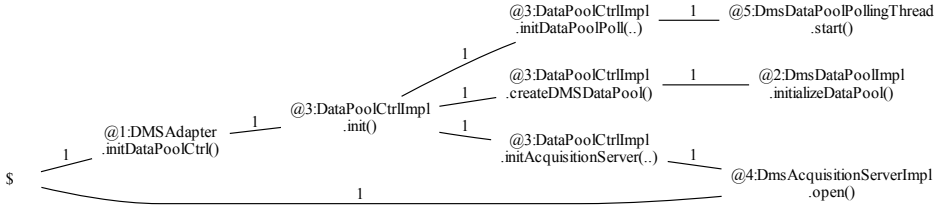

Abbildung 5.12. Aufrufbaum für die Methoden zum Initialisieren der Datenpools

Tabelle 5.7. Statistische Werte für die Ausführungszeiten der Methoden beim Initialisieren des Datenpools in Millisekunden während des Neustarts von Lapap MK II

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
DMSAdapter	initDataPoolCtrl()	413,06	1193,79	816,26	819,54
DataPoolCtrlImpl	init()	381,77	1073,09	692,69	682,84
	createDMSDataPool()	18,93	148,40	44,67	30,50
	initAcquisitionServer()	69,57	293,25	139,26	126,20
	initDataPoolPoll()	283,78	826,55	496,56	474,45
DmsDataPoolImpl	initializeDataPool()	9,26	121,99	23,17	13,01
DmsDataPoolPollingThread	start()	271,47	815,09	479,43	462,76
DmsAcquisitionServerImpl	open()	0,20	191,43	72,91	58,79

`initAcquisitionServer()` erfolgt, benötigt stets 50 bis 200 Millisekunden. Der zweite Aufruf hingegen nimmt mit 0 bis 70 Millisekunden deutlich weniger Zeit in Anspruch. Somit zeigen sich an dieser Stelle ebenfalls keine Auffälligkeiten. Die Methode `start()` ist mit durchschnittlich 479 Millisekunden die aufwendigste in diesem Abschnitt. Hier erfolgt der Start des Threads, der kontinuierlich die Daten des Data-Management-Systems abfragt.

Insgesamt zeigen diese Tests, dass an vielen Stellen des Startvorgangs von Lapap MK II Abweichungen in den Ausführungszeiten auftreten können. Bedingt durch die Tatsache, dass der Laptop neben Lapap MK II weitere Prozesse ausführt, sind Abweichungen aber nicht frappierend. Auch wenn sich die Ausführungszeiten im Einzelnen deutlich unterscheiden können, wie es etwa für die Methode `start()` aus Tabelle 5.7 der Fall ist, zeigt sich für die Betrachtung von größeren Abschnitten nur ein geringer Einfluss. Daher bestätigt sich der Eindruck, dass die für den gesamten Startvorgang nötige Zeit nahezu konstant ist.

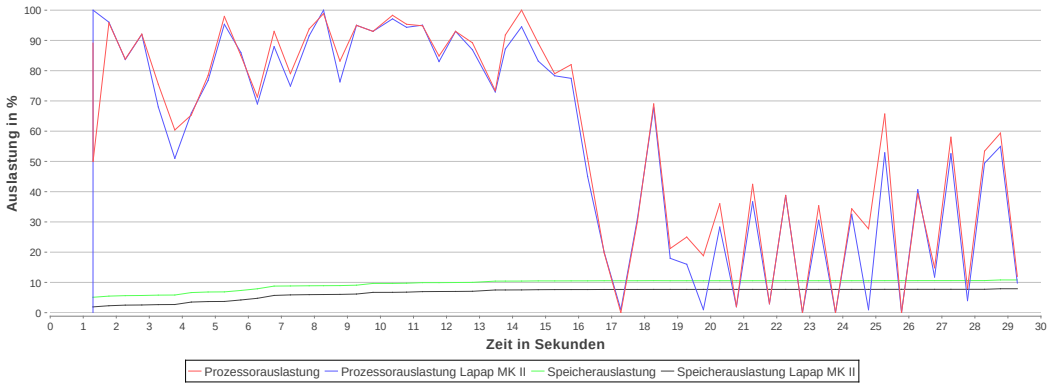

Abbildung 5.13. Auslastung des Prozessors und des Arbeitsspeichers im Ganzen sowie anteilig von Lapap MK II für Durchgang 29 der Prozedur

Auslastung der Ressourcen

Bezüglich der Auslastung der Ressourcen zeigt der Startvorgang von Lapap MK II in allen Durchgängen ein ähnliches Verhalten. Die Auslastung des Prozessors lässt sich stets in zwei Bereiche unterteilen. Die erste Hälfte zeichnet sich dadurch aus, dass die Werte durchgehend über 50%, zumeist sogar über 70% liegen. In der zweiten Hälfte bewegen sich die Werte regelmäßig unterhalb von 40%. Höhere Werte treten hier nur vereinzelt auf. Abbildung 5.13 zeigt dieses Muster anhand der Werte aus Durchgang 29. Hier ist auch ersichtlich, dass Lapap MK II den größten Teil der Prozessorauslastung verursacht. Nur an wenigen Punkten weicht die Kurve deutlich von der Gesamtauslastung ab. Der Verlauf der Speicherauslastung zeigt hier keine Auffälligkeiten. Wie es bei einem Programmstart zu erwarten ist, steigt sie zu Beginn an. Bis zum Ende des Startvorgangs erreicht sie einen Wert von etwa 10%. Die Auslastung durch Lapap MK II liegt durchgehend nur wenige Prozentpunkte darunter.

Abbildung 5.14 zeigt zusätzlich zur Auslastung des Prozessors die Bereiche der überwachten Methoden sowie die Anzahl der Datei-Deskriptoren von Lapap MK II. Hier ist ersichtlich, dass die Ausführungszeiten der Methoden `Launcher.main()` und `SystemFacadeImpl.connect()` zusammen fast die erste Hälfte des Startvorgangs ausmachen. Der Auslastung des Prozessors zufolge verursachen sie einen wesentlichen Teil der Arbeitslast in diesem Bereich. In der ersten Hälfte zeigt sich zudem ein deutlicher Anstieg der Anzahl der Datei-Deskriptoren. Ein solcher Anstieg geht hier zumeist mit einer hohen Prozessorauslastung einher. In Abbildung 5.14 zeigt sich zum Zeitpunkt von etwa acht Sekunden ein besonders starker Anstieg der Zahl der Datei-Deskriptoren, der hier mit einer Auslastung von bis zu 100% zusammenfällt. In anderen Durchgängen verlief dieser Anstieg weniger sprunghaft und zog sich oft in den Bereich der Methode `SystemFacadeImpl.connect()` hinein. Für die Anzahl der Threads von Lapap MK II lässt sich hier ebenfalls ein markanter Anstieg feststel-

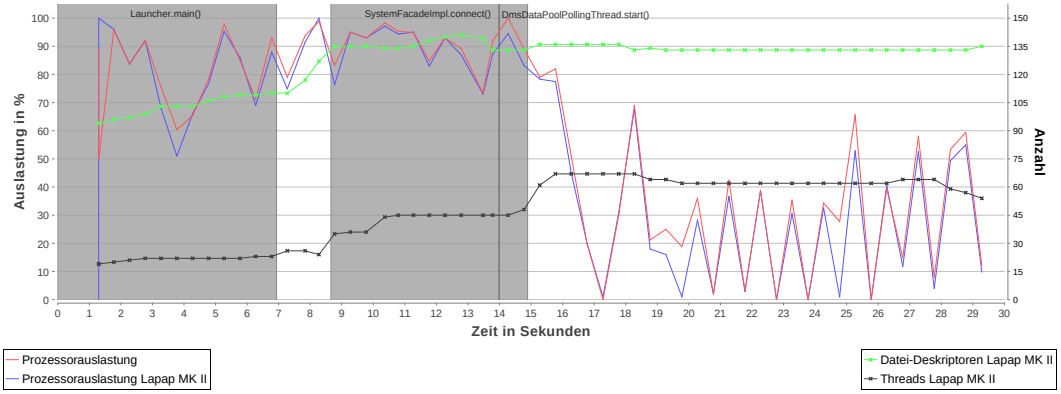

Abbildung 5.14. Auslastung des Prozessors, Anzahl der Datei-Deskriptoren und Threads von Lapap MK II sowie die Bereiche für die Ausführung der Methoden `Launcher.main()` und `SystemFacadeImpl.connect()` für Durchgang 29 der Prozedur

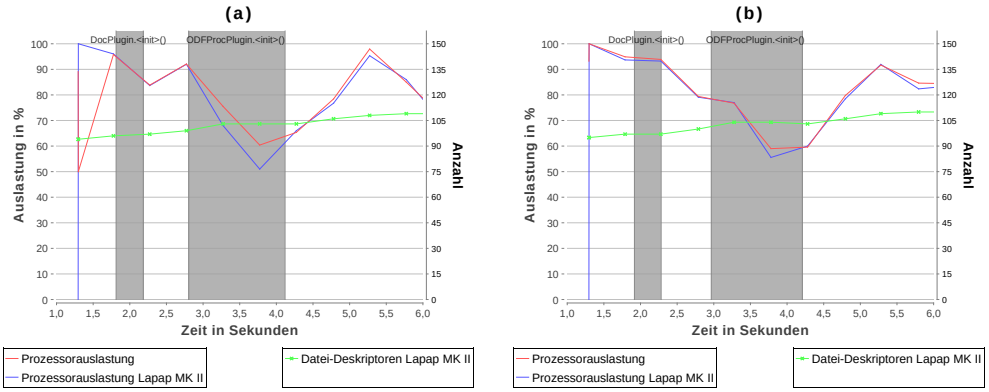

Abbildung 5.15. Auslastung des Prozessors, Anzahl der Datei-Deskriptoren sowie die Bereiche für die Ausführung der Konstruktoren der Klassen `DocPlugin` und `ODFProcPlugin` für (a) Durchgang 29 und (b) Durchgang 11 der Prozedur

len, der in allen Durchläufen nach dem Starten des `DmsDataPoolPollingThreads` ersichtlich ist.

