Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik
Lehr- und Forschungsgebiet Softwaretechnik

Diplomarbeit

Automatische Generierung von Testskriptkommandos durch Capturing von Nutzereingaben in GUI-Programmen

eingereicht von: Andreas Hirth

Betreuer: Prof. Dr. Klaus Bothe

Berlin, den 11. September 2005

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 11. September 2005

Einverständniserklärung

Ich erkläre hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den 11. September 2005

Inhaltsverzeichnis

1. Einführung

Bei der Neuerstellung, aber auch bei der Pflege und Erweiterung von Software-Systemen wird der Software-Entwickler bei wachsendem Umfang der Projekte schnell mit steigender Komplexität konfrontiert. In diesem Fall, oder wenn an einem Software-Projekt eine ganze Entwicklergruppe arbeitet, kann der Einzelne das System als Ganzes nicht mehr im Detail überblicken. So kann es passieren, dass Seiteneffekte von Änderungen oder neuen Entwicklungen nicht rechtzeitig erkannt werden und später an unerwarteten Stellen Fehler verursachen, die nur sehr schwer ihrer Ursache zugeordnet werden können. Deshalb ist es wichtig, den gesamten Entwicklungsprozess durch ständiges Testen zu begleiten. Dabei spielt besonders der Funktionstest, d.h. das Testen des Software-Systems gegen seine Spezifikation, eine wichtige Rolle. Allgemein bedeutet aber die manuelle Durchführung von Software-Tests einen erheblichen Arbeitsaufwand, der zu Lasten der regelmäßigen Durchführung geht.

Hier setzt die am Lehrstuhl Softwaretechnik des Instituts für Informatik der Humboldt-Universität zu Berlin entwickelte Testsuite ATOS an. Das Programm erlaubt es, Funktionstests automatisiert ablaufen zu lassen. Dabei werden die Eingaben eines virtuellen Testers simuliert und die Reaktionen des Testobjekts beobachtet und analysiert. Die Erstellung und Pflege der verschiedenen Testszenarien musste bisher aber manuell stattfinden, was bei komplexeren Benutzungsoberflächen der Testobjekte einen erheblichen Aufwand erforderte.

Ziel dieser Diplomarbeit war es, eine Möglichkeit zu entwickeln, in der Testsuite ATOS neben der automatischen Testdurchführung auch die Erstellung und Pflege von Testszenarien weitgehend zu automatisieren. Gelungen ist dies durch die Erweiterung von ATOS um einen so genannten Capturing-Mechanismus: Während der Tester das Testobjekt bedient und den Testfall abarbeitet, schneidet ATOS die Eingaben mit und erzeugt automatisch die entsprechend sinnvollen Testkommandos. Außerdem kann der Tester zu jedem Zeitpunkt Befehle zur Überprüfung des Zustands des Testobjekts generieren lassen. Die Erstellung und Pflege von Testszenarien gestaltet sich durch diese Erweiterung wesentlich direkter und kostet deutlich weniger Zeit, was die Motivation zu umfassenden Regressionstests steigert und auch die Bereitschaft, die Testfälle immer auf aktuellem Stand zu halten.

2. Testen von Applikationen mit grafischen Benutzungsoberflächen (GUI)

Im Folgenden soll darauf eingegangen werden, was Programme mit grafischen Benutzungsoberflächen auszeichnet und welchen Einfluss dieses speziell für die Definition und Durchführung von Funktionstests hat.

2.1. Besonderheiten bei GUI- im Vergleich zu textbasierten Anwendungen

Textbasierte Programme sind im Allgemeinen dadurch gekennzeichnet, dass zu jeder Zeit ihrer Abarbeitung nur eine überschaubare und wohl definierte Anzahl möglicher Eingaben existiert. Dies wird dadurch gefördert, dass aufgrund der Einschränkung auf Textausgaben der Umfang gleichzeitig darstellbarer Informationen stark reduziert ist. Besonders aber die Beschränkung auf Tastatureingaben stellt sicher, dass die Abarbeitungspfade durch die Programme in den meisten Fällen abzählbar und als Konsequenz in ihrer Gesamtheit testbar sind. Die Arbeitsweise eines textbasierten Programms läuft in der Regel so ab, dass der Benutzer aufgefordert wird, eine bestimmte Eingabe zu tätigen. Nachdem dies erfolgt ist, begibt sich das Programm in einen neuen definierten Zustand und erwartet die nächste Eingabe [11], [1].

