TeeJay - ein Werkzeug zur interaktiven Definition und Ausführung funktionsorientierter Tests auf Java-Objekten

Ralph Weires, Rainer Oechsle
`weiresr@fh-trier.de, oechsle@informatik.fh-trier.de`

Zusammenfassung

In diesem Artikel wird das Testwerkzeug *TeeJay* kurz vorgestellt, das im Rahmen einer Diplomarbeit des Studiengangs Angewandte Informatik an der Fachhochschule Trier entstanden ist. Es dient zum funktionsorientierten Testen von Java-Objekten, wobei insbesondere auch eine Unterstützung für über RMI nutzbare Objekte gegeben ist.

Über eine grafische Benutzeroberfläche können verschiedene elementare Operationen (z.B. Methodenaufrufe) durchgeführt werden, um die Testumgebung zu manipulieren und auf Konsistenz zu überprüfen. Durch eine Aufzeichnungsfunktion kann eine Reihe derartiger Aktionen protokolliert werden, um daraus einen Testfall zu erstellen. Dieser kann im weiteren Verlauf zur erneuten Ausführung verwendet werden.

Schlüsselwörter: Interaktive Test-Definition, funktionsorientierter Test, Testen von Java-Objekten, Testen von RMI-Objekten

1 Einleitung

Das Testen von Programmen stellt eine zentrale und in der Praxis viel genutzte Aufgabe im Bereich der Software-Qualitätssicherung dar. Um Tests jedoch einfach und effizient durchführen zu können, sind geeignete Werkzeuge in diesem Bereich von Nutzen.

TeeJay ist ein in Java geschriebenes Werkzeug, mit dem durch eine grafische Benutzeroberfläche Tests für andere Programme der selben Programmiersprache interaktiv erstellt und ausgeführt werden können, ohne dass der Benutzer dazu eigenen Quellcode schreiben muss. Die Tests bestehen dabei aus Sequenzen einzelner einfacher Operationen, die z.B. Konstruktor- oder Methodenaufrufe darstellen. Um diese Funktionalität zu realisieren, macht TeeJay intensiven Gebrauch von der *Java Reflection API*.

Auf diese Weise lässt sich eine Testumgebung, die im Wesentlichen aus Klassen und Objekten besteht, interaktiv manipulieren, um so Testfälle zu definieren. Damit währenddessen geprüft werden kann, ob sich die Testumgebung aktuell im gewünschten, spezifizierten Zustand befindet, können zwischendurch Bedingungen an die vorhandenen Objekte gestellt werden (z.B. durch einen gewünschten Rückgabewert einer Methode).

TeeJay erfordert zur Ausführung eine installierte Java-Laufzeitumgebung (JRE) der Version 1.4 oder höher und kann unter den Bedingungen der GNU General Public License (GPL) frei genutzt und kopiert werden. Die gesamte zugrunde liegende Diplomarbeit [1] sowie die Software sind auf den Internetseiten des Autors [3] verfügbar.

2 Funktionalität

TeeJay bietet zwei generelle Arbeitsbereiche, wovon einer zur Definition von Tests und der andere zur Ausführung bestehender Tests genutzt wird. Auf eine Darstellung dieser Ansichten wird hier aus Platzgründen verzichtet; stattdessen werden einzelne Komponenten der Oberfläche beispielhaft gezeigt und erläutert, um einen groben Überblick über das Programm zu vermitteln.

2.1 Definitionsansicht

Die Definitionsansicht stellt u.A. die aktuelle Testumgebung dar, die aus Klassen, entfernten Diensten, Objekten und primitiven Variablen besteht. Entfernte Dienste werden dabei z.B. durch eine *RMI Registry* repräsentiert und dienen zum Beschaffen fernaufrufbarer RMI-Objekte, die anschließend im Wesentlichen wie lokale Objekte genutzt werden können. Allerdings werden Operationen auf RMI-Objekten automatisch mit einem Timeout versehen, um sie nach einer gewissen (vom Benutzer definierten) Zeit abzubrechen, womit möglichen Störungen bei der Verbindung zum entfernten Dienst Rechnung getragen wird.