Besonders aufwendig ist das Instanziieren der Plug-ins `DocPlugin` und `ODFProcPlugin`. Abbildung 5.15 zeigt die Bereiche für die Ausführungszeiten der Konstruktoren dieser Plug-ins in (a) Durchgang 29 und (b) Durchgang 11. Für Durchgang 29 ist ersichtlich, dass das markante Absinken der Prozessorauslastung mit dem Instanziieren des `ODFProcPlugins` zusammenfällt. Die übrigen Durchläufe bestätigen dies, wie Abbildung 5.15 (b) anhand von Durchgang 11 belegt. Für den Konstruktor des `DocPlugins` zeigt sich ebenfalls häufig eine sinkende oder zumindest gleich bleibende Auslastung des Prozessors. Daraus folgt, dass

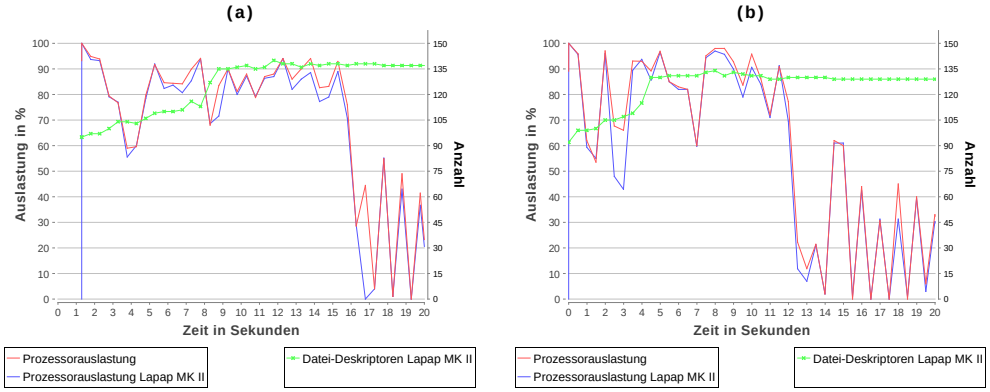

Abbildung 5.16. Auslastung des Prozessors sowie Anzahl der Datei-Deskriptoren für (a) Durchgang 11 der Prozedur für den Neustart von Lapap MK II, (b) Durchgang zwei der Prozedur für das Öffnen von PDF-Dokumenten

die im Vergleich zu anderen Plug-ins hohe Ausführungszeit nicht aus einer außerordentlich hohen Arbeitslast folgt. Besonders beim `ODFProcPlugin` lässt sich vermuten, dass ein Teil der Zeit auf Kommunikation im Netzwerk zurückzuführen ist. Für die übrigen Plug-ins lassen sich an dieser Stelle keine detaillierteren Analysen durchführen. Die Erhebung der Daten über die Ressourcenauslastung erfolgte mit einer Frequenz von zwei Hz. Die Ausführungszeiten der Konstruktoren liegen aber zumeist deutlich darunter, häufig in der Größenordnung von wenigen Millisekunden. Eine Aufstellung der Zeiten für die einzelnen Plug-ins ist in Abschnitt B.1 zu finden.

Zwischen dem Verlauf der Ressourcenauslastung während der Durchführung der Prozedur für den Startvorgang und der Prozedur für das Öffnen von PDF-Dokumenten existiert eine starke Ähnlichkeit. Dies ist nicht verwunderlich, da die Aufzeichnung der Werte über die Auslastung der Ressourcen in beiden Fällen mit dem Start von Lapap MK II beginnt. Ein detaillierterer Vergleich zeigt hier aber dennoch Unterschiede. Abbildung 5.16 zeigt in (a) den Verlauf der Ressourcenauslastung für die ersten 20 Sekunden aus Durchlauf 11 der Prozedur für den Neustart von Lapap MK II. Zum Vergleich sind in Abbildung 5.16 (b) die ersten 20 Sekunden für den zweiten Durchgang der Prozedur für das Öffnen von PDF-Dokumenten dargestellt. Hier ist ersichtlich, dass die Prozessorauslastung zu Beginn in beiden Fällen ähnlich hoch ist. Das markante Absinken der Auslastung erfolgt in Abbildung 5.16 (a) jedoch etwa zwei Sekunden später als in Abbildung 5.16 (b). Ein Vergleich mit anderen Durchführungen bestätigt die Größenordnung von 1,5 bis zwei Sekunden. Die Ursache für diese Verzögerung dürfte im Wesentlichen in der Erhebung der Daten mittels Kieker zu finden sein. Durch den Einsatz von Kieker erhöht sich die Ausführungszeit jeder überwachten Methode. Die Differenz beträgt nach van Hoorn u. a. [2012] zwar weniger als 10% der Ausführungszeit, jedoch reicht dies hier bereits aus, um einen Teil der Verzögerung zu verursachen.

Tabelle 5.8. Statistische Werte für die Ausführungszeiten der Methoden beim Initialisieren des Navigators in Millisekunden für den automatischen Start von Lapap MK II nach dem Start des Laptops

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
Launcher	main()	10071,05	15088,55	10989,16	10765,15
	run()	10069,69	15087,03	10984,89	10762,63
Navigator	main()	7635,72	12522,95	8362,88	8181,72
	init()	7629,19	12522,76	8362,38	8181,26
	loadPlugins()	6158,72	10806,53	6832,29	6689,15
	initNavigatorPluginIntegration()	706,56	1148,84	838,33	796,53
	activatePlugins()	15,21	74,74	33,67	32,94
	performPluginInitialization()	526,55	925,05	654,62	646,48
	initLookAndFeel()	433,97	1270,43	603,89	512,16

Bezüglich der Ressourcenauslastung zeigt dieser Test, dass der Startvorgang von Lapap MK II den Prozessor an einigen Stellen fast vollständig auslastet. Besonders während der Initialisierung der Plug-ins und des `DMSAdapters` zeigen sich vermehrt Spitzenwerte von über 90%. Da keine durchgehende Auslastung in dieser Höhe auftritt und es sich nur um Spitzenwerte handelt, liegt hier keine Überlastung des Prozessors vor. Der Bedarf an Arbeitsspeicher war in diesem Test durchgehend unauffällig.

5.2.2. Neustart des Laptops

Der erste Start von Lapap MK II erfolgt automatisch, sobald das Laden des Betriebssystems abgeschlossen ist. Dieser Startvorgang unterscheidet sich dahingehend von allen weiteren, dass er mit durchschnittlich 120 Sekunden mehr als dreimal so viel Zeit benötigt. Auch hier gilt der Vorgang als abgeschlossen, sobald das DMS-Icon im Status-Display durch seine grüne Hintergrundfarbe den erfolgreichen Aufbau der Verbindung zum DMS signalisiert und die Synoptic-Displays mit den aktuellen Werten aus dem Datenpool gefüllt sind. Die Auswertung der Daten für den Neustart des Laptops erfolgt ebenfalls in zwei Abschnitten.

Ausführungszeiten der Methoden

Der erste Abschnitt des Startvorgangs umfasst das Initialisieren des Navigators und der Plug-ins. Der Zusammenhang der Methoden entspricht auch hier Abbildung 5.10. Statistische Werte für die Ausführungszeiten der Methoden sind in Tabelle 5.8 dargestellt. Anhand der Werte der Methode `run()` zeigt sich bereits, dass während des ersten Startvorgangs für diesen Abschnitt mehr Zeit notwendig ist als bei allen weiteren. Mit durchschnittlich fast 11 Sekunden entspricht dies mehr als dem Eineinhalbfachen, mit etwa 15 Sekunden im schlechtesten Fall sogar der doppelten Zeit. Den größten Anteil davon macht wie auch bei

Tabelle 5.9. Statistische Werte für die Ausführungszeiten der Methoden beim Initialisieren des `DMSAdapters` in Millisekunden für den automatischen Start von Lapap MK II nach dem Start des Laptops

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
SystemFacadeImpl	connect()	112994,45	114528,82	113711,50	113715,58
DMSAdapter	setup()	112929,73	114515,25	113672,41	113685,61
	initSWMessagingCtrl()	5,02	15,67	7,75	5,21
	initDMSServerProcess()	3073,86	3970,44	3236,70	3141,95
	initNotificationCtrl()	7,66	101,59	16,86	10,40
	initCDIConfiguration()	108972,18	110330,73	109803,07	109805,79
	getCDI(3)	60,43	95449,87	17332,30	282,34
	getCDI(2)	60,05	95449,19	17331,48	281,44
	initDataPoolCtrl()	426,44	872,31	506,83	469,81
	initStatusTmSubscription()	15,94	406,76	59,13	44,12

einem Neustart von Lapap MK II die Methode `init()` aus. Hier sind durchschnittlich 8362,38 Millisekunden nötig und damit etwa 2,5 Sekunden mehr als bei einem Neustart. Das Laden der Plug-ins und damit die Methode `loadPlugins()` bleibt der aufwendigste Teil in diesem Abschnitt. Gegenüber einem Neustart von Lapap MK II sind bereits an dieser Stelle etwa 1,9 Sekunden mehr nötig. Die Auswirkungen auf die einzelnen Plug-ins sind jedoch unterschiedlich. Besonders auffällig ist hierbei das `ResetFunctionPlugin`. Der Durchschnittswert beträgt zwar nur 18,6 Millisekunden, jedoch entspricht dies mehr als dem Sechsfachen des Wertes beim Neustart von Lapap MK II. Eine Reihe anderer Plug-ins wie beispielsweise das `USSPlugin` oder das `MessageProviderPlugin` benötigen mehr als die doppelte Zeit. Bei anderen Plug-ins zeigt sich kein nennenswerter Unterschied. Eine vollständige Aufstellung der Zeiten für alle Plug-ins ist in Abschnitt B.2 zu finden.

Die Methode `initNavigatorPluginIntegration()` benötigt mit 838,33 Millisekunden durchschnittlicher Ausführungszeit mehr als das 1,4-fache im Vergleich zum Neustart. Diese Verzögerung ist hier vollständig auf die zusätzliche Initialisierung der Plug-ins mittels der Methode `performPluginInitialization()` zurückzuführen. Die Methode `activatePlugins()` zeigt als einzige in diesem Abschnitt keine signifikante Änderung der Ausführungszeit. Der Durchschnitt liegt hier weiterhin bei etwa 33 Millisekunden. Ein größerer Unterschied ist bei der Methode `initLookAndFeel()` ersichtlich. Mit durchschnittlich 603,89 Millisekunden ist hier im Vergleich zum Neustart von Lapap MK II mehr als die doppelte Zeit notwendig. Der Maximalwert von 1270,43 Millisekunden zeigt zudem, dass die Abweichung hier noch höher ausfallen kann.

Im zweiten Abschnitt des Startvorgangs, der das Initialisieren des `DMSAdapters` beschreibt, zeigen sich deutlichere Änderungen. Bei den statistischen Werten in Tabelle 5.9 fällt besonders der stark erhöhte Wert für die Methode `setup()` auf, für deren Ausführung nun mehr als 113 Sekunden nötig sind. Dies entspricht einem Anstieg auf fast das

Tabelle 5.10. Statistische Werte für die Ausführungszeiten der sechs Aufrufe der Methode `getCDI(2)` in Millisekunden für den automatischen Start von Lapap MK II nach dem Start des Laptops

Aufruf	CDI	Dateigröße	Min	Max	$\bar{x}$	$\tilde{x}$
1	OPS_CDI	1228,80 KiB	94989,11	95449,19	95155,29	95133,94
2	PRIV_DATAPOOL_CDI	76,00 KiB	98,89	273,74	128,46	122,74
3	PUB_DATAPOOL_CDI	263,00 KiB	60,05	163,54	93,91	80,22
4	EVENT_CDI	97,39 KiB	7693,57	8453,63	8074,91	8090,94
5	EQUIPMENT_CDI	3,10 KiB	289,14	478,49	340,44	324,01
6	MONITORING_TAB_CDI	0,15 KiB	121,85	262,24	195,85	198,20

19-fache gegenüber dem Neustart von Lapap MK II. Der Zusammenhang der Methoden in Abbildung 5.11 macht deutlich, dass die Ursache der Verzögerung an dieser Stelle in der Methode `getCDI(2)` zu finden ist. Der Maximalwert belegt, dass für einen einzigen Aufruf dieser Methode bereits mehr als 95 Sekunden nötig sein können. Da der Median hier nur etwa 280 Millisekunden beträgt, deutet dies zunächst auf ein anormales Verhalten hin. Eine detailliertere Auswertung zeigt aber, dass in jedem einzelnen Durchgang ein Maximalwert von etwa 95 Sekunden auftritt. Die statistischen Werte für die Methode `getCDI(2)` aus Tabelle 5.9 beruhen auf sechs Aufrufen pro Durchlauf. In Tabelle 5.10 ist die Auswertung der sechs einzelnen Aufrufe pro Durchgang dargestellt. Hier ist ersichtlich, dass der erste Aufruf der Methode stets etwa 95 Sekunden benötigt. Bei einem Neustart von Lapap MK II lag dieser Wert bei nur etwa 263 Millisekunden.