Eine ganz neue Situation bietet sich bei Programmen, die über eine grafische Benutzungsoberfläche verfügen. Hier kann viel mehr Information gleichzeitig in verschiedenen sich (teilweise) überlagernden Fenstern dargestellt werden. Die Bedienung erfolgt hauptsächlich über eine potentiell beliebige Anzahl von grafischen Steuerelementen[1], die dem Benutzer alle gleichzeitig zur Verfügung stehen. Die Bedeutung der Steuerung über optische Eingabegeräte (Computermaus) steigt in diesem Umfeld immens und damit direkt auch die Freiheit des Benutzers bei der Eingabe. Das GUI-basierte Programm wird dadurch nun mit einer unüberschaubaren Vielfalt an möglichen Eingaben in jedem Moment der Lauf-

[1] **Steuerelemente**: In Fenstern positionierte grafische Interaktionsobjekte, über die Benutzereingaben und Ausgaben realisiert und Ereignisprozeduren ausgelöst werden können. Beispiele sind Schaltflächen, Texteingabefelder oder Auswahllisten.

zeit konfrontiert. Der Benutzer erhält die Möglichkeit, in beliebiger Reihenfolge von einer Eingabe zur nächsten zu wechseln und ebenso nach Belieben zwischen mehreren gleichzeitig laufenden Programmen hin und her zu springen [11], [1].

Als Konsequenz kann ein GUI-Programm auf der Ebene seiner grafischen Benutzungsoberfläche einerseits mit nichts und muss andererseits mit allem rechnen.

Unter den meisten Betriebssystemen mit grafischer Benutzungsoberfläche arbeiten GUI-Programme ereignisgesteuert (event-driven), d.h. ihr Ablauf wird hauptsächlich durch die auftretenden Geschehnisse bestimmt. Dies umfasst ganz konkret eben auch alle Nutzereingaben, die zu jeder Zeit und in beliebiger Menge und Zusammensetzung auf das Programm „einströmen“ können. Daneben gibt es noch eine Vielzahl weiterer Ereignisse, auf welche die Anwendung reagieren muss, z.B. wenn ihre Oberfläche auf einmal ganz oder teilweise gelöscht (vielleicht durch ein anderes Fenster überdeckt) wurde und neu gezeichnet werden muss. Näher wird auf den Mechanismus der ereignisgesteuerten Natur von GUI-Programmen in Kapitel 5.1 eingegangen [11].

Der mit der Verwendung einer grafischen Benutzungsoberfläche einhergehenden Komplexität wird in verschiedener Hinsicht begegnet. Einerseits geschieht dies durch eine starke Modularisierung der GUI, die sich z.B. ich einem hierarchischen Aufbau der Oberfläche und der Zusammenfassung von funktionellen Einheiten niederschlägt. Daneben steht in den allermeisten Fällen eine Standardbehandlung für alle Ereignisse zur Verfügung, die für das konkrete Programm nicht (oder momentan nicht) von Bedeutung sind. Doch trotz allem existieren auch unter der Beschränkung auf sinnvolle Eingaben für die meisten Anwendungen so viele verschiedene mögliche Abarbeitungspfade, dass vollständige Funktionstests[2] auf der Ebene der grafischen Benutzungsoberfläche oftmals schlicht nicht möglich sind – schon wenn nur fünf Steuerelemente angezeigt werden und deren Benutzung in unterschiedlicher Reihenfolge getestet werden soll, ergeben sich $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$ Testfälle [19], [11].

Umfassende Funktions- und Regressionstests[3] auf der Ebene der GUI sind also aufgrund der angesprochenen Schwierigkeiten besonders aufwändig – aus denselben Gründen aber auch besonders wichtig. Eine Automatisierung ist so-

[2] **Funktionstest**: Es werden Abweichungen der tatsächlichen von der geforderten Funktionalität gesucht.

[3] **Regressionstest**: Ein erneuter Test einer bereits getesteten Software nach deren Modifikation mit dem Ziel nachzuweisen, dass durch die vorgenommene Änderung keine neuen Defekte eingebaut oder bisher maskierte Fehlerzustände freigelegt werden (nach IEEE Standard für Software Engineering Terminologie)

mit dringend erwünscht und notwendig, stellt aber wiederum eine eigene große Herausforderung dar. Im Gegensatz dazu lässt sich das Testen von textbasierten Programmen vergleichsweise leicht automatisieren, ähnlich wie das separate Testen der tieferen Funktionsschichten von GUI-Anwendungen.