Im Bereich der Klassen und Objekte werden z.B die jeweils verfügbaren Elemente (Members) angezeigt, die für die Durchführung von Operationen verwendet werden können. Anhand der Klassenansicht wird im Folgenden grob erläutert, auf welche Weise grundlegende Operationen in TeeJay durchgeführt werden. Die Abbildung 1 zeigt dazu beispielhaft eine Ansicht vorhandener Klassen in der Testumgebung. Diese werden nach den jeweiligen Paketen sortiert in einer Baumstruktur dargestellt. Wie am Beispiel der Klasse `Boolean` zu sehen ist, enthalten die Klassen ihre Konstruktoren sowie die statischen Attribute und Methoden. Allerdings kann in TeeJay nur auf öffentliche Elemente (Modifizierer `public`) zugegriffen werden - eine Umgehung der definierten Kapselung einer Klasse ist also nicht möglich (auch wenn sich dies in einer zukünftigen Version durchaus ändern kann). Die verfügbaren Elemente werden durch ihre jeweili-

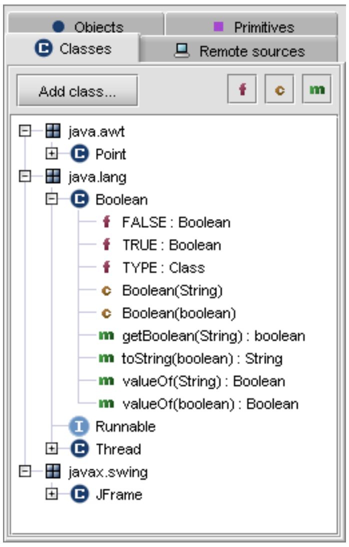

Abbildung 1: Klassenansicht der Testumgebung

ge Signatur visualisiert und können zur Ausführung einer Operation verwendet werden. So wäre es in der Abbildung z.B. möglich, die statische Methode `getBoolean(String):boolean` aufzurufen.

Um einen Methodenaufruf durchzuführen, müssen die jeweils erforderlichen Parameter spezifiziert werden. Hierzu können vorhandene Objekte und primitive Variablen aus der Testumgebung verwendet werden, die während der vorherigen Arbeit bereits beschafft wurden (diese sind grundsätzlich unter einem definierten Bezeichner ansprechbar). Neben Argumenten aus der Testumgebung können bei primitiven Parametern auch feste Werte angegeben werden; bei Objektparametern kann alternativ der Wert `null` oder (falls der Parametertyp dazu kompatibel ist) auch ein literaler String Verwendung finden. Sofern die Methode einen Rückgabewert besitzt, besteht zudem die Möglichkeit, diesen für eine spätere Nutzung in der Testumgebung unter einem definierten Bezeichner abzulegen.

Alternativ zum Ablegen in der Testumgebung kann der Rückgabewert einer Methode auch auf eine zu spezifizierende Bedingung getestet werden, um einen gewünschten Zustand zu überprüfen. Diese Möglichkeit ist hauptsächlich für primitive Datentypen gegeben, die mit Hilfe üblicher Vergleichsoperatoren wie `==` oder `<=` und der Angabe eines typkompatiblen Referenzwertes (z.B. wieder aus der Testumgebung) geprüft werden können.

Nach der vollständigen Spezifikation einer derartigen Operation wird sie anschließend mit Hilfe der Java Reflection API ausgeführt. Auf diese Weise lässt sich die Testumgebung Stück für Stück interaktiv zusammenstellen, wobei die Konsistenz zwischendurch durch geeignete Bedingungen geprüft werden kann. Das Ergebnis einer Operation wird dabei in einem Protokollfenster angezeigt. Sollte es bei der Ausführung zu einem Fehler in Form einer geworfenen Ausnahme kommen, wird diese zusammen mit dem zugehörigen *Stack trace* ebenfalls im Protokoll angezeigt, um so die Fehlersuche zu erleichtern.

