PG 458 3Debug

Abbildungsverzeichnis

KAPITEL 1

Einleitung

Alexander Fronk, Jens Schröder, Sören Blom

KAPITEL 2

2.3 Konstruktion

Die Entwicklung des Debuggers folgt dem XP-Ansatz, der sich bereits bei der Durchführung der Projektgruppen 415 und 444 am Lehrstuhl für Software-Technologie bewährt hat. Auf Grund der ganzheitheitlichen Teamorientierung und der Eigenverantwortlichkeit der einzelnen Entwickler, die auch die selbstständige aber angeleitete Planung der auszuführenden Tätigkeiten umfasst, bietet sich XP als Entwicklungsprozess auch für diese Projektgruppe an.

Dem XP-Ansatz folgend ergibt sich eine inkrementelle Entwicklung mit vielen kleineren Releases, bei der die frühen Releases einen eher prototypischen Charakter haben, die dann zu einen vollständigen System führen.

2.4 Berichte

Die gesamten Arbeiten werden jeweils durch einen *Zwischen-* und einen *Abschlussbericht* dokumentiert.

Der Zwischenbericht dokumentiert die Ergebnisse des ersten Semesters, insbesondere die Anforderungsanalyse und die ersten Releases des zu erstellenden Systems.

KAPITEL 3

Abbildung 4.3.: von Testpersonen bevorzugte Realisierungen, Quelle: Irani u. a.

Abbildung 4.5.: Walrus-Visualisierung einer Verzeichnisstruktur, Quelle: CAIDA (2004)

Das Hauptziel besteht darin, die Darstellung so zu optimieren, dass sie für den Betrachter wichtige Beziehungen und strukturelle Merkmale in den Daten aufdeckt. Dies kann beispielsweise das *Clustering* beinhalten. Cluster sind hier Partitionen von Knoten, die z. B. dadurch gekennzeichnet sind, dass innerhalb eines Clusters alle Knoten stark miteinander (durch Kanten) verbunden sind, aber nur wenige Verbindungen hinausführen. Hierbei müssen Möglichkeiten gefunden werden, den Berechnungsaufwand zu reduzieren.

4.3.1

Abbildung 4.7.: einfaches UML-Klassendiagramm (links) und zwei Walrus-Visualisierungen mit unterschiedlichen Spannbäumen.

Neben der Definition von Knoten und gerichteten Kanten können in Walrus auch Attribute

Abbildung 4.9.:

dimensionalen Objekten: Assoziationslinien zu Röhren, Klassenrechtecke zu flachen Boxen

KAPITEL 5

Fehler durch Nebenläufigkeit durch gleichzeitigen Zugriff auf eine gemeinsame Variable/Resource. Diese Fehler können sporadisch auftreten, da viele Betriebssysteme und Laufzeitumgebungen mit nichtdeterministischen Schedulern arbeiten.

und unzählig viele weitere Gründe.

Neben diesen Programmierfehlern haben Entwicklungsteams auch mit ganz anderen Problemen zu kämpfen:

Designbugs resultieren aus schlechter Planung des Zusammenspiels einzelner Softwarekomponenten. Diese Fehler lassen nicht das Programm abstürzen, sie werfen aber den ganzen Programmierstil und den Einsatz der (falschen) Designpatterns über den Haufen. Zum Beispiel kann eine Referenz auf ein bestimmtes Objekt benötigt werden, die Kapselung der gesamten Softwarekomponente verhindert aber den direkten Zugriff. Hässlicher, unsauberer Programmcode mit „Quickhacks“ ist die Folge.

Planungsbugs

Abbildung 5.2.: Ausführungskontrolle in Eclipse

Abbildung 5.3.: Variablendarstellung in Eclipse

Quellcodeebene.

Insbesondere im Debugging ist beim Aufstellen von Hypothesen und Testen von Lösungen das Situation Model am wichtigsten und somit eine stärkere Unterstützung des Programmierers sehr wünschenswert.

5.3 Grafische Tools

Grafische Ansätze zur Darstellung von Programminformationen könnten die Lücke für das Situation Model füllen. Grafiken werden in der Regel schneller aufgenommen, verarbeitet und besser im Gedächtnis gespeichert als rein textuelle Darstellungen und bieten eine natürliche Abstraktion des Codes, ohne die Struktur zu verdecken. Zudem sind Diagramme oft auch intuitiver zu bedienen und zu manimulieren sowie von sprachlichen Grenzen unabhnngiger. Allerdings erfordert die grafische Informationsdarstellung meist schon in einfachen Formen einen deutlich gesteigerten Ressourceneinsatz, der z.B. die Darstellung zumfassender Debugging-Informationen in Echtzeit zur Laufzeit des Programmes nahezu unmöglich erscheinen lässt.