2.2. Merkmale des GUI-basierten Testens

Grafische Benutzungsoberflächen setzen sich im Allgemeinen aus mehreren Steuerelementen zusammen, die hierarchisch gestaffelt und in einem oder mehreren Fenstern zusammengefasst sind. Zusätzlich steht meist eine geordnete Sammlung von Befehlen in Form eines Menüs zur Verfügung. Eingaben erhält die Anwendung hauptsächlich auf die Weise, dass der Benutzer die Steuerelemente manipuliert oder Kommandos aus dem Menü wählt. Als Reaktion darauf erhält er ein Feedback vom Programm, welches auch wieder mit Hilfe von Steuerelementen vermittelt wird. Konsequenterweise ist nun genau diese Methodik zu realisieren, um eine Applikation anzusteuern, wenn die Funktionalität auf der Ebene ihrer grafischen Benutzungsoberfläche getestet werden soll. Nach einer Eingabe muss kontrolliert werden, ob die Anwendung wie erwartet reagiert hat, was eben durch Prüfung des Zustands ihrer Steuerelemente geschieht. Zustände können dabei z.B. sein, dass ein Objekt unsichtbar ist oder inaktiv (d.h. momentan nicht bedient werden kann), andere Steuerelemente können Text aufnehmen oder zwischen verschiedenen Modi wechseln [11], [29].

Ein Merkmal von GUI-Programmen ist, dass es oft viele Wege gibt, wie eine Anwendung in eine bestimmte Situation gebracht werden kann. Abgesehen davon, dass sich die Aktionskette beliebig mit Maßnahmen auffüllen lässt, die keine Auswirkung auf die Erreichung des Zielzustands haben (sollten), lassen sich Befehle oder Einstellungen meist über mehrere Methoden realisieren. Häufige Möglichkeiten des Zugriffs sind über Tastaturkürzel, über die Funktionstasten und über die Maus (unter Nutzung von Steuerelementen oder Menüs) realisiert. Für die Entwicklung von Tests stellt sich nun die Frage, ob das Ziel sein soll, alle denkbaren Wege aufzunehmen, oder ob zumindest jede Zugriffsmöglichkeit einmal verwendet werden soll, oder ob es reicht, wenn die hinter der GUI liegende Funktionalität auf beliebigem Weg einmal aufgerufen wird. Die Entscheidung wird nicht nur davon abhängen, was die zuverlässigsten Testergebnisse liefert (sicher die erste Vorgehensmethode), sondern vor allem davon, wie viel mit den vorhandenen Ressourcen überhaupt machbar ist [11].

Die GUI-basierten Tests für eine Anwendung werden häufig noch immer manuell

durchgeführt – auf Kosten von Regelmäßigkeit (begründet durch den hohen Aufwand) und Exaktheit. Unterstützung durch Tools findet sich allenfalls relativ weit verbreitet durch Nutzung von Capture&Replay-Tools, die Maus- und Tastatureingaben aufzeichnen (Capturing) und später reproduzieren (Replay) können ([3], [1], [29]). Der Tester muss die Testfälle dann nur noch einmal zur Aufzeichnung manuell abarbeiten und kann sich in späteren Durchläufen darauf beschränken, die Reaktionen der Anwendung auf die aufgezeichneten Eingaben zu überwachen. Leider machen schon kleine Änderungen an der grafischen Benutzungsoberfläche im Verlauf der Entwicklung eines Programms oft eine vollständige Neuaufzeichnung der Eingaben (Recapturing) notwendig, zumindest aber Modifikationen an den beim Capturing erzeugten Skripts.

Auch wenn durch den Einsatz von Capture&Replay-Tools eine gewisse Automatisierung erreicht wird, kann dieser Ansatz in keiner Weise den Anforderungen genügen. Der Aufwand bleibt immer noch so hoch, dass in der Praxis umfangreiche Tests in der notwendigen Regelmäßigkeit nicht geleistet werden können. Erforderlich sind komplexe Software-Systeme, die den GUI-basierten Test vollständig automatisieren, so dass der Tester sich stärker auf seine eigentliche Aufgabe konzentrieren kann, nämlich das Spezifizieren von guten, vollständigen Testfällen. Für den komplett automatisierten Funktions- und Regressionstest müssen neben der Steuerung über die grafische Benutzungsschnittstelle auch umfangreiche Möglichkeiten für die Zustandsprüfung und verschiedene weitere Funktionen realisiert sein.

Neben kommerziellen Lösungen ([16], [21]) bietet die am Lehrstuhl Softwaretechnik des Instituts für Informatik der Humboldt-Universität zu Berlin entwickelte Testsuite ATOS (Automatisierter Test Oberflächenbasierter Systeme) bereits gute Möglichkeiten für automatische GUI-basierte Funktions- und Regressionstests (siehe Kapitel 3).

3. Das ATOS-Testsystem

Dieses Kapitel beschreibt das Software-System ATOS (Automatisierter Test Oberflächenbasierter Systeme) in seinem bisherigen Funktionsumfang. Die Testsuite entstand im Zeitraum 2001 bis 2003 am Lehrstuhl Softwaretechnik des Instituts für Informatik der Humboldt-Universität zu Berlin.