Zu Beginn einer neuen Sitzung in TeeJay werden typischerweise die anfänglich benötigten Klassen (durch die Angabe des jeweiligen voll qualifizierten Klassennamens) in die Testumgebung importiert. Dadurch kann die Klassenansicht mit Inhalt gefüllt werden, wie in Abbildung 1 für mehrere Klassen bzw. Schnittstellen zu sehen ist. Durch die Attribute, Methoden und Konstruktoren dieser Klassen können dann nach und nach Objekte bzw. primitive Variablen beschafft werden, auf bzw. mit denen wiederum weitere Operationen durchgeführt werden. Somit können aufeinander aufbauende Sequenzen einzelner Operationen erstellt werden.

Damit TeeJay auf die jeweils benötigten Klassen zugreifen kann, lassen sich in den Benutzereinstellungen Verzeichnisse und Bibliotheken (in Form von jar- und zip-Dateien) angeben, unter denen die Klassen zu finden sind. Dabei sind lediglich die kompilierten class-Dateien erforderlich; der Quellcode wird also nicht benötigt. Für RMI-Objekte kann auf Wunsch auch dynamisches Klassenladen verwendet werden, wodurch die benötigten Klassen nicht zwingend lokal vorhanden sein müssen. Um von dieser Möglichkeit Gebrauch zu machen, muss TeeJay allerdings mit besonderen Optionen gestartet werden.

Die Objektansicht, die in Abbildung 2 beispielhaft gezeigt wird, kann auf ähnliche Art wie die Klassenansicht genutzt werden. Hier werden die

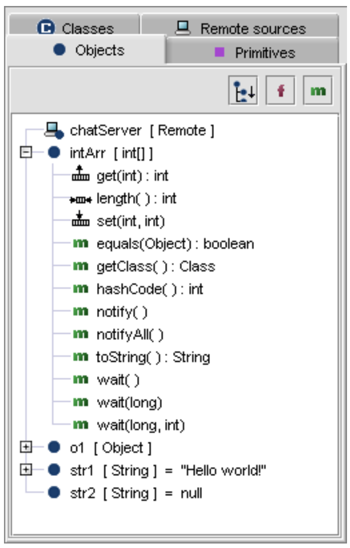

Abbildung 2: Objektansicht mit Pseudo-Arraymethoden

(öffentlichen) Instanzattribute und -methoden der vorhandenen Objekte zur Verfügung gestellt. Durch Filter lassen sich dabei z.B. geerbte Attribute und

Methoden ein- bzw. ausblenden. Jede Referenz besitzt hier einen Typ, der sich ähnlich zu einer Referenz im Java-Code verhält.

Der initiale Typ der Referenz eines Objektes, das durch einen Methodenaufruf gewonnen wird, richtet sich daher nach dem formalen Typ des Rückgabewertes der entsprechenden Methode. Die angezeigten und damit verfügbaren Attribute und Methoden hängen ebenfalls vom aktuellen Referenztyp ab, der vom Benutzer jedoch durch eine spezielle Operation zur expliziten Typumwandlung (Cast) verändert werden kann.

Falls die Referenz eines Objektes ein Feld (Array) darstellt, werden zudem Pseudo-Methoden zur Verfügung gestellt, die es in dieser Form in Java eigentlich nicht gibt. Durch diese Methoden werden die üblichen Aktionen auf Feldern (lesen, schreiben, Länge abfragen) derart realisiert, dass sie für den Benutzer ebenso wie die 'normalen' Methoden genutzt werden können. Ein Beispiel für diese Methoden, deren Signatur z.T. vom Typ des Feldes abhängt, wird in Abbildung 2 dargestellt.