Dieser Unterschied ist durch das Verhalten von Lapap MK II begründet. Die CDIs sind zunächst nicht lokal auf dem Laptop gespeichert, sie befinden sich auf der MMU. Das Nachladen über das Netzwerk erfolgt im Zuge des ersten Startvorgangs von Lapap MK II. Bei allen folgenden Startvorgängen findet eine Verifikation der CDIs statt und das Nachladen entfällt bei Erfolg. Bezüglich der Dateigröße von 1228,80 KiB erscheint eine Ladezeit von 95 Sekunden dennoch als erhöht. Die durchschnittliche Ladezeit der übrigen CDIs erweckt ebenfalls diesen Eindruck. Der Grund dafür liegt jedoch in der *slow_copy*-Funktion. Die Entwicklung der slow_copy-Funktion war notwendig, um Probleme zu beheben, die bei der Verwendung von unterschiedlichen Network-File-Systems (NFSs) auf dem Laptop und der MMU aufgetreten sind. Die slow_copy-Funktion verlangsamt die Lese- und Schreibzugriffe und verhindert dadurch eine Blockade der MMU durch Programme des Laptops [SPR-23838; Columbus System Laptop ADD].

Für die übrigen Methoden in diesem Abschnitt zeigen sich schwächere Auswirkungen. Die Ausführungszeiten der Methoden `initDMSServerProcess()` und `initNotificationCtrl()` sind in etwa gleich. Mit durchschnittlich 59,13 Millisekunden sind zum Ausführen der Methode `initStatusTmSubscription()` sogar etwa 28% weniger Zeit notwendig als bei einem Neustart von Lapap MK II.

Ebenfalls weniger Zeit benötigt die Methode `initDataPoolCtrl()`, die mit 506,83 Milli-

Tabelle 5.11. Statistische Werte für die Ausführungszeiten der Methoden beim Initialisieren des Datenpools in Millisekunden für den automatischen Start von Lapap MK II nach dem Start des Laptops

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
DMSAdapter	initDataPoolCtrl()	426,44	872,31	506,83	469,81
DataPoolCtrlImpl	init()	390,59	749,63	450,41	427,58
	createDMSDataPool()	22,11	62,49	26,54	23,57
	initAcquisitionServer()	58,19	78,54	66,51	67,15
	initDataPoolPoll()	257,16	322,00	286,91	287,96
DmsDataPoolImpl	initializeDataPool()	11,53	33,56	14,20	12,50
DmsDataPoolPollingThread	start()	247,27	312,41	275,00	275,94
DmsAcquisitionServerImpl	open()	2,08	63,48	29,19	30,89

sekunden bei nur noch 62% der Zeit während des Neustarts liegt. Besonders frappierend ist an dieser Stelle, dass sich diese positiven Auswirkungen bei allen Methoden im dritten Abschnitt widerspiegeln, wie die Werte in Tabelle 5.11 belegen. Die Ausführungszeiten der meisten Methoden sind auf 57% bis 65% der bei einem Neustart von Lapap MK II gemessenen Werte zurückgegangen. Besonders deutlich zeigt dies der durchschnittliche Wert der Methode `initAcquisitionServer()`, der mit 66,51 Millisekunden um mehr als 50% gesunken ist. Dieses Verhalten war hier nicht zu erwarten, da das Erzeugen und Befüllen des Datenpools bei jedem Start von Lapap MK II gleichermaßen durchzuführen ist.

Entsprechend zeigt die Auswertung hier insgesamt ein zweiteiliges Ergebnis. Bezüglich Ziel Z2 hat sich bestätigt, dass das Nachladen der CDIs unter Verwendung der slow_copy-Funktion eine Ursache für die Verzögerung des Startvorgangs ist. Mit etwa 95 Sekunden Ladezeit macht bereits eine der sechs CDIs mehr als 75% des Startvorgangs aus. Als weitere Ursache hat sich aber auch das Initialisieren der Plug-ins herausgestellt.

Auslastung der Ressourcen

Für den Verlauf der Ressourcenauslastung zeigt sich für den ersten Startvorgang ein anderes Bild als bei allen folgenden. Die Nutzung des Arbeitsspeichers ist unauffällig wie zuvor. Sie steigt innerhalb von 20 Sekunden auf etwa 10% an und ist im weiteren Verlauf konstant. Die Auslastung des Prozessors unterscheidet sich jedoch. Abbildung 5.17 zeigt die Ressourcenauslastung aus Durchgang 13. Hier lassen sich drei Bereiche erkennen. In den ersten 20 Sekunden zeigt sich eine hohe Prozessorauslastung mit Spitzenwerten von mehr als 90%, die auch bei den weiteren Startvorgängen erkennbar ist. Hier fällt jedoch auf, dass dieser Abschnitt mit etwa 20 Sekunden größer ist als bei einem Neustart. Zudem ist ersichtlich, dass die durch Lapap MK II verursachte Prozessorauslastung stärker von der Gesamtauslastung abweicht als bei einem Neustart. Dies ist möglicherweise auf Nachwirkungen des Systemstarts zurückzuführen.

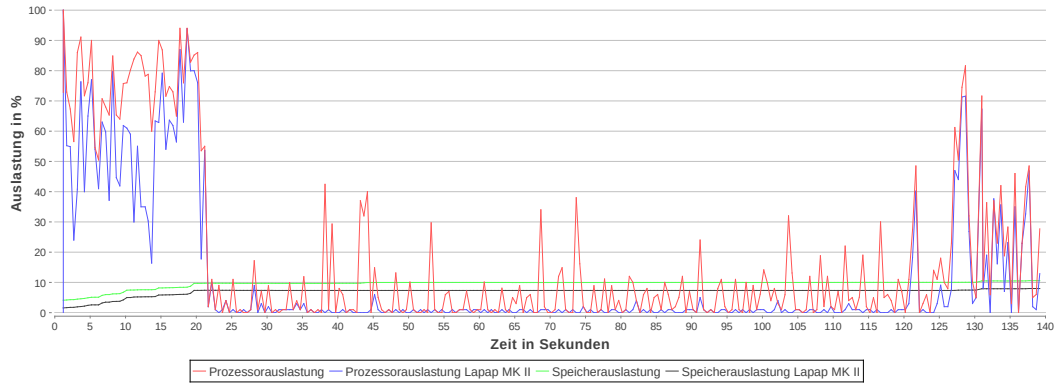

Abbildung 5.17. Auslastung des Prozessors und des Arbeitsspeichers im Ganzen sowie anteilig durch Lapap MK II für Durchgang 13 der Prozedur für den Neustart des Laptops

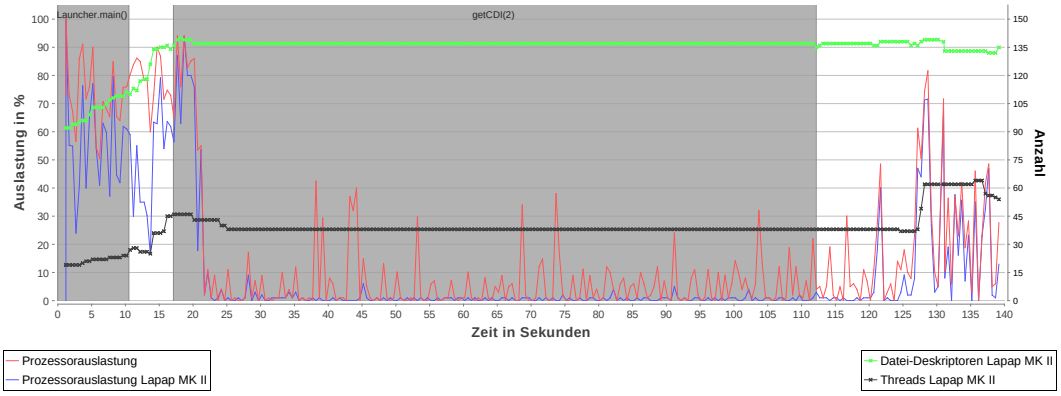

Abbildung 5.18. Auslastung des Prozessors, Anzahl der Datei-Deskriptoren und Threads von Lapap MK II sowie Markierungen von Methodenaufrufen für Durchgang 13 der Prozedur für den Neustart des Laptops

Im zweiten Abschnitt ist ein deutlicher Rückgang der Prozessorauslastung erkennbar. Über einen Zeitraum von fast 100 Sekunden zeigen sich Werte von etwa 10%. Nur vereinzelt treten Werte über 30% auf. Damit zeigt sich der Rückgang der Auslastung deutlich stärker, als es bei einem Neustart von Lapap MK II der Fall ist. Dies deutet bereits darauf hin, dass in diesem Bereich das Laden der CDIs stattfindet.

Der letzte Abschnitt lässt wieder eine höhere Prozessorauslastung erkennen. Die Werte liegen hier überwiegend zwischen 30% und 40%, Spitzenwerte aber auch darüber. Dieses Verhalten ist mit dem zweiten Abschnitt der Prozessorauslastung bei einem Neustart von Lapap MK II vergleichbar, wie es auch in Abbildung 5.13 dargestellt ist.

Abbildung 5.18 zeigt zusätzlich zur Auslastung des Prozessors die Ausführungszeiten der Methode `main()` sowie des ersten Aufrufs von `getCDI(2)`. Wie bereits anhand der Ausführungszeiten ersichtlich war, erstreckt sich die Methode `main()` hier über einen Zeitraum von mehr als 10 Sekunden mit hoher Prozessorauslastung. Hier bestätigt sich außerdem die Vermutung, dass in dem mittleren Abschnitt das Laden der CDIs stattfindet. Zu Beginn des Ladevorgangs zeigt sich noch eine hohe Prozessorauslastung von mehr als 70%. In Verbindung mit der Anzahl der Threads von Lapap MK II deutet dies auf eine starke nebenläufige Nutzung des Prozessors hin. In dem folgenden Abschnitt ist offenbar keine weitere Nutzung von Nebenläufigkeit möglich. Die Auslastung des Prozessors geht deutlich zurück. Dieses Bild ändert sich erst mit dem Starten des `DmsDataPoolPollingThreads`, was zu einer Prozessorauslastung von mehr als 50% führt. Im gleichen Zug steigt die Anzahl der Threads um mehr als 20 an. Anschließend zeigt sich wieder eine regelmäßige Auslastung zwischen 30% und 40%, die auf das Synchronisieren der Daten mittels des `DmsDataPoolPollingThreads` zurückzuführen ist. Bei einem Neustart von Lapap MK II stellt sich dies anders dar. Durch die deutlich kürzere Ladezeit der CDIs fällt das Starten des `DmsDataPollPollingThreads` mit der Ausführung anderer Threads zusammen. Wie Abbildung 5.14 zeigt, ist hier kein starker Rückgang der Prozessorauslastung ersichtlich. Das Absinken und wieder Ansteigen der Anzahl der Threads, wie es in Abbildung 5.18 während der Ladezeit dargestellt ist, fällt bei einem Neustart von Lapap MK II zusammen.