Das Einsatzgebiet ist der Funktions- und Regressionstest von Software mit grafischer Benutzungsschnittstelle für Microsoft® Windows®. ATOS bietet dafür die Funktionalität für das Testmanagement, die Testdurchführung und Testauswertung sowie Unterstützung bei der Erstellung von Tests [12].

3.1. Testmanagement

ATOS fasst alle Dateien und Einstellungen, die zu einem Testobjekt[1] gehören, in einem Projekt zusammen. Dies umfasst einige allgemeine Einstellungen wie z.B. die ausführbare Datei des Testobjekts oder den Titel des Hauptfensters, vor allem aber wird eine spezielle Verzeichnisstruktur angelegt, in der alle Testskript[2]-Dateien (Testsequenzen), Referenz-Dateien für Vergleichstests, Protokoll-Dateien usw. abgelegt werden. Mehrere Testsequenzen eines Projekts lassen sich dann in Testpaketen zusammenfassen, um so semantisch zusammengehörige Testfälle[3] entsprechend organisieren und auch als Einheit sequenziell ausführen zu können [13].

[1] **Testobjekt**: Komponente, integriertes Teilsystem oder System (in einer bestimmten Version), das einem Test unterzogen wird.

[2] **Testskript**: Anweisungen zur automatischen Ausführung eines Testfalles oder einer Testsequenz (oder übergeordnete Ablaufsteuerung weiterer Testwerkzeuge) in einer geeigneten Programmiersprache.

[3] **Testfall**: Ein Testfall enthält die festgelegten Rahmenbedingungen, die zur Überprüfung eines Testlaufs benötigt werden.

3.2. Testdesign

Um ATOS für ein Testobjekt einsetzen zu können, muss dessen Quelltext verfügbar sein. Genauer ist ATOS auf die speziellen Ressourcen-Dateien der verwendeten Entwicklungs-Umgebung (z.B. MS Visual C++, Borland C++ Builder) angewiesen, in denen diese Programme unter anderem die Informationen zum Aufbau der vom Testobjekt implementierten Dialoge und Menüs in einem jeweils unterschiedlichen Format ablegen. Für ATOS steht ein spezielles Zusatzprogramm *RCParser* zur Verfügung, das aus diesen Ressourcen alle wichtigen Daten herausfiltert. Das sind die Dialoge und die darin enthaltenen Steuerelemente – und zwar genauer die Kennziffern der Objekte – und nach Möglichkeit sinnvolle verbale Bezeichner. Dafür ist wichtig, dass jedes Fenster (und alle Steuerelement sind nichts anderes als spezielle Fenster) eine Kennziffer besitzt, die programmseitig gesetzt wird und somit immer gleich ist. Dialog-Elemente lassen sich somit üblicherweise darüber identifizieren, da auch das Testobjekt selbst auf diese Weise mit seinen Dialogen arbeitet. Das Handle[4], das Windows jedem Fenster zuordnet, ist für den Zweck einer wiederholten Identifikation nicht geeignet, da es nur zur aktuellen Laufzeit gültig ist. Für Menüs ermittelt der *RCParser* die vollständige Menüstruktur und für jeden Menüpunkt auch dessen Kennziffer und seine Bezeichnung.

Der Testskript-Designer braucht für seine spätere Arbeit die Kennziffern nicht zu wissen, da in den Testskript-Befehlen mit den verbalen Bezeichnungen der Steuerelemente oder Menüpunkte gearbeitet wird. Während der Testdurchführung werden die entsprechenden Kennziffern dann unter Zuhilfenahme der vom *RCParser* bereitgestellten Daten aus den verbalen Bezeichnern bestimmt. Das hat aber neben den Vorteilen der komfortablen Erstellung und besseren Lesbarkeit der Testskripte allerdings auch den Nachteil, dass für jedes Steuerelement ein eindeutiger verbaler Bezeichner festgelegt werden muss, was leider nur teilweise automatisch durch den *RCParser* geschehen kann [13], [14].

Mit Einführung der Capturing-Funktionalität können nun allerdings auch Projekte für Testobjekte realisiert werden, deren Quelltexte nicht vorliegen. Mehr dazu in Kapitel 4.2.

Testkommandos werden in ATOS unter Verwendung einer eigenen Skriptsprache *HTS* (High-Level Test Script Language) [15] angegeben. HTS umfasst Befehle verschiedener Kategorien (siehe Tabelle 3.1), die es erlauben, ein weites

[4]**Handle**: Als Handle werden Kennziffern bezeichnet, die verwendet werden, um Objekte einer Klasse (z.B. Fenster, Menüs, Icons usw.) eindeutig zu identifizieren. Bei Windows® sind das vorzeichenlose 32-Bit-Werte.