Abbildung 5.7.:

bieten aber nicht immer die gerade benötigten Informationen in der geeigneten Form an.

KAPITEL 6

Debugging-Werkzeuge

Boris Brodski

6.1 Einleitung

Dies ist eine Seminarausarbeitung der PG 458, die wichtige Prinzipien und Konzepte eines Debuggers vorstellt. Für den Vortrag und die Ausarbeitung wurde ausschließlich das Buch Rosenberg

her un zu analysi5ger. Meistens werder Debugger (nuötig,t)-259(um)-258(Pogramm,fehler)-259(zu)-
der oftvor Softwareentwicklerr undTesterr (nutzlt.)-285(Die)-28[(Idee)-285(des)-285(Deb)20(uggers
übernimm,t um das Pogramm, stopper und schrittweise ausführer zu könner. Mar kanr der
Kontext des Pogramm,s ar jede Stelle üb Der Kontext fast die gesamte Information
über den Ablauf des Pogramm,s zusam,er. Er beinhaltet zum Beispiel die Belegung aller
V den Zustand des Pogzessors, die Zeile im Pogramm,t die gerade ausgeführt wird

geführt wird. Die tatsächlich durchgeführten Prozessorschritte werden hinter dem Programmquelltext versteckt.

Im folgenden Kapitel werden Heisenberg Prinzipien, die bei der Erstellung einer Debugger zu beachten sind, vorgestellt. In Kapitel 6.3 wird der Kontext präsentiert. In Kapitel 6.4 werden wichtige Bestandteile eines Debuggers erläutert. In Kapitel 6.5 werden drei wichtige

beim Debuggen vom optimierten Code, weil die Umkehrung von Optimierung eine besonders schwierige Aufgabe ist (siehe Kapitel

rekursiven Aufruf von Funktionen.

In großen Programmen werden oft gleiche Funktionen von verschiedenen Stellen aufgerufen. Es bedeutet, dass es nicht ausreicht zu wissen, welche Funktionen gerade ausgeführt

abonniert. Die Programmstoppbenachrichtigung wird benötigt, um die Datenstrukturen freizugeben und den Nutzer das Programmenden zu melden. Benachrichtigung über die Erzeugung bzw. Zerstörung eines Thread. Diese Benachrichtigungen werden benötigt,

6.5.3 Step Over Algorithmus

Step Over bedeutet „Führe nächste Zeile des Programmquelltext aus und stoppe. Dabei sollen alle inzwischen aufgerufenen Funktionen mit volle Geschwindigkeit ausgeführt werden”.

In Vergleich mit dem Step Into Algorithmus ist der Step Over Algorithmus einfacher zu realisieren. Es gibt mehrere Strategien, die man verfolgen kann. Man kann jeden einzelnen Befehl ausführen und die Situation analysieren, was nach wie vor sehr ineffizient ist. Man kann

KAPITEL 7

Mathematische Grundlagen der 3D Modellierung

7.1.3 Verknüpfungen

u

Fr Vektoren gibt es zwei wichtige Verknüpfungen, eine ist die Vektoraddition, die andere die Skalarmultiplikation. Sollen die beiden Vektoren $u, v \quad \mathsf{R}^n$ addiert werden, ergibt sich

$$u + v = \begin{matrix} \square \\ \square \\ \square \\ \square \\ \square \end{matrix} \begin{matrix} \\ u_0 \\ u_1 \\ \\ \end{matrix}$$

7.2 Matrizen

In der 3D-Modellierung müssen oft Objekte verschoben, rotiert oder skaliert werden. Man verwendet Matrizen, um diese Operationen durchzufhren. Das bedeutet, es können sowohl die Objekte in 3D als auch Transformationen auf diesen Objekten durch Matrizen beschrieben werden. Begriffe und Operationen, die dafür benötigt werden, werden hier vorgestellt.

7.2.1 Matrix

Eine

7.2.3 Matrizenmultiplikation

dem Normalenvektor der Ebene, ist das Ergebnis Null. Das liegt daran, dass der Normalenvektor senkrecht auf dem anderen Vektor steht und das Skalarprodukt von senkrechten Vektoren Null liefert. Die implizite Form einer Ebene lautet also

$$(q - u) \cdot n$$

Die andere Art der Projektion ist die perspektivische Projektion. Der

Transformationsmatrizen in umgekehrter Reihenfolge angewendet, etwa

(T_4

KAPITEL 8

Java3D - Eine Einführung

Henning Zeller

8.1

Entwickler werden dabei auf Basis einer High-Level-API Werkzeuge zur Arbeit mit dreidimensionalen, geometrischen Objekten zur Verfügung gestellt. Diese Objekte befinden sich innerhalb eines sogenannten virtuellen Universums, das gerendert werden soll. Das Rendering geschieht automatisch, wahlweise mit den Grafikbibliotheken OpenGL oder DirectX, die von Java3D beide unterstützt werden.