Diese Analyse hat ergeben, dass die Verzögerungen während des ersten Startvorgangs von Lapap MK II einzig auf die erhöhten Ausführungszeiten zurückzuführen sind. Zwar zeigt sich zu Beginn und gegen Ende des Startvorgangs eine hohe Auslastung des Prozessors, doch entspricht dies zumeist dem Verhalten, wie es auch bei allen weiteren Startvorgängen aufgetreten ist. Der mittlere Abschnitt von etwa 100 Sekunden Länge, in dem das Laden der CDIs stattfindet, verzeichnet hingegen nur selten eine Auslastung von mehr als 15%. Entsprechend liegt an dieser Stelle keine Performance-Anomalie vor.

5.3. Weitere Ergebnisse

Wie bereits in Abschnitt 4.3 beschrieben, trat bei der Durchführung der Testprozeduren mehrfach ein fehlerhaftes Verhalten von Lapap MK II auf. Demnach war es nach mehrmaligem Neustart von Lapap MK II nicht mehr möglich, eine Verbindung zum Data-Management-System aufzubauen. Es hat sich herausgestellt, dass dieses Verhalten gezielt rekonstruierbar ist. Ab dem neunten Startvorgang von Lapap MK II ist keine erneute Anmeldung beim DMS-Prozess des Laptops möglich, ohne zuvor einen Neustart des Laptops durchzuführen. Die angezeigte Fehlermeldung besagte, dass Lapap MK II wiederholt versuchen würde, die Verbindung herzustellen, bis dies gelungen sei. Nach Ansicht einiger Entwickler ist dies in der aktuellen Version jedoch nicht mehr der Fall. Dieses Verhalten zeigte sich neben dem Laptop, auf dem die Durchführung der Testprozeduren erfolgte, auch auf einem weiteren Laptop, auf dem die Standardversion von Lapap MK II ohne Kieker-Integration installiert war. Es trat sowohl an der Testanlage 1 als auch an der

Testanlage 2 auf. Entsprechend folgte daraus ein neuer SPR [SPR-25408].

Eine weiterführende Analyse, die parallel zu dieser Arbeit stattfand, hat ergeben, dass dieses Verhalten auf eine begrenzte Anzahl möglicher Anmeldungen bei dem DMS-Prozess auf dem Laptop zurückzuführen ist. In der Cycle 13-Version erfolgte der Start des DMS-Prozesses im Zuge des Starts von Lapap MK II. Ebenso führte ein Beenden von Lapap MK II zum Beenden des DMS-Prozesses. Entsprechend war zu keiner Zeit mehr als eine Anmeldung von Lapap MK II bei dem DMS-Prozess notwendig. In der aktuellen Cycle 14-Variante erfolgt der Start des DMS-Prozesses im Zuge des Bootvorgangs. Jeder Neustart von Lapap MK II erfordert eine erneute Anmeldung beim DMS-Prozess. Die Anzahl der Anmeldungen ist der Analyse zufolge jedoch auf acht begrenzt [SPR-25408].

Kapitel 6

Verwandte Arbeiten

Ein wichtiger Schritt zum Erreichen der in Abschnitt 1.2 beschriebenen Ziele besteht darin, den Aufbau, die Architektur und die Funktionsweise von Lapap MK II und einzelnen Komponenten zu verstehen. Neben statischen Informationsquellen wie der Dokumentation und dem Quellcode erfolgte in dieser Arbeit auch die Auswertung von dynamisch erhobenen Daten über das Laufzeitverhalten. Somit lässt sich diese Arbeit in den Themenbereich Reverse-Engineering einordnen.

In diesem Kapitel sind verwandte Arbeiten aufgeführt, die sich ebenfalls in den Bereich Reverse-Engineering einordnen lassen. Sie verwenden ebenfalls statische oder dynamische Analyse und lassen sich in zwei Kategorien einteilen. In Abschnitt 6.1 sind Arbeiten beschrieben, deren Ziel es ist, Informationen über den Aufbau oder die Funktionsweise einer Anwendung zu liefern. Andere Ansätze, die eine Analyse der Performance oder das Auffinden von Fehlern umfassen, sind in Abschnitt 6.2 aufgeführt.

6.1. Aufbau und Funktionsweise von Software

Im Bereich Reverse-Engineering existieren zahlreiche Ansätze, die sich ebenfalls mit dem Verständnis des Aufbaus und der Funktionsweise von Software befassen.

Ritsch und Sneed [1993] haben in ihrer Arbeit einen Ansatz vorgestellt, um die Funktionsweise einer Anwendung zu rekonstruieren. Sie verwenden zunächst statische Analyse, um Informationen über den Aufbau aus dem Quellcode zu extrahieren. Dies umfasst unter anderem die Struktur des Programms, verwendete Datentypen sowie Kontrollfluss-Strukturen. Mittels dynamischer Analyse erheben sie weitere Informationen, insbesondere über die Zusammenhänge zur Laufzeit sowie die Performance. Durch das Zusammenführen dieser Informationen entsteht ein umfassenderes Bild von der Funktionsweise der Anwendung.

Ritsch und Sneed [1993] verwenden die dynamische Analyse in einer ähnlichen Weise, wie es auch in der vorliegenden Arbeit der Fall ist. Sie nutzen die Daten um beispielsweise Ausführungszeiten einzelner Pfade zu berechnen. Dabei stützen sie die Analyse im Wesentlichen auf I/O-Operationen und Verzweigungen. Für die Untersuchung von Lapap MK II sind jedoch auch andere Kriterien relevant, wie etwa die Reaktionszeiten von Datenstrukturen, Verzögerungen durch Netzwerkkommunikation und die Verwendung der Systemressourcen.

Zaidman u. a. [2006] verfolgen in ihrer Arbeit unter anderem das Ziel Informationen über die Funktionsweise eines Altsystems zu gewinnen. Sie verwenden die dynamische Analyse zum einen, um Aufrufgraphen erzeugen und daraus Abhängigkeiten zwischen Methoden oder Prozeduren rekonstruieren zu können, zum anderen, um Metriken über die Verwendung von Methoden oder Prozeduren berechnen zu können.

Der Ansatz von Zaidman u. a. [2006] kann hilfreich sein, um Zusammenhänge innerhalb eines Programms zu verstehen und die Qualität des Quellcodes anhand von Metriken zu bewerten. Für die Analyse von Lapap MK II ist dieser Ansatz jedoch nicht verwendbar, da Informationen über die Zusammenhänge der Methoden allein nicht ausreichen, um beispielsweise die Ursachen von Verzögerungen identifizieren zu können.

Ein ähnliches Ziel verfolgen auch Knoche u. a. [2012] in ihrer Arbeit. Um dynamische Abhängigkeiten von COBOL-Systemen untersuchen zu können, führen sie zunächst eine statische Analyse durch. Anhand der gewonnen Informationen erfolgt die Auswahl der Monitoringpunkte für die anschließende dynamische Analyse. Zwar können sie auf diese Weise Abhängigkeitsgraphen konstruieren, jedoch sind auch hier relevante Kriterien wie Ausführungszeiten nicht berücksichtigt.

Weitere Ansätze nutzen sowohl die statische als auch die dynamische Analyse, um den Aufbau einer Anwendung anhand von extrahierten Informationen bezüglich der Architektur zu beschreiben. Riva und Rodriguez [2002] nutzen zunächst die statische Analyse, um für die Architektur relevante Elemente im Quellcode zu identifizieren. Mittels dynamischer Analyse erfolgt anschließend die Extraktion von Informationen über das Verhalten dieser Elemente. Durch weitere Abstraktionen erzeugen sie daraus Visualisierungen der Architektur.

Im Rahmen des DynaMod-Projekts haben van Hoorn u. a. [2011a, 2013] modellgetriebene Techniken zur Modernisierung von Altsystemen untersucht. Sie haben ebenfalls eine Kombination aus statischer und dynamischer Analyse verwendet, um Informationen über die Architektur und Nutzungsprofile eines Altsystems extrahieren zu können.

Die Architektur kann auch Aufschluss über den Aufbau einer Anwendung geben. Entsprechend erfolgte die Analyse in der vorliegenden Arbeit auch unter Zuhilfenahme von Informationen über die Architektur von Lapap MK II. Die Rekonstruktion der Architektur ist jedoch nicht zielführend, da neben dem Aufbau vor allem Informationen bezüglich des Verhaltens relevant sind.

6.2. Performance-Analyse und Fehler-Lokalisation

Andere Arbeiten nutzen die Techniken aus dem Bereich Reverse-Engineering, um die Performance einer Anwendung zu analysieren und Fehlerquellen zu lokalisieren.

Cheung und Madden [2008] haben in ihrer Arbeit ein Konzept vorgestellt, um die Performance eines laufenden Programms zu überwachen. Sie nutzen die dynamische Analyse in Verbindung mit einer gezielten Instrumentierung, um das Verhalten einer laufenden Anwendung zu untersuchen. Dadurch können sie beispielsweise Erkenntnisse

über die Ausführungszeiten von Operationen gewinnen. Für die Analyse von Lapap MK II ist das Konzept weniger geeignet, da es auf Profiling, also die permanente Überwachung einer laufenden Anwendung ausgelegt ist. Besonders die Analyse des Startvorgangs ist jedoch mit kontinuierlichen Neustarts verbunden und erfordern keine Überwachung über einen längeren Zeitraum.

Wechselberg [2013] untersucht in seiner Arbeit Performance-Probleme von Perl-basierten Webanwendungen. Dies geschieht ebenfalls mittels dynamischer Analyse unter Verwendung des Kieker-Frameworks. Die Grundlagen seiner Untersuchungen bilden die Ausführungszeiten der Methoden sowie die Anzahl der Methodenaufrufe. Ein Teil der Analyse von Lapap MK II basiert ebenfalls auf der Auswertung von Ausführungszeiten. Um das Verhalten von Lapap MK II bezüglich der Auslastung der Systemressourcen und möglicher Performance-Anomalien bewerten zu können, ist jedoch das regelmäßige Erheben zusätzlicher Daten unerlässlich.

Bensalem u. a. [2006] verwenden in ihrer Arbeit die dynamische Analyse, um lock- und unlock-Ereignisse in einem nebenläufigen Programm aufzuzeichnen. Aus diesen Daten rekonstruieren sie einen Graphen, der potentielle Deadlocks aufzeigt. Einen weiteren Ansatz zum Auffinden von Deadlocks haben Joshi u. a. [2010] vorgestellt. Sie verwenden die dynamische Analyse, um alle relevanten Methoden in einem nebenläufigen Programm zu identifizieren. Mittels Model-Checking erfolgt anschließend die Analyse der nebenläufigen Logik. Lapap MK II verwendet an vielen Stellen ebenfalls Nebenläufigkeit. Für das Verhalten beim Öffnen von PDF-Dokumenten lässt sich ein Deadlock als Ursache nicht vollständig ausschließen. Das Beschränken der Untersuchung allein auf Deadlocks als Ursache ist jedoch nicht zielführend, da andere Aspekte wie Ausführungszeiten oder Verzögerungen durch Netzwerkkommunikation nicht abgedeckt sind.