Kategorie	**Beispiele**
Steuern des Testobjekts	Mausklick auf eine Schaltfläche, Auswahl eines Kontrollkästchens oder Eintrag in ein Eingabefeld
Abfragen des Testobjekts	Status eines Kontrollkästchens, Sichtbarkeit eines Dialogfensters oder Auswahl eines Listenfensters
Operationen mit Dateien	Kopieren und Löschen, Ausführen anderer Programme, etwa zum Dateivergleich
midrule Sonstiges	Wertvergleiche, Schleifen, Interaktionen mit dem Tester, Kommentare

Tabelle 3.1.: Übersicht über den Umfang von HTS

Spektrum an Testfällen zu definieren und so zu automatisieren. Abfolgen von Testkommandos werden in Testsequenzen zusammengefasst, deren Erstellung direkt im ATOS-Hauptprogramm erfolgt. Dabei steht ein intelligenter Editor zur Verfügung, der die Programmierung stark vereinfacht und zur Fehlervermeidung beiträgt. Jedes Kommando besteht aus mehreren Komponenten oder Parametern und wird schrittweise zusammengesetzt, wobei der Editor, unter Verwendung der aus den Quelltexten des Testobjekts extrahierten Daten, zu jeder Zeit immer nur die Optionen anbietet, die zu einer gültigen Befehlszeile führen können. Auf diese Weise gestaltet sich der Prozess der Testskript-Programmierung zwar zeitintensiver als bei Verwendung eines einfachen Texteditors, dafür entfallen aber Recherchen in der HTS-Spezifikation oder den Testobjekt-Daten, und die später erforderliche Fehlerbearbeitung fällt deutlich geringer aus [13], [14].

Eine Stärke von ATOS ist die Übernahme von Änderungen, die sich aus Modifikationen der grafischen Benutzungsoberfläche des Testobjekts ergeben. Wenn die Quellen des Testobjekts mit dem Zusatzprogramm *RCParser* eingelesen wurden, kann dieses genutzt werden, um nach erfolgten Änderungen eine erneute Analyse der Ressourcen-Dateien durchzuführen. Die aktualisierten Daten werden ATOS zur Verfügung gestellt, wodurch direkt alle nun ungültig gewordenen Befehlszeilen als fehlerhaft markiert werden und vom Testskript-Designer überarbeitet werden können [13], [14].

Wie bereits deutlich wurde, ist ATOS in der Lage, die GUI des Testobjekts direkt zu bedienen. Es werden also nicht reine Maus- oder Tastatureingaben simuliert

(z.B. Mausklick an bestimmten Koordinaten des Bildschirms), wie es bei den meisten Capture&Replay-Tools der Fall ist, sondern die Steuerelemente werden direkt angesprochen. Das macht die Steuerung des Testobjekts durch ATOS robust und unabhängig von Größe und Position der (Dialog-)Fenster sowie gegenüber leichten Änderungen am Layout der grafischen Benutzungsoberfläche.

3.3. Testdurchführung und -auswertung

Die Testdurchführung kann – in Abhängigkeit von den Testskripten – vollständig automatisch ablaufen. Werden in den Testskripten interaktive Befehle verwendet (z.B. Fragen an den Tester über den Zustand des Testobjekts), dann ist hierfür an entsprechender Stelle natürlich das Eingreifen des Testers notwendig. Ansonsten aber erfordert auch die Abarbeitung großer Testpakete seine Anwesenheit nicht, denn die Ausführung jedes HTS-Kommandos wird überwacht und in eine Protokolldatei geschrieben. Sollte ein Fehler auftreten, setzt die Abarbeitung ggf. bei der nächsten Testsequenz wieder ein, so dass nach Abschluss des gesamten Durchlaufs eine Zusammenfassung über alle Testfälle mit Problemen zur Verfügung steht [13], [14].

Mit der ATOS-Testsuite lassen sich also GUI-basierte Funktionstests definieren, verwalten, durchführen und auswerten. Die Funktionalität ist mit der von kommerziellen Werkzeugen vergleichbar, allerdings bestehen auch deutliche Einschränkungen in bestimmten Bereichen (siehe dazu Abschnitt 4.1).

4. Motivation für eine Capturing-Erweiterung

In diesem Kapitel werden alternative Softwaresysteme zu ATOS vorgestellt und daran angelehnt wird darauf eingegangen, was zur Idee der Erweiterung von ATOS um die Funktionalität des Capturings geführt hat.