Mit Java3D verfolgte Sun hauptsächlich die Idee, objektorientierten Prinzipien Einzug in die

8.3 Das Szenegraph-Konzept

Wir wollen zunächst überlegen, wie sich ein Szenegraph charakterisieren lässt und welche

Lizenz hat es den großen Vorteil, dass es als universelles Tool einsetzbar ist. Dem Eclipse-

KAPITEL 10

Das Eclipse Plug-In Modell

Boris Düdder

10.1 Einleitung

10.3.2 Features

Ein Feature stellt eine logische Gruppierung von Plug-Ins dar. Diese Gruppierung ist meistens Aufgaben- oder Themengebunden, wie z.B. ein Feature für die Java-Entwicklung. Der Name des Features korrespondiert mit dem Namen eines Plug-Ins, dem sogenannten Feature-Plug-In. Zusätzlich bietet das Feature-Konzept für die Plug-Ins noch einige wichtige Bonbons. Ein Feature kann eine Begrüssungsseite, eine anfängliche Perspektive sowie Cheat-Sheets, d.h.

10.4.1

KAPITEL 11

Debugging APIs

Jonas Mathis

11.1 Einführung

Die Ausarbeitung zum PG-Seminar gibt eine kurze Einführung in die internen Abläufe beim Debugging einer Java-Applikation. Zum einen werden dabei Konzepte der Java-API, wie auch zum anderen Konzepte der Debugging-API der Eclipse-Entwicklungsumgebung angesprochen. Der Text beruht insbesondere auf den Online-Artikeln Szurszewski (2003) und Wright und Freeman-Benson

Back-end Das in einer „nativen“ Sprache geschriebene Back-end leitet Anfragen vom Front-

11.3.2 Platform Debug Model

Das Eclipse-eigene Platform Debug Model ist eine Sammlung von Klassen und insbesondere Schnittstellen, die implementiert und erweitert werden, um das Grundgerüst für Laufzeitun-

K

12.2.2 XP als Lösung

XP hilft dabei, diese Risiken zu minimieren. Dies wird durch die Kombination von verschiedenen Techniken erreicht. So werden zum Beispiel die wichtigsten Funktionalitäten des Programms zuerst implementiert und es existiert zu jedem Zeitpunkt eine lauffähige Version der bis dahin verwirklichten Programmteile (mögen diese auch noch so rudimentätnerud25c),4tnerudois sser

muss, dann geschieht dies erst dann, wenn es notwendig wird. Dies ist besser, als von Vornherein zeitaufwändig an einer komplexen Lösung zu arbeiten, die hinterher überhaupt nicht gebraucht wird.

12.5.9 Programmieren in Paaren (Pair Programming)

Programmiert wird in XP immer zu zweit in einem Paar. Dabei nimmt jeder aus dem Paar eine andere Rolle ein. Ein Partner arbeitet mit Tastatur und Maus und übernimmt die Implementierungsarbeit. Dabei wird er vom anderen Partner unterstützt. Dieser erkennt kleinere Fehler, die immer wieder beim Programmieren auftreten, und denkt darüber hinaus strategischer. Er beschäftigt sich also nicht nur mit der gerade implementierten Methode an sich, son,ei272(anmat)-2J 0 -11.955 TD [(anch,) z.B. Ge
imno fut
gibili-273(13Dumds(Ausdrad.4e(Ged,2193627359Sr4uef7(gabi)293eob)-27367,21ii9f0viduen)-237367antwT1 ETntont-27367icht37367antwT1 ETnt

12.6.3 Tester

In XP sind die Komponententests so stark mit der Programmierung verkoppelt, dass die Pro-

Durch *Paar-Programmierung* können *Erwartungstäuschung* und

Team darauf hinzuweisen.

Projektverantwortlicher

Universität Dortmund geleitet wird.

TEIL 3

Releasebeschreibungen

KAPITEL 13

Systemmetapher

Carina Klar, Antonio Pedicillo

14.3.1 Aufbau der grundlegenden Infrastruktur

Einrichtung der Debug-Pakete
Beschreibung:

Erweiterung der Datenstruktur um Attribute
Beschreibung:

14.4 Vorstellung der Architektur

Daniel Maliga, Jonas Mathis, Michael Striewe

Im Nachfolgenden wird die in diesem Release implementierte Architektur vorgestellt. Dazu wird zunächst die geplante Architektur beschrieben, dann die realisierte Architektur festgehalten und abschließend beide verglichen.