Um nun mehrere einzelne Operationen, von denen einige beispielhaft beschrieben wurden, zu einem Test zusammenzufassen, bietet TeeJay eine Art Capture/Replay-Funktion, mit der Tests definiert und ausgeführt werden können. Zur Definition eines Tests kann das Programm in einen Aufzeichnungsmodus versetzt werden, in dem alle Operationen protokolliert werden, die der Nutzer ab diesem Zeitpunkt in der Testumgebung ausführt (z.B. Importieren von Klassen, Aufruf von Methoden etc.). Beim Beenden einer solchen Aufnahme steht diese Folge der durchgeführten Operationen als eine Testeinheit zur Verfügung.

Neben solchen aufgezeichneten Tests können in der Definitionsansicht auch Test-Suiten definiert werden, die zum Zusammenfassen anderer Tests (ggf. auch weiteren Suiten) genutzt werden. Auf diese Weise ist es z.B. möglich, aufeinander aufbauende Testeinheiten in einer Test-Suite zusammenzufassen, damit diese später als Ganzes ausgeführt werden können. Zudem können die enthaltenen Tests in einer Suite mit einem Wiederholungsfaktor versehen werden, damit ggf. eine mehrmalige Ausführung erfolgt.

Definierte Tests können zudem persistent gespeichert werden, um sie in späteren Sitzungen erneut zu verwenden. Um einen vorhandenen Test auszuführen, existiert in TeeJay eine eigene Ansicht (im Gegensatz zur Definitionsansicht), die im Folgenden kurz beschrieben wird.

2.2 Testlauf-Ansicht

Im zweiten Bereich der grafischen Oberfläche von TeeJay kann der Benutzer einen Test aus der Menge aller vorhandenen Tests auswählen, um diesen wiederholt ausführen zu lassen. Dadurch können einmal definierte Tests in späteren Sitzungen, z.B. nachdem sich der zu testende Quellcode geändert hat, erneut durchgeführt werden. Auf diese Weise kann das Programm für Regressionstests verwendet werden - also um zu prüfen, ob sich durch die erfolgten Änderungen keine neuen Fehler (zumindest bezüglich der definierten Tests) in das Programm eingeschlichen haben.

Der Verlauf eines hier gestarteten Tests kann während der Ausführung beobachtet werden, wobei die Geschwindigkeit einstellbar ist. Die Ergebnisse werden auf zwei Arten dargestellt:

1. In der detaillierten Anzeige wird das Ergebnis jeder einzelnen Operation angezeigt, wobei im Fehlerfall der jeweilige Grund dargestellt wird (z.B. der Typ einer geworfenen Ausnahme inklusive zugehörigem *Stack trace*).

2. In der Zusammenfassung der bisherigen Ergebnisse des gesamten Testlaufs wird eine Statistik angezeigt, um einen groben Überblick über den Verlauf zu erhalten. Abbildung 3 zeigt ein Beispiel für diese Zusammenfassung nach der Beendigung eines Testlaufs. Die Anzeige

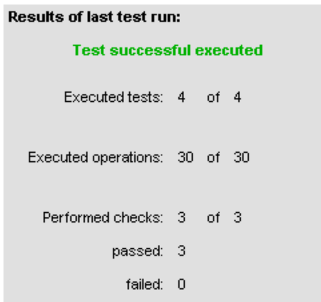

Abbildung 3: Zusammenfassung eines Testlaufs

beinhaltet Informationen über die Anzahl durchgeführter Tests, ausgeführter Operationen und Prüfungen (wobei die Prüfungen ebenfalls Operationen sind). Bei den Prüfungen wird zudem angezeigt, wie viele davon erfolgreich bzw. fehlgeschlagen sind. Nach der Beendigung eines Testlaufes wird außerdem eine zusammenfassende Meldung angezeigt, die Aufschluss über Erfolg oder Misserfolg gibt. Die in der Abbildung gezeigte Zusammenfassung (*Test successful executed*) erscheint z.B. nur, falls der Testlauf insgesamt erfolgreich beendet wurde (kein Abbruch durch den Benutzer oder eine fehlgeschlagene Operation) und zudem alle getesteten Bedingungen erfüllt waren.