4.1. Funktionsumfang von Konkurrenzprodukten

Es gibt nicht viele kommerzielle Produkte, die das Einsatzgebiet von ATOS abdecken. Die weitaus größte Anzahl von Tools für den automatischen Software-Test arbeitet auf Code-Ebene und realisiert Komponenten-Tests. Dabei wird der Quelltext des Testobjekts durch zusätzliche externe Programmteile erweitert, die dann – entsprechend konfiguriert – zur Laufzeit die zu testenden Funktionen direkt programmintern aufrufen. Für diesen Ansatz stehen auch weit verbreitete Open-Source Produkte zur Verfügung, etwa die Familie der *X*Unit-Frameworks (z.B. JUnit [10], CppUnit [9], NUnit [27], PhpUnit [4]).

Für Funktionstests, die direkt auf der grafischen Oberfläche des Testobjekts arbeiten, wird besonders das Programm *WinRunner* der Firma Mercury™ [21] häufig eingesetzt. Auch IBM bietet mit dem Rational Functional Tester [16] ein ähnliches Produkt im Rahmen ihrer Rational®Software [17] an. Abgesehen davon gibt es eine Reihe von Lösungen kleinerer Software-Firmen, wie z.B. AutoTester®(AutoTester, Inc.) [2], QES/Architect (QES Testing b.v.) [25] oder QAHiperstation (Compuware Corporation) [24].

Diese Programme des kommerziellen Umfelds (besonders der größeren Anbieter) warten mit deutlich erweiterten Einsatzmöglichkeiten auf. So werden viele zusätzliche Steuerelemente unterstützt und es gibt Möglichkeiten zur Einbindung selbstdefinierter GUI-Objekte. Auch das Potenzial für Funktionstests für Anwendungen mit grafischen Oberflächen, die nicht direkt auf Elementen des Windows-API[1] basieren (z.B. Web- oder Java-Applikationen), die Unterstützung von alternativen Betriebssystemen und weitere Funktionen gehören üblicherweise

[1] **Windows-API**: Das Betriebssystem ermöglicht dem Programmierer den Zugriff auf viele integrierte Funktionen über eine definierte Software-Schnittstelle, dem Application Programming Interface (API).

zum Repertoire. Außerdem gestaltet sich die Arbeit mit den Produkten oftmals effizienter, als es mit ATOS der Fall ist. Einerseits ist die Erstellung von gängigen Testfällen einfacher und schneller möglich, andererseits aber stehen auch mächtigere Skriptsprachen als HTS zur Verfügung, um auch sehr spezielle Testsituationen realisieren zu können.

Die Umsetzung vieler der zusätzlichen Funktionen kommerzieller Werkzeuge würde die Möglichkeiten im universitären Umfeld übersteigen. Andererseits sind verschiedene Erweiterungen auch von untergeordnetem Interesse. Als wichtige Arbeitspunkte lassen sich aber die Unterstützung weiterer Standard-Steuerelemente und die Vereinfachung der Testskript-Erstellung hervorheben. Ersteres ist bislang dadurch eingeschränkt, dass das Konzept der Kommunikation von ATOS mit Steuerelementen die Einbindung komplexerer Controls zurzeit nicht erlaubt. Hier müsste eine tiefer liegende Funktionsschicht von ATOS umfassend überarbeitet werden. Thema dieser Diplomarbeit aber ist die Erweiterung von ATOS mit dem Ziel, das Umsetzen von Testfällen in Testskripte zu vereinfachen und weitgehend zu automatisieren.

Realisiert wurde dies durch Implementation eines Eingabe-Capturings mit nachgeschalteter Eingabe-Interpretation und -Übersetzung. Im Resultat genügt es zur Erstellung eines Testskripts, dass der Tester den Testfall direkt an dem Testobjekt durchführt. Mit Hilfe des Capturings werden dabei alle Eingaben (via Maus oder Tastatur) von ATOS mitgelesen und in Bezug auf das Testobjekt interpretiert. Ausgehend davon werden automatisch die passenden Testskript-Kommandos erzeugt, um die Aktionen zu reproduzieren. Auch die Generierung von Kommandos zur Statusüberprüfung findet direkt während der Bedienung des Testobjekts statt. Eine ähnliche Funktionalität wird auch von den kommerziellen Produkten geboten.

4.2. Vorteile der Arbeit mit Capturing gegenüber der manuellen Testskript-Erstellung

4.2.1. Neue Einsatzmöglichkeiten

Werden die Testskript-Befehle automatisch erzeugt, während das Testobjekt bedient wird, so können alle dafür benötigten Daten direkt ermittelt werden. Somit ist es nicht mehr zwingend notwendig, Informationen aus den Ressourcen-Dateien des Testobjekts auszulesen, und es wird möglich, Funktionstests für Applikationen zu erstellen und durchzuführen, für die keine Quelltexte verfügbar oder dessen Ressourcen-Dateien zu ATOS inkompatibel sind. Natürlich kann auch die

manuelle mit der automatischen Kommandoerzeugung kombiniert werden.