14.4.1 Beschreibung der geplanten Architektur

Für die gestellte Aufgabe bietet sich die Verwendung des Model-View-Control-Konzeptes (MVC) an, das insbesondere bei graphischen Benutzeroberflächen Verwendung findet. Bei diesem Konzept werden die zugrundeliegende Datenstruktur (*Model*) und die Darstellung der Daten (*View*) explizit voneinander getrennt und von spezialisierten Programmteilen übernommen. Dabei sind auch unterschiedliche Darstellungen zur selben Zeit möglich. Zwischen diesen beiden Teilen liegt der *Control*, der Koordinierungsaufgaben sowie die Kernfunktionen

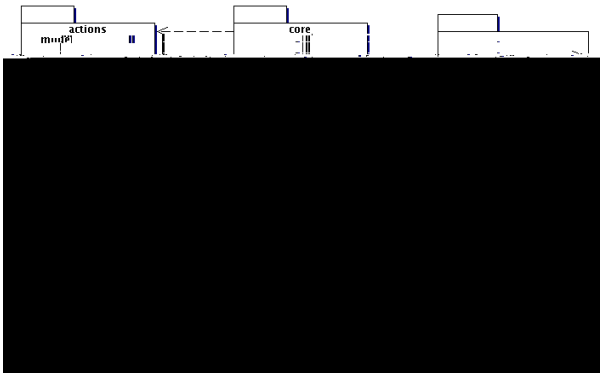


Abbildung 14.1.: Paketstruktur der realisierten Systemarchitektur (Release 1)

14.4.2

14. Vorstellung der Architektur

Abbildung 14.5.:Realisierte Systemarchitektur Release 1

Das Plug-In soll unter Linux ohne Anleitung installiert werden können.
Testergebnis: nachträglich bestanden
Anmerkung: Der Benutzer wird durch geeignete Hinweise in den Fenstern durch die Installation geleitet.

Das Plug-In soll unter Windows ohne Anleitung installiert werden können.
Testergebnis: nachträglich bestanden
Anmerkung: Der Benutzer wird durch geeignete Hinweise in den Fenstern durch die Installation geleitet.

Allgemeiner Programmdurchlauf: Eclipse wird gestartet. – Das zu debuggende Programm wird geöffnet. – Es werden mehrere Breakpoints gesetzt. – In der Auswahl der anzuzeigenden Klassen wird ein ganzes Package ausgewählt. – Der 3D-Debug-Button wird betätigt.
Testergebnis: bestanden
Anmerkung: Dieser Test prüft die Grundfunktionalität der Software und stellt die Ausgangsbasis für die weiteren Tests dar. Ein erfolgloser Test an dieser Stelle würde auch alle folgenden Tests hinfällig werden lassen.

Es soll möglich sein, ganze Hierarchien von Klassen auszuwählen.
Testergebnis: bestanden
Anmerkung:

Die Maus wird über einige Elemente der 3D-Debug-Umgebung geführt. An geeigneten Stellen erscheinen Tooltips.
Testergebnis: bestanden
Anmerkung: Die Tooltips im Fenster zur Auswahl der anzuzeigenden Klassen er-

15.4 Reflexion über die Tasks

Carina Klar, Michael Striewe

Im Folgenden wird über die Tasks des zweiten Release reflektiert. Hierzu werden die einzelnen Tasks kurz vorgestellt und insbesondere auf die signifikanten Probleme und die Differenzen

Darstellung eines Arrays

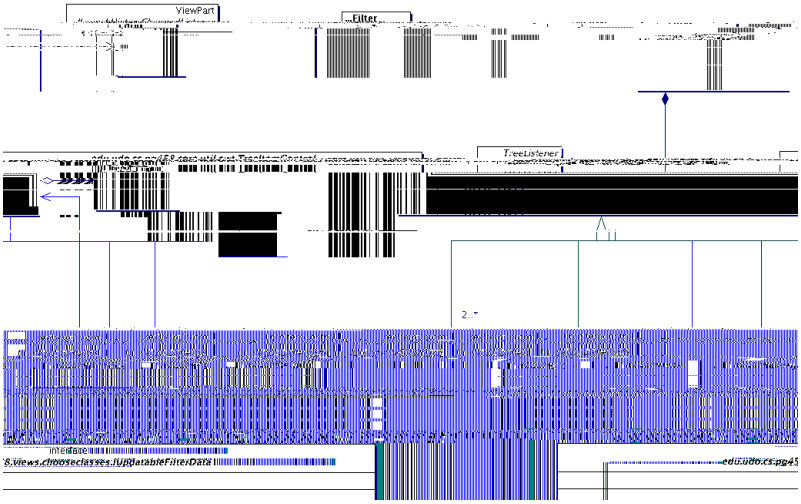
Beschreibung: Es muss eine geeignete Darstellungsweise für Arrays gefunden und implementiert werden.