3 Vergleich

Es existieren einige wenige dem Autor bekannte Werkzeuge, die eine z.T. ähnliche Funktionalität wie TeeJay besitzen. An dieser Stelle werden daher Unterschiede und die sich daraus ergebenden Vor- bzw. Nachteile dieser Programme im Vergleich betrachtet.

3.1 JUnit

JUnit [4] ist sicherlich eines der am meisten verbreiteten Werkzeuge zum Testen von Java-Programmen, das wie TeeJay zur Erstellung funktionsorientierter Tests genutzt wird. Es ist einfach strukturiert und vielseitig einsetzbar, da es die Definition der Tests direkt durch Quellcode erfordert. Dies schränkt die Möglichkeiten zur Definition von Tests nicht ein, während in TeeJay im Gegensatz dazu z.B. keine Kontrollstrukturen innerhalb der Tests genutzt werden können.

Der wichtigste grundlegende Unterschied (abgesehen von der grafischen Oberfläche) zwischen den beiden Programmen besteht darin, dass die Definition von Tests bei JUnit *statisch* erfolgt, während sie bei TeeJay *dynamisch* durchgeführt wird. Dynamisch bedeutet in diesem Zusammenhang, dass die definierten Operationen bereits unmittelbar nach ihrer Spezifikation ausgeführt werden.

Daher können die Ergebnisse und Effekte der einzelnen Operationen in TeeJay bereits zur Definitionszeit beobachtet werden, wodurch TeeJay das Capture/Replay-Verhalten für die durchgeführten Aktionen realisieren kann. Auf diese Weise ist auch eine ständig aktuelle Übersicht der Testumgebung (also der vorhandenen Objekte, Klassen etc.) verfügbar, weshalb bereits während der Erstellung eines Tests eine Analyse des Verlaufs möglich ist. Auftretende Fehler können damit sofort bemerkt werden.

Des weiteren besteht ein Unterschied darin, dass die einzelnen Testfälle in JUnit unabhängig voneinander sind und demnach problemlos in verschiedener Reihenfolge ausgeführt werden können. Dadurch stellt jeder Testfall also einen abgeschlossenen kleinen Test dar.

Im Gegensatz dazu können die Tests in TeeJay durchaus Abhängigkeiten besitzen. Wird hier in einem Test z.B. ein Objekt zur Testumgebung hinzugefügt, bleibt dieses Objekt auch nach der Beendigung des Tests für die folgenden Operationen verfügbar (sofern es nicht explizit wieder entfernt wird). Auf diese Weise können aufeinander aufbauende Tests erstellt werden, wodurch eine gewisse Modularisierung genutzt werden kann. So kann z.B. ein Test mit einer Menge von Vorbereitungsschritten erstellt werden, der später wahlweise von mehreren anderen Tests genutzt wird.

Diese Vorgehensweise in TeeJay besitzt jedoch den Nachteil, dass (gerade aufgrund der Möglichkeit aufeinander aufbauender Tests) bei einer gemeinsamen Ausführung mehrerer Testfälle (in einer Test-Suite) der Abbruch eines Testfalls zum Abbruch des gesamten Testlaufes führen muss. In JUnit ist ein solcher Abbruch aufgrund der Unabhängigkeit der Tests dagegen nicht nötig; bei einem Fehlschlag in einem Testfall können die restlichen dennoch problemlos ausgeführt werden.

3.2 Exacum

Das kommerzielle Testwerkzeug Exacum [5] wurde erst gegen Ende der Entwicklung von TeeJay entdeckt und hat daher keinen Einfluss auf dieses Programm gehabt. Dennoch gibt es interessanterweise viele Ähnlichkeiten zwischen Exacum und TeeJay. Allerdings existieren auch einige Unterschiede, die hier dargestellt werden.