Kapitel 7

Fazit und Ausblick

In diesem Kapitel ist ein abschließendes Resümee der Ergebnisse dieser Arbeit gegeben. In Abschnitt 7.1 sind die Ergebnisse der einzelnen Tests zusammengefasst und bezüglich der Ziele Z1 bis Z3 bewertet. Ein Überblick über offene und weiterführende Aufgaben ist in Abschnitt 7.2 dargestellt.

7.1. Fazit

In dieser Arbeit erfolgte eine Analyse von Lapap MK II. Das Ziel bestand darin, mittels dynamischer Analyse Informationen über das Verhalten zur Laufzeit zu gewinnen und anhand dieser Daten die Ursachen für bisher nicht abschließend geklärte Verhaltensweisen von Lapap MK II zu identifizieren. Die Auswertung der Daten umfasste eine Analyse der Ausführungszeiten von einzelnen Methoden sowie die Auslastung der Ressourcen.

Öffnen von PDF-Dokumenten

Bezüglich Ziel Z1 hat sich für das Verhalten von Lapap MK II während des Öffnens von PDF-Dokumenten unter Verwendung der Standardkonfiguration ein eindeutiges Ergebnis herausgestellt. Die Analyse der Ausführungs- und Reaktionszeiten hat ergeben, dass die Ursache für die Verzögerung beim Öffnen eines PDF-Dokuments nicht durch die innerhalb von Lapap MK II implementierten Methoden gegeben ist. Die Ausführungszeiten dieser Methoden liegen hier durchgehend im Millisekundenbereich. Die Reaktionszeiten stellen mit durchschnittlichen Werten unter 350 Millisekunden ebenfalls keinen Grund für Verzögerungen von mehr als 30 Sekunden dar. Eine maximale Reaktionszeit von mehr als 1,8 Sekunden hat hingegen zu keinem wahrnehmbaren Einfluss geführt. Die Betrachtung der Prozessor- und Speichernutzung hat für den Ladevorgang kein außergewöhnliches Verhalten gezeigt. Zwar treten durchaus Werte von mehr als 90% für die Auslastung des Prozessors auf, jedoch besteht hier kein Zusammenhang mit dem Öffnen von PDF-Dokumenten. Allenfalls ein kurzzeitiger Anstieg der Auslastung im Anschluss an die überwachten Methoden lässt sich möglicherweise durch das Anzeigen der Dokumente mittels der `acroread`-Komponente begründen. Die Nutzung des Arbeitsspeichers war durchgehend unauffällig.

Ein weiterer Test mit reduziertem Heap-Speicher hat ähnliche Erkenntnisse geliefert. Das Arbeiten mit Lapap MK II war unter diesen Bedingungen langwieriger, da für jeden

einzelnen Schritt mehr Zeit notwendig war als zuvor. Zu schnelles Arbeiten hat häufig zum Auftreten eines *OutOfMemoryErrors* geführt. Somit ist es an dieser Stelle empfehlenswert, die minimale Größe des Heap-Speichers auf mindestens 128 Megabyte zu erhöhen. Entsprechend zeigen sich zwar für die Ausführungszeiten der überwachten Methoden höhere Werte, jedoch liegen diese weiterhin in der Größenordnung von Millisekunden. Auch die durchschnittlichen Werte der Reaktionszeiten waren nur unwesentlich größer. Ein signifikanter Anstieg der Reaktionszeit ließ sich nur in Einzelfällen feststellen. Die Auslastung des Prozessors fiel in diesem Szenario höher aus als zuvor. Auf Grund von vermehrtem Swapping war dieses Verhalten jedoch zu erwarten. Die Speicherauslastung blieb auch hier ohne Auffälligkeiten.

Diese Erkenntnisse belegen, dass die Ausführung der innerhalb von Lapap MK II implementierten Methoden nur einen geringen Anteil der Zeit vom Anstoßen eines Ladevorgangs bis zum Anzeigen des Dokuments ausmacht. Dementsprechend ist die Ursache für die Verzögerung beim Öffnen von PDF-Dokumenten nicht durch die Ausführung dieser Methoden gegeben. Vielmehr hat sich gezeigt, dass eine durch den Anwender wahrgenommene Verzögerung nicht mit überdurchschnittlichen Ausführungszeiten zusammenhängen muss. Beides kann unabhängig von dem jeweils anderen auftreten. Da die Ressourcenauslastung während der Ladevorgänge sowohl mit Standardkonfiguration als auch mit reduziertem Heap-Speicher unauffällig war, liegt an dieser Stelle keine Performance-Anomalie vor. Entsprechend sind die Verzögerungen auch nicht auf Speicherprobleme zurückzuführen. Die Ergebnisse deuten außerdem darauf hin, dass die Ursachen für die Verzögerung im `WebRenderer` oder der `acroread`-Komponente zu finden sind. Daraus folgt, dass durch die Architektur von Lapap MK II keine Begünstigung des untersuchten Verhaltens festzustellen ist.

Startvorgang von Lapap MK II

Die Analyse des Verhaltens von Lapap MK II während des ersten Startvorgangs sowie bei einem Neustart hat bezüglich Ziel Z2 mehrere Ursachen für die Verzögerung aufgezeigt. Die Auswertung der Ausführungszeiten einzelner Methoden belegt, dass in zwei der überwachten Abschnitte während des ersten Startvorgangs mehr Zeit notwendig ist als bei allen weiteren. Die größte Abweichung ist für das Initialisieren des `DMSAdapters` ersichtlich. Hier zeigt sich, dass das Nachladen der Configuration-Data-Items von der MMU während des ersten Startvorgangs den größten Teil der Zeit ausmacht. Bereits für das erste CDI sind circa 95 Sekunden nötig. Dies entspricht etwa drei Viertel der Zeit des ersten Startvorgangs. Als Ursache für die langen Ladezeiten lässt sich an dieser Stelle die slow_copy-Funktion identifizieren, die für Lese- und Schreibzugriffe auf die MMU unerlässlich ist [SPR-23838]. Bei einem Neustart von Lapap MK II hingegen entfällt das Nachladen der CDIs, sofern sie lokal vorhanden sind.

Weitere Abweichungen haben sich für das Instanziieren und Initialisieren der Plug-ins von Lapap MK II gezeigt. Für einige Plug-ins war während des ersten Startvorgangs mehr Zeit notwendig als bei einem Neustart. Die Werte stiegen hier zum Teil bis auf

das Sechsfache an. Obwohl ein Großteil der Ausführungszeiten der Konstruktoren im Millisekundenbereich lag, haben sich die durchschnittlichen Verzögerungen auf zwei Sekunden aufsummiert. Im schlechtesten Fall war sogar die doppelte Zeit gegenüber dem Neustart nötig.

Für die Auslastung des Prozessors zeigen sich im ersten Abschnitt des Startvorgangs hohe Werte von zum Teil mehr als 90%. Die Auswertung belegt, dass diese Werte im Wesentlichen auf die Initialisierung der Plug-ins und des `DMSAdapters` zurückzuführen sind. Die Anzahl der Threads deutet hier zudem auf eine nebenläufige Auslastung hin. Während des ersten Startvorgangs findet eine deutliche Unterbrechung des ersten Abschnitts statt. An dieser Stelle erfolgt das Nachladen der Configuration-Data-Items. Im zweiten Abschnitt des Startvorgangs war die Auslastung des Prozessors geringer. Die Speicherauslastung verlief durchgehend unauffällig.

Daraus folgt, dass das Nachladen der CDIs von der MMU den größten Teil der Verzögerung während des ersten Startvorgangs ausmacht. Als weitere Ursache hat sich das Initialisieren der Plug-ins herausgestellt. Auch hier war mehr Zeit notwendig als bei einem Neustart. Die Auslastung des Prozessors zeigte zwar hohe Werte zu Beginn des Startvorgangs, jedoch ließ sich eine Überlastung ausschließen. Dementsprechend liegt an dieser Stelle keine Performance-Anomalie vor. Viel mehr als die Architektur von Lapap MK II ist es hier die zugrunde liegende Hardware des Columbus Software Systems, die durch unterschiedliche Network-File-Systems den Einsatz von slow_copy erfordert und dadurch den größten Teil der Verzögerung verursacht.

Lösungsansätze

Bezüglich Ziel Z3 ist festzustellen, dass der überwiegende Teil der identifizierten Ursachen nicht innerhalb von Lapap MK II zu finden ist. Die Verzögerungen beim Öffnen von PDF-Dokumenten lassen sich möglicherweise auf die Verwendung des `WebRenderers` zurückführen. Daher könnte es an dieser Stelle hilfreich sein, die aktuellste Version des `WebRenderers` zu verwenden. Der `WebRenderer` verwendet zum Anzeigen von PDF-Dokumenten den auf Betriebssystemebene installierten Adobe Reader. Somit könnte das Aktualisieren der `acroread`-Komponente ebenfalls dazu beitragen, die Ursachen der Verzögerungen zu beheben. Neben dem Adobe Reader gibt es auch andere Anwendungen, die in der Lage sind PDF-Dokumente anzuzeigen. Möglicherweise lässt sich die Verzögerung auch durch das Verwenden einer anderen Anwendung beseitigen.

Um die Verzögerungen durch das Nachladen der Configuration-Data-Items zu minimieren, sind verschiedene Ansätze denkbar. Das Verwenden der gleichen Network-File-Systems auf den On-Board-System-Laptops und dem SPC könnte dazu führen, dass das Laden der CDIs ohne die Verwendung von slow_copy und entsprechend mit einer höheren Geschwindigkeit möglich ist. Dies hat jedoch aufwendige Anpassungen des SPCs oder der Laptops zur Folge. Ein weiterer Ansatz besteht in einer Anpassung des derzeit verwendeten Verfahrens. Durch das permanente Bereithalten einer lokalen Kopie der CDIs könnte der langwierige Ladevorgang auch während des ersten Startvorgangs entfallen. Um die Aktualität der loka-

len CDIs zu gewährleisten, müsste Lapap MK II im Zuge des Startvorgangs eine Verifikation durchführen. Dies könnte beispielsweise mittels eines Prüfsummenverfahrens oder anhand eines Zeitstempels erfolgen. Somit wäre ein Nachladen der CDIs nur dann notwendig, wenn die lokale Kopie von dem Original der MMU abweicht, jedoch nicht automatisch bei jedem Neustart des Laptops.