geplante Zeit: 2 Tage

reale Zeit:

15.5.1 Beschreibung der geplanten Architektur

Den Ausgangspunkt für die Architektur dieses Releases stellt die bestehende Architektur des letzten Releases dar. Sie basiert auf dem Konzept *Model-View-Control*


ViewPart
Filter
TreeListener
2..*
IMetaBase

Hauptsächlich diente das Programm „Dave“ als zu debuggendes Programm. Um spezielle Tests durchführen zu können, erstellten die Kunden aber auch eigene kleinere Programme. Im

Wenn Eclipse in der Java-Perspektive ist, sollte beim Starten des 3D-Debuggers die Perspektive automatisch zur 3D-Debug-Perspektive wechseln oder danach gefragt werden.
Testergebnis: bestanden
Anmerkung: keine

Anhalrsh0bj 9.963.7l2rr7 Programm0bj 9.42(solj 9.42aubj 9.96(der)-.423-Deb)20(ug-P)20(erspekti)10(v)10(e)]TJ T*[(mög

16.2 Reflexion über die Tasks

Antonio Pedicillo, Andrey Lysenko

Im Folgenden wird über die Tasks des dritten Release reflektiert. Hierzu werden die einzelnen Tasks kurz vorgestellt und insbesondere auf die signifikanten Probleme und die Differenzen zwischen Zeitabschätzungen und tatsächlich benötigter Zeit eingegangen.

Die Tasks wurden in vier Gruppen gegliedert, die thematisch als Schwerpunkte aus der Releaseplanung hervor gingen. Die Kategorie „Aufräumarbeiten“ beinhaltet Tasks, die aus einem

lyse von Heap-Dumps der Virtual Machine. Beide Möglichkeiten boten unterschiedliche Vor- und Nachteile in Bezug auf Laufzeit und Informationsgewinn, die im Folgenden näher erläutert werden. Daher wurden beide Ansätze parallel verfolgt. Zusätzlich wurde auch überprüft, inwieweit der bisherige *Recursive Snapshot Collector*

betrachtet und in kleinen Ausschnitten durchaus manuell gelesen sowie durch den von SUN entwickelten Parser HAT (Foote (2004

1. Die Verfolgung aller Aufrufe benötigt enorm viel Zeit verglichen mit der reinen Ausführungszeit. Die Verwendung eines geschickt gewählten Filters könnte bei kleiner Auswahlmenge von überwachten Klassen die Verfolgungszeit in akzeptable Dimensionen bringen.
2. Es scheint keine Möglichkeit zu existieren die Erzeugung von Array-Objekten zu überwachen. Die Snapshots besitzen ohne Arrays keinen hohen Informationsgehalt und können zu Verwirrungen führen.

Diese Nachteile haben uns dazu bewogen, die JDI Methode nicht für den e Aneualctorn-

Die in den ersten beiden Releases eingetzbate Art der Infobeschaffung wurde für dieses Release mit der Bezeichnung „Recursive-ehaltalctorn-“ versehen, da sie r(revsi)25(v)-251(auf)-252(dem)]TJ T* 0(v)20(on)-324(Eclip

16.4.1 Definition des Interfaces

Aus dem letzten Release stammen die Algorithmen „Linlog“ und „Force-directed Layout“.

16.5 Refactoring der Visualisierungen

Daniel Maliga, Daniel Vogtland, Sebastian Vastag

Im vorherigen Release wurden alle grafischen Objekte für jeden Visualisierungsvorgang neu erzeugt, wie auch die Visualizer-Klassen selbst. Die Visualisierungen wurden also vollständig unabhängig voneinander behandelt. Dadurch bedingt wurde in der `Java3DView` bei