Exacum ist nicht nur mit TeeJay, sondern auch mit JUnit vergleichbar. Im Gegensatz zu JUnit erfolgt die Arbeit mit Exacum jedoch über eine grafische Oberfläche. Wie in TeeJay können in Exacum z.B. Konstruktor- und Methodenaufrufe sowie Bedingungen definiert werden, aus denen sich ein Testfall zusammensetzt. Diese Definition erfolgt in Exacum allerdings *statisch*. Die Ausführung kommt demnach auch hier erst durch einen expliziten Testlauf der definierten Testfälle zustande; während der Definition eines Tests werden also noch keine tatsächlichen Testschritte durchgeführt.

In dieser Hinsicht verhält sich Exacum also wie JUnit, weshalb die dazu im letzten Abschnitt genannten Vor- und Nachteile im Vergleich mit TeeJay auch hier gelten. Auch die Unabhängigkeit der einzelnen Testfälle voneinander ist in Exacum wie in JUnit vorhanden, weshalb hier ebenfalls die bereits genannten Folgen diesbezüglich zum Tragen kommen.

Insgesamt ist Exacum stark mit JUnit vergleichbar, wenn von der grafischen Oberfläche abgesehen wird. Im Vergleich hat JUnit hier allerdings wieder den Vorteil, dass durch die direkte Definition des Quellcodes alle Möglichkeiten der Programmierung offen stehen.

Ansonsten ist Exacum in einigen Bereichen recht ähnlich zu TeeJay, da in beiden Programmen eine interaktive Definition der einzelnen Testschritte über eine grafische Oberfläche möglich ist und die so erstellten Testfälle über die Oberfläche ausgeführt werden. Neben den genannten Unterschieden bestehen jedoch auch noch eine Reihe weiterer Differenzen. Im Bereich verteilter Anwendungen z.B. unterstützt TeeJay Techniken auf RMI-Basis, während Exacum hier dagegen den Schwerpunkt auf EJBs und Servlets legt.

3.3 BlueJ

BlueJ [6] stellt kein Testwerkzeug, sondern eine Entwicklungsumgebung für Java dar, die insbesondere Programmieranfängern Konzepte objektorientierter Programmierung näher bringen will. Sie diente TeeJay als Vorbild in der Hinsicht, dass hier ebenfalls interaktive Aktionen mit Klassen und Objekten durchgeführt werden können. Auch in BlueJ werden vorhandene Klassen und Objekte in einer grafischen Oberfläche angezeigt, durch die zudem z.B. ein interaktives Aufrufen vorhandener Methoden ermöglicht wird. Diese Eigenschaft wird in TeeJay auf ähnliche Weise realisiert.

3.4 JavaCHIME

Wie BlueJ ist das Werkzeug JavaCHIME ebenfalls nicht auf das Testen von Anwendungen ausgerichtet, sondern dient ähnlich wie BlueJ im Wesentlichen zu Lehrzwecken. Es bietet ähnlich wie TeeJay Möglichkeiten, um über eine interaktiv Oberfläche mit Klassen und Objekten zu interagieren. Des weiteren können z.B. die Attribute der vorhandenen Objekte genauer inspiziert werden. Über dieses Projekt ist neben einer kurzen Abhandlung [7] jedoch bisher keine weitere Information zu finden, da es sich noch in der Entwicklung befindet. Daher kann ein eingehender Vergleich mit TeeJay hierfür nicht erfolgen.

4 Ausblick

Bislang unterstützt TeeJay lediglich eine Test-Definition durch die explizite Angabe aller Einzelschritte, die sich i.W. auf Quellcode-Ebene befinden. Ein solches Vorgehen wird jedoch schnell sehr aufwändig; zudem ist das verbreitete Testwerkzeug JUnit in diesem Bereich deutlich effizienter und universeller einsetzbar.