7.2. Ausblick

Die Analyse des Verhaltens beim Öffnen von PDF-Dokumenten hat bereits gezeigt, dass die innerhalb von Lapap MK II implementierten Methoden nicht die Ursache für die Verzögerung sind. Die Ergebnisse deuten darauf hin, dass die Ursachen durch die Verwendung des `WebRenderers` oder der `acroread`-Komponente gegeben sind. Bisher war es jedoch nicht möglich, diese Vermutungen eindeutig zu belegen. Somit sind an dieser Stelle weitere Analysen notwendig, um auch das Verhalten des `WebRenderers` und der `acroread`-Komponente abschließend beurteilen zu können. Dies könnte ebenfalls mittels dynamischer Analyse erfolgen.

Die Analyse des Startvorgangs von Lapap MK II hat bereits mehrere Stellen aufgezeigt, an denen im Zuge des ersten Starts mehr Zeit nötig ist als bei allen weiteren. Bisher umfasste die Analyse drei Abschnitte des Startvorgangs. Besonders in der zweiten Hälfte, im Anschluss an das Initialisieren des `DMSAdapters`, ist das genaue Verhalten von Lapap MK II aber weiterhin unklar. Somit sind auch hier weitere Untersuchungen mittels dynamischer Analyse denkbar, um möglicherweise weitere Unterschiede und Verzögerungen in den übrigen Abschnitten identifizieren zu können.

Darüber hinaus bietet sich die Verwendung von dynamischer Analyse an, um den Einfluss von Wartungsarbeiten und Anpassungen auf die Performance von Lapap MK II zu überwachen. Bei einem Austausch des PDF-Readers ließe sich beispielsweise analysieren, ob dies einzelne Komponenten von Lapap MK II beeinflusst. Ein Vergleich der Ausführungszeiten der Event-Handler des `WebRenderers` vor und nach dem Austausch könnte entsprechende Auswirkungen auf das Verhalten des `DocumentationViewers` aufzeigen. Besonders bei komplexen Systemen wie dem Columbus-Modul und der ISS können Änderungen an einer Komponente auch Auswirkungen auf andere Komponenten des Systems haben. Auf ähnliche Weise ließe sich auch der Einfluss des Angleichens der Network-File-Systems auf dem Laptop und dem SPC auf den Startvorgang von Lapap MK II analysieren.

Anhang A

Die Testprozeduren

Dieses Kapitel enthält die in Abschnitt 3.4 definierten Testprozeduren in vollem Umfang. Abschnitt A.1 beschreibt zunächst die Prozedur für das Öffnen von PDF-Dokumenten. In Abschnitt A.2 ist die Testprozedur für den Startvorgang von Lapap MK II erläutert.

A.1. Das Öffnen von PDF-Dokumenten

Dieser Abschnitt beschreibt die Prozedur zum Testen des Verhaltens von Lapap MK II beim Öffnen eines PDF-Dokuments. Der Test besteht aus insgesamt zwei Teilen, die jeweils eine andere Konfiguration von Lapap MK II zugrunde legen. Der erste Teil setzt die Standardkonfiguration von Lapap MK II voraus. Unter diesen Bedingungen sollen insgesamt 25 Durchführungen der Prozedur erfolgen. Der zweite Teil des Tests setzt voraus, dass der maximal nutzbare Heap-Speicher für Lapap MK II in dem entsprechenden Shell-Skript auf 64 Megabyte beschränkt ist. Mit dieser Konfiguration sollen ebenfalls 25 Durchführungen erfolgen. Die folgende Auflistung enthält die fortlaufend nummerierten Schritte sowie Erläuterungen zu den erwarteten Ergebnissen:

1. Press <Ctrl><4>.
   - ▻ Documentation becomes actual workspace.
2. Enter search term “CTCU 1” and select “Displays”, “Titles” and “Search”.
   - ▻ A list of USS displays, with CTCU 1 at the top, is displayed.
3. Select link [Synoptic Display] CTCU 1.
   - ▻ Synoptics becomes the actual workspace.
   - ▻ The display opens.
4. Enter “3.301” and select Search.
   - ▻ Documentation is the actual workspace.
   - ▻ A list of results for “3.301” is visible.
5. Select “3.301 Cabin Air Return Grid Clogging”.

   ▻ Synoptics & Procedures becomes actual workspace.
   ▻ The PDF file opens.

6. Select Documentation from the ComboBox.

   ▻ Documentation becomes actual workspace.

7. Select “Lapap MK II User Manual” shown within the documentation overview.

   ▻ Lapap MK II Manual opens.

8. Scroll downwards and select chapter “Situational Awareness Icons” in “Situational Awareness”.

   ▻ Situational Awareness Icons chapter opens.

9. Select Back two times.

   ▻ Page Documentation with five links is visible.

10. Select link “Laptop System User Manual”.

    ▻ Acrobat reader opens with Laptop SW User Manual.

11. Scroll downwards and upwards.

    ▻ PDF documentation is scrollable.

12. Select Acrobat Reader +.

    ▻ Display is magnified.

13. Select Acrobat Reader -.

    ▻ Scaled down.

14. Select Back.

    ▻ Page Documentation with five links is visible.

15. Select “File Transfer Browser (FTB) User Manual”.

    ▻ FTB SOFTWARE USER’S MANUAL opens.

16. Select Back.

    ▻ Page Documentation with five links is visible.

17. Select “CLSW Configuration User Manual”.

    ▻ CLSW Maintenance LaptopSW User Manual opens.

18. Select Back.

    ▻ Page Documentation with five links is visible.

19. Select “Glossary”.

    ▻ Glossary opens.

20. Select “RefDoc” from Crumb Trail.

    ▻ Page Documentation with five links is visible.

21. Shut down Lapap MK II.

    ▻ Lapap MK II is closed.

22. Create a folder within the “part1” folder (see Figure 1). These folders should be numbered consecutively starting with 1.

    ▻ Folder is created.

23. Open /tmp/ and move the Kieker folder named similar to “kieker-20130822-124151359-UTC-DEL01184-KIEKER” to the folder created in step 22. Open /col_data/lapap/usr and move the log files generated by Lapap MK II to the same folder. Open /col_-data/dms/logs and COPY the log file to the folder.

    ▻ Data is collected.

24. Restart Lapap MK II, if you want to continue.

    ▻ Lapap MK II is started.

A.2. Der Startvorgang von Lapap MK II

In diesem Abschnitt ist die Prozedur zum Testen des Startvorgangs von Lapap MK II beschrieben. Dieser Test besteht aus zwei Teilen. Der erste Teil untersucht das Verhalten bei einem Neustart von Lapap MK II. Für diesen Test sollen 30 Durchführungen der Prozedur erfolgen. Der zweite Teil untersucht den ersten Startvorgang von Lapap MK II, der automatisch nach dem Starten des Betriebssystems erfolgt. Auch hier sind insgesamt 30 Durchgänge durchzuführen. Die folgende Auflistung enthält die fortlaufend nummerierten Schritte der Prozedur sowie Erläuterungen zu den erwarteten Ergebnissen. Für den letzten Schritt sind beide Varianten aufgeführt. Dies umfasst den Neustart von Lapap MK II für den ersten Teil sowie den Neustart des Laptops für den zweiten Teil des Tests.

1. Start Lapap MK II.

    ▻ Lapap MK II is started.

2. Wait until Lapap MK II finished startup process.

   ▻ The DMS status display should be green just as the labels within the synoptic displays.

3. Shut down Lapap MK II.

   ▻ Lapap MK II is closed.

4. Create a folder within the “part1”-folder. These folders should be numbered consecutively starting with 1.

   ▻ The folder is created.

5. Open /tmp/ and move the Kieker folder named similar to “kieker-20130822-124151359-UTC-DEL01184-KIEKER” to the created folder. Open /col_data/lapap/usr and move the log files generated byLapap MK II to the created folder. Open /col_data/dms/logs and COPY the log file to the created folder.

   ▻ Log files cleared.

6. Restart Lapap MK II. / Reboot the laptop.

   ▻ Lapap MK II is restarted. / The laptop is rebooted.

Anhang B

Statistische Werte für die Initialisierung der Plug-ins

Dieses Kapitel enthält weitere statistische Werte für die Ausführungszeiten von Methoden und Konstruktoren der Plug-ins. In Abschnitt B.1 sind entsprechende Werte für den Neustart von Lapap MK II dargestellt. Die Werte für den ersten Startvorgang von Lapap MK II im Anschluss an den Bootvorgang sind in Abschnitt B.2 enthalten.

B.1. Neustart von Lapap MK II

Dieser Abschnitt enthält die statistischen Werte für die Initialisierung der Plug-ins während des Neustarts von Lapap MK II. In Tabelle B.1 sind die Werte für die Plug-ins aufgeführt, deren Initialisierung ausschließlich durch den Konstruktor erfolgt. Die Werte für die übrigen Plug-ins, für deren Initialisierung zusätzlich ein Aufruf der Methode `performInitialization()` notwendig ist, sind in Tabelle B.2 dargestellt.

Tabelle B.1. Statistische Werte für die Ausführungszeiten der Konstruktoren beim Instanziieren der Plug-ins in Millisekunden während des Neustarts von Lapap MK II

Plug-in	Min	Max	$\bar{x}$	$\tilde{x}$
SMPPlugin	29,34	38,13	33,64	33,62
HotKeyHelpPlugin	0,34	0,51	0,36	0,35
UncaughtExceptionHandlingPlugin	2,55	45,93	4,89	2,73
ODFEventCodePlugin	16,23	58,28	38,11	37,23
ResetFunctionPlugin	1,55	9,64	2,99	2,39
ODFBookBrowserPlugin	133,94	216,86	148,93	145,34
ServerConnectionHandlingPlugin	16,21	19,94	17,21	17,06
SystemInterfacePlugin	14,09	50,74	18,32	15,15
ODFProcPlugin	1243,36	1827,19	1362,11	1312,11
DMSMsgLoggerPlugin	37,21	101,08	84,45	92,30
UserLoggerPlugin	0,19	0,24	0,20	0,20
MessageProviderPlugin	13,96	78,71	19,19	15,01

Tabelle B.1. Statistische Werte für die Ausführungszeiten der Konstruktoren beim Instanziieren der Plug-ins in Millisekunden während des Neustarts von Lapap MK II

Plug-in	Min	Max	$\bar{x}$	$\tilde{x}$
StatusPlugin	65,81	124,32	111,67	115,37
StandardDisplayNavigationCommandPlugin	2,48	208,93	36,18	9,46
SearchPlugin	64,42	118,07	95,48	96,83

Tabelle B.2. Statistische Werte für die Ausführungszeiten der Konstruktoren beim Instanziieren der Plug-ins sowie der Methode `performInitialization()` in Millisekunden während des Neustarts von Lapap MK II

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
DisplayHierarchyPlugin	<init>()	11,68	50,86	21,98	16,25
	performInitialization()	0,11	0,13	0,12	0,12
DisplayPerformanceIndicatorPlugin	<init>()	22,80	207,20	64,52	43,67
	performInitialization()	0,02	3,66	0,14	0,02
ToolTipConfigurationPlugin	<init>()	2,20	3,21	2,57	2,51
	performInitialization()	0,35	5,31	0,62	0,38
DocPlugin	<init>()	185,17	577,72	383,57	369,83
	performInitialization()	0,01	0,02	0,01	0,01
USSPlugin	<init>()	95,07	142,03	109,85	106,41
	performInitialization()	147,03	271,46	170,30	163,61
DTGPlugin	<init>()	13,73	89,98	18,44	14,84
	performInitialization()	139,88	281,72	257,51	264,83
DataPoolLoggerPlugin	<init>()	22,56	202,06	42,98	37,77
	performInitialization()	13,09	68,15	22,84	21,20

B.2. Neustart des Laptops

Die Werte für die Initialisierung der Plug-ins während des ersten Startvorgangs sind in diesem Abschnitt zusammengefasst. Tabelle B.3 enthält Werte für die Plug-ins, deren Initialisierung ausschließlich durch den Konstruktor erfolgt. Die Werte für die übrigen Plug-ins, für deren Initialisierung neben dem Konstruktor ein Aufruf der Methode `performInitialization()` notwendig ist, sind in Tabelle B.4 dargestellt.