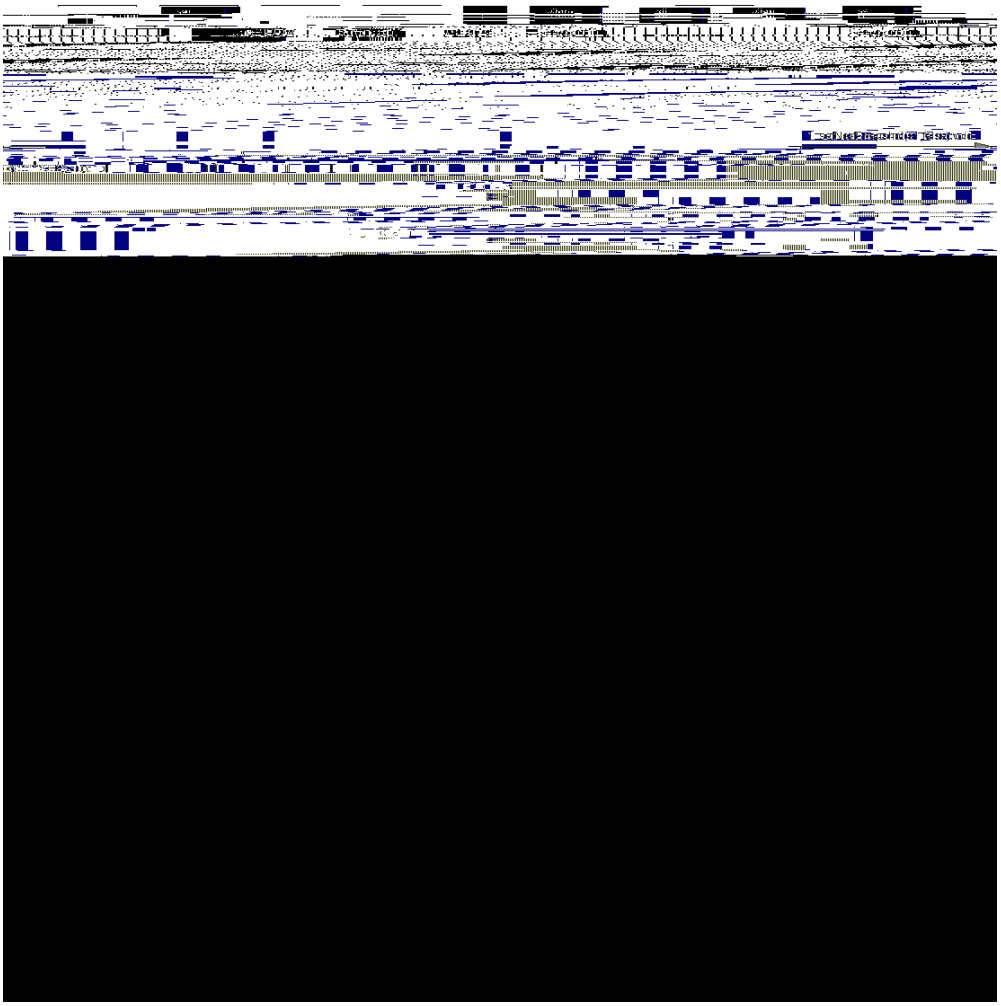


Abbildung 16.12.: Animation eines Snapshot-Übergangs

Die Kugel im Hintergrund ist an die TransformGroup der Szene gebunden. Drehen sich die Objekte der Szene vor dem Bildausschnitt des Betrachters, so dreht sich auch die Sphere mit. Dadurch findet der Benutzer Objekte immer an derselben Stelle des Hintergrundes wieder,

17.2.2 Usability (Benutzerinteraktion)

Export der Animation in einen 2D-Film ermöglichen

Beschreibung: Aus einer Sequenz Snapshot-Bildern im JPEG-Format sollte ein Film generiert werden, welcher dann in einem gängigen Format vorliegt.

geplante Zeit: 3 Tage

reale Zeit:

17.2.5 Übrig gebliebene Arbeiten aus Release 3

Infobeschaffung über JDI implementieren
Beschreibung: Es sollte ein weiterer Kollektor realisiert werden, der sich zur Beschaf-

Einsatz von 3D-Tooltips verbessern/überdenken

Beschreibung:	Die bisher verwendeten 3D-Tooltips hatten den Nachteil, dass ihre Sichtbarkeit vom Abstand der Szene zum Betrachter abhing, d.h. je weiter die Szene herausgezoomt wurde, desto mehr verkleinerten sich die Tooltips bis hin zur Unlesbarkeit.
geplante Zeit:	2 Tage
reale Zeit:	1,5 Tage; Die Idee von 3D-Tooltips wurde fallen gelassen und durch die Nutzung von 2D-Tooltips außerhalb der 3D-Darstellung ersetzt.