4.2.2. Logische Korrektheit der Testskripte

Manuell mit ATOS erstellte Kommandos sind syntaktisch korrekt und auch semantisch fehlerfrei, soweit das unter Verwendung der aus den Ressourcen-Dateien des Testobjekts extrahierten Daten überprüft werden kann. Automatisch mittels Capturing der Eingaben für das Testobjekt erzeugte Kommando-Sequenzen sind immer syntaktisch und semantisch korrekt, zusätzlich aber auch logisch ohne Fehler, d.h. Designfehler, wie ein vergessener oder falscher Aktionsschritt, können nicht auftreten.

4.2.3. Bessere Performance bei der Testskripterstellung

Die für den Anwender sicher markanteste Verbesserung dürfte ein enormer Performance-Gewinn sein. Die automatische Kommandoerzeugung gewährleistet eine deutliche Herabsetzung des Zeitaufwands zur Erstellung von üblichen Testszenarien. In vielen Fällen müssen nur noch Befehle für das Starten des Testobjekts und ggf. für Operationen mit Konfigurationsdateien manuell eingegeben werden.

5. Technische Grundlagen für das Capturing

Die Realisierung des Capturing-Mechanismus basiert größtenteils auf dem Hook-Konzept von Microsoft® Windows®. Dieses Konzept baut auf dem Windows-Nachrichtensystem auf, weshalb im Folgenden auf beides eingegangen werden soll.

5.1. Windows-Nachrichten

Das Nachrichten-Konzept von Microsoft® Windows® bildet grundsätzlich die Basis für sämtliche Kommunikation eines Fensters mit dem Restsystem. Das gilt für alle Arten von Fenstern, angefangen bei Haupt-Programmfenstern und Dialog-Fenstern bis hin zu Teilen von komplexen Steuerelementen. Allerdings können auch Programme, die keine grafische Oberfläche erzeugen, über Windows-Nachrichten kommunizieren. Die Kommunikationswege führen dabei in alle und aus allen möglichen Richtungen. So bekommt das Fenster vom System über Nachrichten z.B. mitgeteilt, wenn Eingaben stattfanden (bes. Maus, Tastatur), die das Fenster betreffen, oder wenn das Fenster verdeckt wurde und sich neu zeichnen muss. Wenn es sich um ein Steuerelement eines Dialog-Fensters handelt, so wird es vom Dialog über Nachrichten gesteuert und anders herum kann es auch den Dialog informieren, wenn sich z.B. sein Status geändert hat. Auch Timer können auf diese Weise realisiert werden, so kann sich ein Fenster in regelmäßigen Abständen eine entsprechende Nachricht vom System schicken lassen [23], [22].

Für jeden Thread[1], der Nachrichten verarbeiten soll, muss eine so genannte Window-Prozedur definiert werden. Wenn das System eine Nachricht ‚sendet', dann bedeutet das im Klartext, dass eben diese Funktion aufgerufen wird. Nachrichten, die das System nicht direkt ‚ausliefert', werden in eine Warteschlange eingereiht, die dem Programm zugeordnet ist und von diesem abgearbeitet

[1] **Thread**: Ein Programm kann (unter einem multitaskingfähigen Betriebsystem) seine Anweisungsfolgen in mehrere Ausführungspfade (Threads) aufteilen, die dann vom System im Sinne des Multitaskings quasi-parallel abgearbeitet werden.

werden muss. Zu diesem Zweck implementiert eine Windows-Anwendung üblicherweise eine so genannte Ereignis-Warteschleife, die immer, wenn das Programm nicht anderweitig ‚beschäftigt' ist, die Nachrichten der Warteschlange verarbeitet, d.h. die Window-Prozedur dafür aufruft. Zusammen mit einem Nachrichten-Typ, werden der Window-Prozedur noch zwei 32-Bit-Parameter (für die 32-Bit-Versionen von Windows®) übergeben, die je nach Art der Nachricht unterschiedlichste Informationen enthalten [23], [22].

Mit dem Nachrichten-Konzept von Windows werden zwei Bereiche gleichzeitig abgedeckt. Nachrichten, die direkt ‚gesendet' werden, indem die Window-Prozedur explizit aufgerufen wird, fungieren quasi als Interrupts, da das Programm in seinem Verhalten zu undeterminierten Zeitpunkten beeinflusst wird. Allerdings, und deshalb sind direkte Windows-Nachrichten keine wirklichen Interrupts, kann die Window-Prozedur, wenn sie eine Nachricht verarbeitet, darin nicht unterbrochen werden. Jedoch kann es vorkommen, dass die Window-Prozedur mehrfach gleichzeitig oder auch aus sich selbst heraus aufgerufen wird – deshalb muss sie reentrant[2] programmiert sein. Mit Nachrichten, die über die Warteschlange laufen, kann eine Synchronisierung erreicht werden, was z.B. ständig im Zusammenhang mit Benutzereingaben passiert.