Tabelle B.3. Statistische Werte für die Ausführungszeiten der Konstruktoren beim Instanziieren der Plug-ins in Millisekunden für den ersten Start von Lapap MK II im Anschluss an den Neustart des Laptops

Plug-in	Min	Max	$\bar{x}$	$\tilde{x}$
SMPPlugin	66,28	166,62	78,11	74,84
HotKeyHelpPlugin	0,33	0,36	0,34	0,34
UncaughtExceptionHandlingPlugin	2,55	6,05	3,21	2,73
ODFEventCodePlugin	16,24	45,36	27,75	26,28
ResetFunctionPlugin	9,88	144,99	18,60	14,02
ODFBookBrowserPlugin	166,07	370,41	207,28	201,96
ServerConnectionHandlingPlugin	23,63	82,47	27,98	24,48
SystemInterfacePlugin	14,06	104,01	41,20	39,80
ODFProcPlugin	1432,89	2919,92	1703,81	1582,55
DMSMsgLoggerPlugin	52,45	110,38	89,32	93,89
UserLoggerPlugin	0,19	0,21	0,20	0,20
MessageProviderPlugin	18,58	164,90	54,92	29,21
StatusPlugin	102,24	342,12	133,46	120,00
StandardDisplayNavigationCommandPlugin	2,48	176,54	79,50	83,65
SearchPlugin	75,00	121,20	87,47	86,93

Tabelle B.4. Statistische Werte für die Ausführungszeiten der Konstruktoren beim Instanziieren der Plug-ins sowie der Methode `performInitialization()` in Millisekunden für den ersten Start von Lapap MK II im Anschluss an den Neustart des Laptops

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
DisplayHierarchyPlugin	<init>()	11,76	61,57	23,70	17,36
	performInitialization()	0,11	1,34	0,20	0,12
DisplayPerformanceIndicatorPlugin	<init>()	38,1	395,55	99,36	83,00
	performInitialization()	0,02	0,02	0,02	0,02
ToolTipConfigurationPlugin	<init>()	2,20	11,03	3,03	2,48
	performInitialization()	0,36	0,39	0,38	0,37
DocPlugin	<init>()	639,15	1569,45	745,16	697,96
	performInitialization()	0,01	0,01	0,01	0,01
USSPlugin	<init>()	201,16	449,01	229,70	216,48
	performInitialization()	235,80	484,55	300,70	287,12
DTGPlugin	<init>()	13,37	100,14	19,43	13,97
	performInitialization()	206,50	418,40	297,44	297,69

Tabelle B.4. Statistische Werte für die Ausführungszeiten der Konstruktoren beim Instanziieren der Plug-ins sowie der Methode `performInitialization()` in Millisekunden für den ersten Start von Lapap MK II im Anschluss an den Neustart des Laptops

Klasse	Methode	Min	Max	$\bar{x}$	$\tilde{x}$
DataPoolLoggerPlugin	<init>()	24,43	186,30	57,07	45,32
	performInitialization()	12,51	40,78	21,52	20,49

Anhang C

Referenzdaten für die Auslastung der Ressourcen

Die in Abschnitt 4.2 beschriebenen Referenzdaten sind in diesem Kapitel dargestellt. Die Erhebung der Daten erfolgte vor der Durchführung der Testprozeduren. Abschnitt C.1 beschreibt die Auslastung der Ressourcen ohne die Ausführung von Lapap MK II. Die Auslastung der Ressourcen während der Ausführung von Lapap MK II ist in Abschnitt C.2 beschrieben.

C.1. Auslastung der Ressourcen ohne Lapap MK II

Abbildung C.1 zeigt die Auslastung der Ressourcen ohne die Ausführung von Lapap MK II. Die Aufzeichnung der Daten erfolgte mit einer Frequenz von 4 Hz.

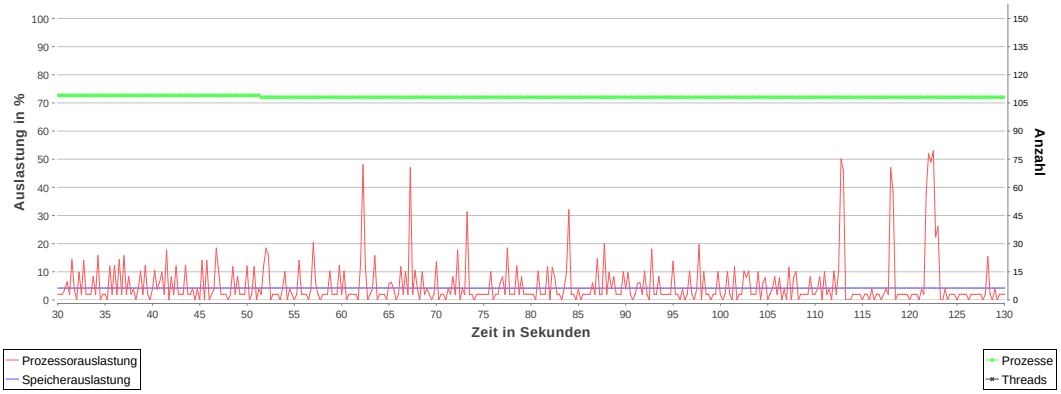

Abbildung C.1. Auslastung der Ressourcen ohne die Ausführung von Lapap MK II; aufgezeichnet mit einer Frequenz von 4 Hz

C.2. Auslastung der Ressourcen mit Lapap MK II

Abbildung C.2 zeigt die Auslastung der Ressourcen während der Ausführung von Lapap MK II. Die Aufzeichnung der Daten erfolgte mit einer Frequenz von 4 Hz. In Abbildung C.3 ist ebenfalls die Auslastung der Ressourcen während der Ausführung von Lapap MK II dargestellt, jedoch erfolgte die Aufzeichnung der Daten hier mit einer Frequenz von nur 2 Hz.

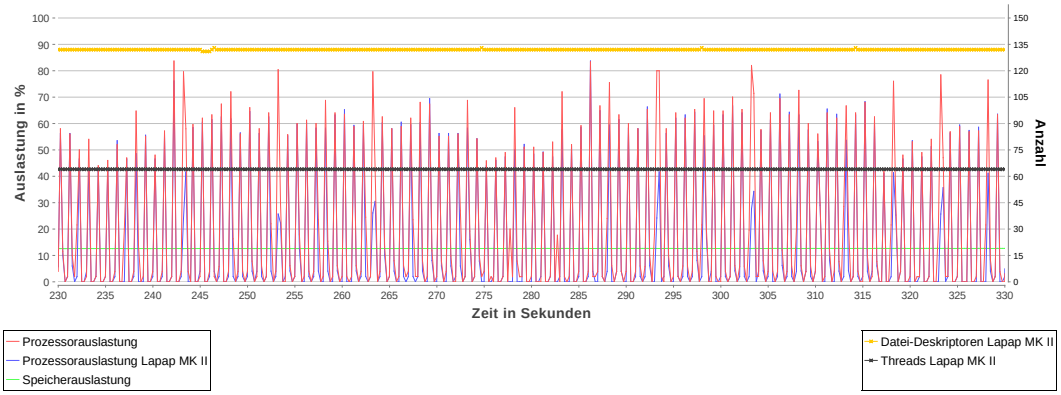

Abbildung C.2. Auslastung der Ressourcen während der Ausführung von Lapap MK II; aufgezeichnet mit einer Frequenz von 4 Hz

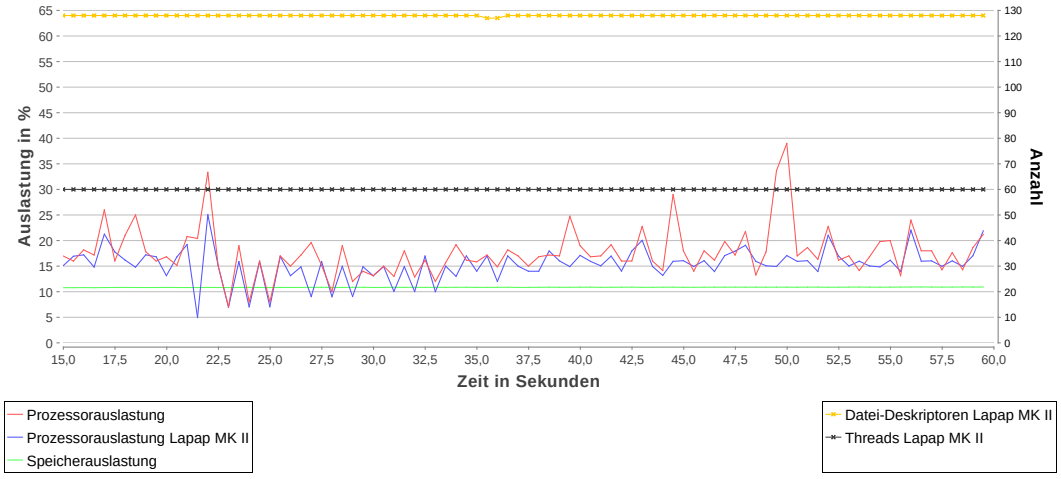

Abbildung C.3. Auslastung der Ressourcen während der Ausführung von Lapap MK II; aufgezeichnet mit einer Frequenz von 2 Hz

Literaturverzeichnis

[AspectJ] AspectJ language extension. Eclipse Foundation. URL: http://www.eclipse.org/aspectj/. (Siehe Seite 11)

[Bensalem u. a. 2006] S. Bensalem, J.-C. Fernandez, K. Havelund und L. Mounier. Confirmation of Deadlock Potentials Detected by Runtime Analysis. In: *Proceedings of the 2006 Workshop on Parallel and Distributed Systems: Testing and Debugging*. PADTAD '06. Portland, Maine, USA: ACM, 2006, Seiten 41–50. (Siehe Seite 79)

[Canfora u. a. 2011] G. Canfora, M. Di Penta und L. Cerulo. Achievements and challenges in software reverse engineering. *Communications of the ACM* 54.4 (Apr. 2011), Seiten 142–151. (Siehe Seiten 8–10)

[Chandola u. a. 2009] V. Chandola, A. Banerjee und V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)* 41.3 (Juli 2009), 15:1–15:58. (Siehe Seiten 5–7)

[Cherkasova u. a. 2008] L. Cherkasova, K. Ozonat, N. Mi, J. Symons und E. Smirni. Anomaly? Application Change? or Workload Change? Towards Automated Detection of Application Performance Anomaly and Change. In: *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*. 2008, Seiten 452–461. (Siehe Seite 7)