Die Funktionalität von TeeJay kann in dieser Hinsicht jedoch erheblich erweitert werden, um es zu einem vielseitigeren Werkzeug zu machen. So können z.B. Möglichkeiten implementiert werden, um die Anforderungen an ein Programm durch eine abstraktere Spezifikation zu definieren. Eine solche Spezifikation könnte von TeeJay dann genutzt werden, um daraus eigenständig passende Testfälle abzuleiten, welche die definierten Anforderungen geeignet prüfen.

Geplant ist in dieser Hinsicht insbesondere eine Option zur grafischen Definition eines Zustandsautomaten in UML-Notation, der z.B. das Verhalten einer einzelnen Klasse beschreibt. Hierdurch kann sich ein Tester eher auf das Verhalten des zu prüfenden Programm(teil)s konzentrieren als auf die detaillierte Definition der Schritte einzelner erforderlicher Testfälle. Aus dem Automaten können unter Zuhilfenahme geeigneter Algorithmen anschließend automatisch Testfälle bestimmt werden, die in ihrer Gesamtheit je nach Anforderung bspw. eine größtmögliche Zustands- oder Kantenüberdeckung erreichen.

Ein derartiger Ansatz wird auch in der Diplomarbeit von D. Sokenou [2] beschrieben, die sich speziell mit der Definition von Zustandsautomaten für den Klassentest befasst. Insbesondere geht diese Arbeit darüber hinaus auf eine Technik zur Vererbung von Zustandsautomaten in Klassenhierarchien ein. Dadurch kann ggf. zusätzlich Aufwand zur Erstellung der Spezifikation eingespart werden, was den Test insgesamt effektiver gestaltet.

Neben der Definition von Automaten können in TeeJay jedoch auch andere Techniken zur Spezifikation eines Programms realisiert werden (z.B. Sequenzdiagramme).

Neben solchen grundlegenden Erweiterungen sind auch einige kleinere Verbesserungen geplant, die sich z.B. auf einen höheren Komfort bei der Bedienung des Programms beziehen. Des weiteren wäre eine erweiterte Darstellung verschiedener Inhalte nützlich, um dem Nutzer einen noch genaueren Überblick über die Testumgebung zu vermitteln. Hier könnte auch die Fähigkeit der Java Reflection API, auch auf Elemente mit eingeschränkter Sichtbarkeit zuzugreifen, zum Tragen kommen, um z.B. zwecks Analyse von Objekten eine detaillierte Inspektion aller Attribute eines Objektes zu ermöglichen.

Zudem könnte die bisher nicht vorhandene Möglichkeit geschaffen werden, bereits definierte (aufgezeichnete) Tests auch im Nachhinein zu editieren. Auf diese Weise führt ein kleiner Fehler bei der Definition nicht unbedingt dazu, dass die gesamte Aufzeichnung wiederholt werden muss.

Des weiteren ist die Möglichkeit zum Zugriff auf entfernte Dienste aktuell nur für RMI Registries implementiert, was z.B. um Dienste in Form von *JINI Lookup Services* oder *Java Spaces* erweitert werden soll.

Literatur

[1] R. Weires (2003): *Entwurf und Implementierung eines Werkzeugs zur interaktiven Definition und Ausführung funktionsorientierter Tests für lokale und über RMI nutzbare Java-Objekte*, Diplomarbeit, Fachhochschule Trier

[2] D. Sokenou (1998): *Entwicklung eines Werkzeugs zur Unterstützung zustandsbasierter Testverfahren für JAVA-Klassen*, Diplomarbeit, Technische Universität Berlin

[3] TeeJay, `http://www.ralph-weires.de/stud-da.php`

[4] JUnit, `http://www.junit.org/`

[5] Exacum, `http://www.ist-dresden.de/products/Exacum/`

[6] M. Kolling: *BlueJ-The Interactive Java Environment*, `http://www.bluej.org/`

[7] P. Tadepalli, H.C. Cunningham (2003): *JavaCHIME: Java Class Hierarchy Inspector and Method Executer*, University of Mississippi `http://www.cs.olemiss.edu/~hcc/papers/JavaCHIMEsubmit.pdf`