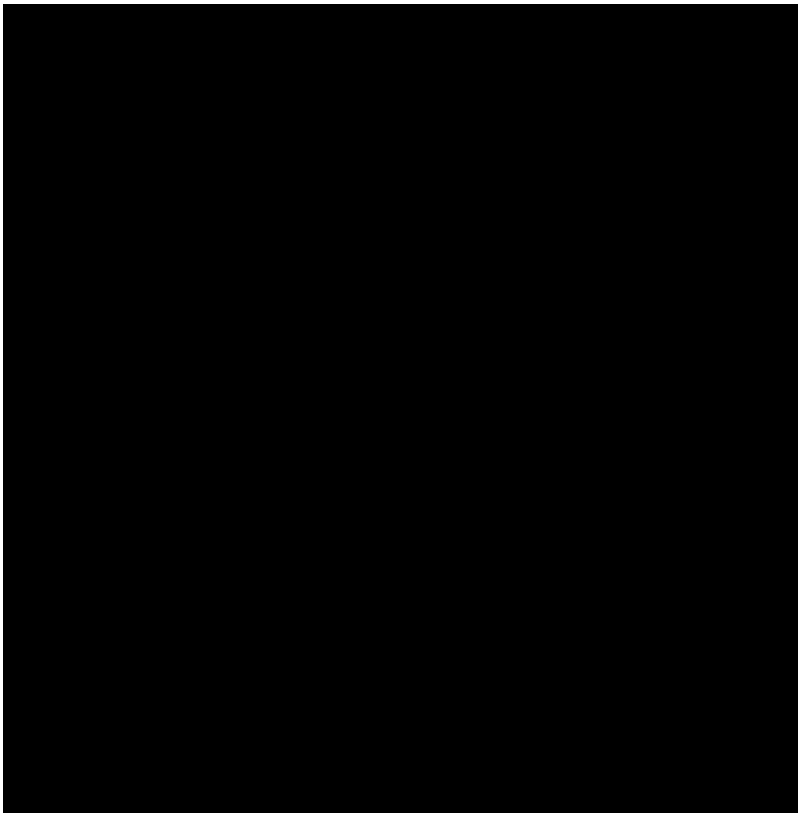


Abbildung 17.1.: Der Snapshotplayer

Zunächst konnte festgestellt werden, dass alle wichtigen Bugfixes, die sich aus der Abnahme von Release 3 ergeben hatten, umgesetzt werden konnten. Danach wurden ausführlich die

K

Alternativ kann der Benutzer auch automatisch Snapshots nach einem festen Zeitintervall generieren lassen. So kann man das Verhalten des Systems über einen längeren Zeitraum hinweg analysieren. Dies wird durch eine animierte Visualisierung erleichtert. Dabei ist durch die Navigationsmöglichkeiten der *3D Debug View* eine freie Wahl der Perspektive möglich. Durch die Option, Snapshots speichern zu können, kann die Betrachtung auch zu späteren Zeitpunkten ohne erneute Sammlung der Debuginformationen und Berechnung der Visualierungsdaten erfolgen. Soll das Systemverhalten in einem Vortrag demonstriert werden (z.B. während der Einarbeitung eines neuen Teams), kann eine Animation auch in einen 2D-Film exportiert und so leicht in die Präsentation eingearbeitet werden.

18.2 Programmmodell

Auf dieser Ebene werden kurze Quelltextabschnitte analysiert. Der Benutzer entwickelt dabei genaue Vorstellungen über den (korrekten) Ablauf von Algorithmen, den Datenfluss im betrachteten Programm und die verwendeten Datenstrukturen. Dies alles wird unter dem Be-

Ist eine Animation zu komplex, um sie sofort vollständig zu erfassen, kann die Repeat-Funktion des *Snapshot Players* von Nutzen sein. So kann der betrachtete Ablauf (mit einer für den Be-

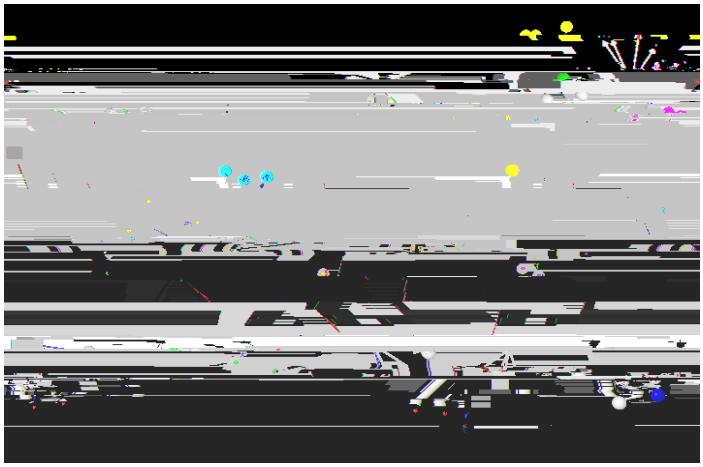


Abbildung 18.2.: Das Auto-Beispiel: Teilstrukturen sind zusammenhängend

Abbildung 18.3.: Entwicklung des Auto-Beispiels: Wo kommen die freihängenden Räder samt Array her?