[Cherkasova u. a. 2009] L. Cherkasova, K. Ozonat, N. Mi, J. Symons und E. Smirni. Automated Anomaly Detection and Performance Modeling of Enterprise Applications. *ACM Transactions on Computer Systems* 27.3 (Nov. 2009), 6:1–6:32. (Siehe Seite 7)

[Cheung und Madden 2008] A. Cheung und S. Madden. Performance Profiling with EndoScope, an Acquisitional Software Monitoring Framework. *Proceedings of the VLDB Endowment* 1.1 (Aug. 2008), Seiten 42–53. (Siehe Seite 78)

[Chikofsky und Cross 1990] E. Chikofsky und I. Cross J.H. Reverse engineering and design recovery: A taxonomy. *Software, IEEE* 7.1 (1990), Seiten 13–17. (Siehe Seite 8)

[Columbus System Laptop ADD]. Columbus System Laptop Software Architectural Design Document. Unternehmensinterne Dokumentation, Dokumenten-Nummer: ESO-IT-ADD-0004. Astrium GmbH. Bremen, DE, März 2013. (Siehe Seiten 12, 14, 16 und 71)

[Cornelissen u. a. 2009] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen und R. Koschke. A systematic survey of program comprehension through dynamic analysis. *Software Engineering, IEEE Transactions on* 35.5 (2009), Seiten 684–702. (Siehe Seite 10)

[Ernst 2003] M. D. Ernst. Static and dynamic analysis: synergy and duality. In: *Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society, 2003, Seiten 25–28. (Siehe Seiten 9, 10)

[Hanenberg u. a. 2009] S. Hanenberg, S. Kleinschmager und M. Josupeit-Walter. Does aspect-oriented programming increase the development speed for crosscutting code? an empirical study. In: *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. ESEM '09. Washington, DC, USA: IEEE Computer Society, 2009, Seiten 156–167. (Siehe Seite 10)

[Joshi u. a. 2010] P. Joshi, M. Naik, K. Sen und D. Gay. An Effective Dynamic Analysis for Detecting Generalized Deadlocks. In: *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE '10. Santa Fe, New Mexico, USA: ACM, 2010, Seiten 327–336. (Siehe Seite 79)

[Kiczales u. a. 1997] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier und J. Irwin. Aspect-oriented programming. In: *ECOOP'97 — Object-Oriented Programming*. Band 1241. Springer, 1997, Seiten 220–242. (Siehe Seiten 10, 12 und 17)

[Kieker Project 2013] Kieker Project. Kieker 1.7 User Guide. Arbeitsgruppe Software Engineering , Christian-Albrechts-Universität zu Kiel, Kiel, Deutschland. Apr. 2013. (Siehe Seiten 17, 37 und 39)

[Knoche u. a. 2012] H. Knoche, A. van Hoorn, W. Goerigk und W. Hasselbring. Automated Source-Level Instrumentation for Dynamic Dependency Analysis of COBOL Systems. In: *Proceedings of the 14. Workshop Software-Reengineering (WSR '12)*. Also appeared in Softwaretechnik-Trends 32(2) (May 2012) 45-46. Mai 2012, Seiten 33–34. (Siehe Seiten 8, 17 und 78)

[Koschke 2009] R. Koschke. Architecture Reconstruction. In: *Software Engineering*. Band 5413. Springer, 2009, Seiten 140–173. (Siehe Seite 9)

[Lapap MK II Manual]. Lapap Mk II Manual. Unternehmensinterne Dokumentation. Astrium GmbH. Bremen, DE, 2011. (Siehe Seiten 13 und 25)

[Lapap MK II SDD]. Lapap MK II Software Design Document. Unternehmensinterne Softwaredokumentation, Dokumenten-Nummer: COL-RIBRE-ADD-0035. Astrium GmbH. Bremen, DE, Okt. 2012. (Siehe Seiten 12–16 und 24)

[Log4j] Apache Log4j. The Apache Software Foundation. URL: `http://http://logging.apache.org/log4j/2.x/`. (Siehe Seite 24)

[Matevska 2013] J. Matevska. ISS Columbus Module On-Board Software Maintenance. In: *Proceedings of the Software Engineering 2013 Workshop*. Aachen: Gesellschaft für Informatik (GI), März 2013, Seiten 209–214. (Siehe Seiten 2, 19, 21, 22)

[Navigator Framework Documentation]. Introduction to the MobileMentor Navigator Framework - Developers Documentation. Unternehmensinterne Dokumentation. Astrium GmbH. Bremen, DE, 2006. (Siehe Seite 14)

[Nierstrasz und Demeyer 2004] O. Nierstrasz und S. Demeyer. Object-Oriented Reengineering Patterns. In: *Proceedings of the 26th International Conference on Software Engineering*. ICSE '04. Washington, DC, USA: IEEE Computer Society, 2004, Seiten 734–735. (Siehe Seite 1)

[Ritsch und Sneed 1993] H. Ritsch und H. Sneed. Reverse engineering programs via dynamic analysis. In: *Proceedings of the Working Conference on Reverse Engineering*. 1993, Seiten 192–201. (Siehe Seiten 1 und 77)

[Riva und Rodriguez 2002] C. Riva und J. Rodriguez. Combining static and dynamic views for architecture reconstruction. In: *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*. 2002, Seiten 47–55. (Siehe Seite 78)

[Rohr u. a. 2008] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, S. Giesecke und W. Hasselbring. Kieker: Continuous Monitoring and on demand Visualization of Java Software Behavior. In: *Proceedings of the IASTED International Conference on Software Engineering 2008 (SE'08)*. ACTA Press, Feb. 2008, Seiten 80–85. (Siehe Seite 18)

[Shneiderman 1984] B. Shneiderman. Response Time and Display Rate in Human Performance with Computers. *ACM Computing Surveys (CSUR)* 16.3 (Sep. 1984), Seiten 265–285. (Siehe Seite 7)

[SIGAR] Hyperic SIGAR API. Hyperic, Inc, 2013. URL: http://www.hyperic.com/products/sigar. (Siehe Seite 18)

[Sneed 2005] H. Sneed. An Incremental Approach to System Replacement and Integration. In: *Ninth European Conference on Software Maintenance and Reengineering, 2005*. 2005, Seiten 196–206. (Siehe Seite 1)

[SPR-23806]. Columbus System Problem Report Database, Eintrag 23806: Excessive load time for pdf documents/procedures. Unternehmensinterne Dokumentation. Astrium GmbH. Bremen, DE, 2012. (Siehe Seite 23)

[SPR-23838]. Columbus System Problem Report Database, Eintrag 23838: Copy command blocks MMU. Unternehmensinterne Dokumentation. Astrium Space GmbH. Bremen, DE, 2013. (Siehe Seiten 71 und 82)

[SPR-25408]. Columbus System Problem Report Database, Eintrag 25408: LAPAP start fails on repeating re-start. Unternehmensinterne Dokumentation. Astrium GmbH. Bremen, DE, 2013. (Siehe Seite 75)

[SSP 50313]. Display and Graphics Commonality Standard. SSP 50313. Revision C. Sep. 2005. (Siehe Seite 13)

[Stingl 2004] P. Stingl. Mathematik für Fachhochschulen - Technik und Informatik. 7. überarbeitete Auflage. Hanser Verlag, 2004. (Siehe Seiten 7, 8)

[TS IX]. APM Software System Test Scenarios - Test Scenario IX: Crew Operations. Unternehmensinternes Testszenario. Astrium GmbH. Bremen, DE, 2012. (Siehe Seiten 21, 23 und 40)

[Van Deursen u. a. 2004] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen und C. Riva. Symphony: view-driven software architecture reconstruction. In: *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture*. WICSA '04. Washington, DC, USA: IEEE Computer Society, 2004, Seiten 122–. (Siehe Seite 9)

[Van Hoorn u. a. 2009] A. van Hoorn, M. Rohr, W. Hasselbring, J. Waller, J. Ehlers, S. Frey und D. Kieselhorst. Continuous Monitoring of Software Services: Design and Application of the Kieker Framework. Forschungsbericht. Christian-Albrechts-Universität zu Kiel, Nov. 2009. (Siehe Seiten 17, 18)

[Van Hoorn u. a. 2011a] A. van Hoorn, S. Frey, W. Goerigk, W. Hasselbring, H. Knoche, S. Köster, H. Krause, M. Porembski, T. Stahl, M. Steinkamp und N. Wittmüss. DynaMod Project: Dynamic Analysis for Model-Driven Software Modernization. In: *Joint Proceedings of the 1st International Workshop on Model-Driven Software Migration (MDSM 2011) and the 5th International Workshop on Software Quality and Maintainability (SQM 2011)*. Band 708. CEUR Workshop Proceedings. Invited paper. März 2011, Seiten 12–13. (Siehe Seite 78)

[Van Hoorn u. a. 2011b] A. van Hoorn, H. Knoche, W. Goerigk und W. Hasselbring. Model-driven instrumentation for dynamic analysis of legacy software systems. In: *Proceedings of the 13. Workshop Software-Reengineering (WSR 2011)*. Also appeared in Softwaretechnik-Trends 31(2) (May 2011) 18?19. Bad Honnef, Deutschland, May 2-4, 2011, Mai 2011, Seiten 26–27. (Siehe Seite 17)

[Van Hoorn u. a. 2012] A. van Hoorn, J. Waller und W. Hasselbring. Kieker: a framework for application performance monitoring and dynamic software analysis. In: *Proceedings of the 3rd joint ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*. ACM, Apr. 2012, Seiten 247–248. (Siehe Seiten 17, 18 und 68)

[Van Hoorn u. a. 2013] A. van Hoorn, S. Frey, W. Goerigk, W. Hasselbring, H. Knoche, S. Köster, H. Krause, M. Porembski, T. Stahl, M. Steinkamp und N. Wittmüss. DynaMod: Dynamische Analyse für modellgetriebene Software-Modernisierung. Forschungsbericht. Department of Computer Science, Kiel University, Kiel, Germany, Juli 2013. (Siehe Seite 78)

[Walker u. a. 1999] R. J. Walker, E. L. A. Baniassad und G. C. Murphy. An initial assessment of aspect-oriented programming. In: *Proceedings of the 21st international conference on Software engineering*. ICSE '99. Los Angeles, California, USA: ACM, 1999, Seiten 120–130. (Siehe Seite 10)

[WebRenderer] WebRenderer. JadeLiquid Software Pty Ltd. URL: `http://www.webrenderer.com/index.php`. (Siehe Seite 26)

[WebRenderer API] WebRenderer API Dokumentation. JadeLiquid Software Pty Ltd. URL: `http://webrenderer.com/products/desktop/developer/doc/index.html`. (Siehe Seiten 28 und 30)

[Wechselberg 2013] N. B. Wechselberg. Monitoring von Perl-basierten Webanwendungen mittels Kieker. Bachelorarbeit. Kiel University, Apr. 2013. (Siehe Seite 79)

[Zaidman u. a. 2006] A. Zaidman, B. Adams, K. De Schutter, S. Demeyer, G. Hoffman und B. De Ruyck. Regaining lost knowledge through dynamic analysis and aspect orientation. In: *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*. 2006, 10 pp.–102. (Siehe Seiten 1 und 78)