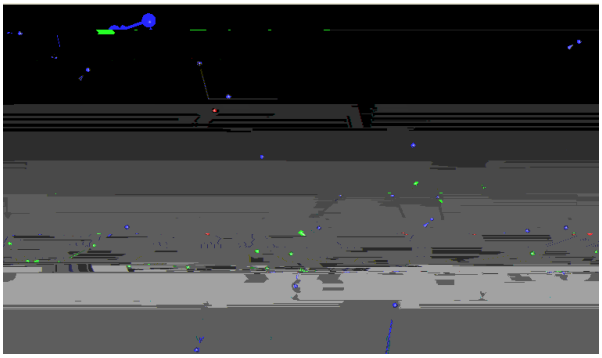


Abbildung 18.7.: Fehlerhafte Verkettung

Abbildung 18.8.:

KAPITEL 19

Grenzen des Plug-Ins

Boris Brodski, Boris Düdder

Dieser Abschnitt behandelt die Grenzen des realisierten Plug-Ins. Diese Grenzen sind die Folge von technischen als auch physiologischen Beschränkungen.

19.1 Kriterien

Es existieren veb-27teri4.4003i.eri4i0 A.isogis1(mationsbh)5(af)25(fisc0)u5oglisc0ogimVPltzisB1dogeb-27t-*3 T0(og)-.iszischhl.is

Um diesem Missstand Abhilfe zu schaffen, unterstützt 3Debug zusätzlich ein proprietäres binäres Speicherformat.

19.3.4

	Snapshotcollector	Snapshotcollector (optimiert)	NetDumper	

19.4.4 JDI Collector

Der JDI Collector basiert auf einem Bestandteil der *Java Debug Architecture* (JDA), dem sogenannten *Java Debug Interface* (JDI). Dieses Interface existiert als Referenzimplementierung von SUN und wird auch von anderen entsprechenden Engines (IBM Blackdown, etc.) zur Verfügung gestellt. Diese Technologie verwendet ein verbindungsorientiertes Kommunikationsprotokoll, JDWP, um mit dem Remote-Debuggee zu interagieren.

te Auswirkungen auf das Ergebnis der Anordnung, wie die Abbildungen 20.1 und 20.2 zeigen. Abbildung 20.1 wurde mit folgenden Parametern erstellt: 100 Iterationen, Gravitation 0.0 sowie Anziehungskräfte 0.25, 0.15, 0.25 und 0.1. Abbildung 20.2 wurde mit folgenden

KAPITEL 22

Erfahrungen mit Java3D

Daniel Vogtland, Henning Zeller

Wir haben uns zu Anfang unseres Projektes dazu entschlossen, dreidimensionale Visualisierungen durch Java3D zu realisieren. Im Folgenden wollen wir diese Entscheidung kurz rückblickend bewerten. Da bereits in Kapitel 8 eine Einführung in die Konzepte von Java3D erfolgte, werden wir hier auf diese nicht mehr detailliert eingehen. Grafische Transformationen wurden in Kapitel 7 erläutert.

22.1 Vorteile von Java3D

Java3D wird kostenlos von SUN zur Verfügung gestellt und bietet eine auf OpenGL oder DirectX aufsetzende Java API zur dreidimensionalen grafischen Ausgabe. Erweiterte Audiomöglichkeiten sind ebenfalls Bestandteil von Java3D, dies war jedoch für unser Projekt nicht relevant.

Java3D ist vollständig objektorientiert. Dies macht das Zusammenspiel mit anderem Java-Code problemlos möglich und ersparte uns die Einarbeitung in eine spezielle Grafikprogrammiersprache und den Wechsel zwischen verschiedenen Programmierkonzepten.

K

ren. Auch für Projekte, die ansonsten nicht XP nutzen, ist es sicher hilfreich, stets ein anschauliches Ziel vor Augen zu haben. Ebenfalls für empfehlenswert halten wir die Unterteilung des Projekts in mehrere kleinere Releases sowie die weitere Aufteilung dieser Releases durch das Planungsspiel. Das Formulieren von Teilaufgaben passender Größe erfordert allerdings bereits ein tiefgehendes Problemverständnis und eine gute Selbsteinschätzung der eigenen Leistungsfähigkes

ANHANG A

Handbuch, Quelltexte und Lizenz

Daniel Maliga

Das Projekt 3DEBUG wird als Open-Source-Projekt unter der

Literaturverzeichnis

[Beck 2000] BECK, Kent: *eXtreme Programming explained*. Addison Wesley, Reading, 2000

LITERATURVERZEICHNIS