5.2. Windows-Hooks

In Microsoft® Windows® steht mit Hooks (<engl.> Haken) ein Mechanismus zur Verfügung, um Ereignisse (wie Nachrichten oder Eingaben von Maus oder Tastatur) zu verarbeiten, bevor sie eine Anwendung erreichen. Dabei besteht in einigen Fällen sogar die Möglichkeit, die Ereignisse zu modifizieren oder die Weiterleitung zu blockieren [20], [22].

Eine Applikation kann einen oder mehrere Hooks installieren. Dabei unterscheidet man zwischen systemweiten und prozessspezifischen Hooks. Im ersten Fall wird Zugriff auf die Ereignisse sämtlicher Threads des Systems erlangt, im zweiten Fall nur auf Ereignisse eines bestimmten Threads (eines beliebigen Prozesses). Nachrichten durchlaufen dann zunächst eine bei der Hook-Installation definierte Funktion (Filterfunktion), bevor sie von der Window-Prozedur der zuge-

[2] **Reentrante Funktion**: Wenn eine einfache Kopie des Programmkodes einer Funktion im Speicher von mehreren Prozessen gleichzeitig ausgeführt werden soll, muss diese eintrittsinvariant (reenterant) programmiert sein, d.h. es muss sichergestellt werden, dass es sich nicht um selbstmodifizierenden Kode handelt und dass prozesseigene Informationen (z.B. lokale Variablen) in getrennten Speicherbereichen verwaltet werden.

ordneten Anwendung verarbeitet werden können. Es stehen verschiedene Typen für Filterfunktionen zur Verfügung, denen jeweils unterschiedliche Ereignisklassen zugeordnet sind (siehe Tabelle 5.2). In Abhängigkeit des Typs der Filterfunktion können die Nachrichten entweder nur inspiziert oder modifiziert oder sogar vollständig blockiert werden. Wenn mehrere Hooks desselben Typs installiert werden, dann durchläuft eine Nachricht der Reihe nach jede der Filterfunktionen. Als erstes erreicht sie dabei den zuletzt installierten Hook, der somit die größte Einflussmöglichkeit hat, da nachfolgende Hooks nur die von Vorgängern nicht blockierten Nachrichten ggf. in modifizierter Form erreichen [20], [22].

5.3. Verwendung von Hooks für das Capturing

Es ist sicher deutlich geworden, dass mit Windows-Hooks ein Mechanismus zur Verfügung steht, der einer Anwendung erlaubt, die Eingaben für ein anderes Windows-Programm mitzuschneiden, also Capturing durchzuführen. Um dieses grundsätzlich zu realisieren, reicht es, wenn die Anwendung eine Filterfunktion installiert, die alle Maus- und Tastatur-Ereignisse für das andere Programm verarbeitet, denn das System sendet für alle Nutzereingaben Nachrichten an das momentan aktive Fenster.

Bei Betrachtung der zur Verfügung stehenden Varianten von Hooks (siehe Tabelle 5.2), fällt sicher der Typ `WH_JOURNALRECORD` als scheinbar besonders geeignet auf, stellt doch Windows in Verbindung mit `WH_JOURNALPLAYBACK` alles für das Record&Replay von Ereignissen zur Verfügung. Allerdings sind die Journal-Hooks tatsächlich für den Zweck des Capturings von Eingaben für ein anderes Programm nicht geeignet, denn während ein Journal-Hook aktiv ist, kann nur die Anwendung bedient werden, die den Hook installiert hat. Abgesehen davon erlaubt ein Journal-Hook nur das reine Mitlesen von Maus- und Tastatur-Ereignissen, was für den angestrebten Funktionsumfang des Capturings in ATOS (vgl. Abschnitt 6.1) nicht genügt. Als beste Lösung hat sich schließlich die gleichzeitige Verwendung von drei verschiedenen Hook-Typen herausgestellt (`WH_KEYBOARD`, `WH_MOUSE` und `WH_GETMESSAGE`), wie in Kapitel 6 detailliert dargestellt wird.

Typ	Beschreibung
WH_CALLWNDPROC	Windows ruft diesen Hook, für Ereignisse, die über die Nachrichten-Warteschlange laufen.
WH_CBT	Dieser Hook ist speziell für Anwendungen für Computer-Based Training gedacht.
WH_DEBUG	Eine Filterfunktion dieses Typs wird immer gerufen, bevor ein Ereignis an einen der anderen Hooks weitergeleitet wird.
WH_FOREGROUNDIDLE	Dieser Hook wird immer dann gerufen, wenn sich der zugeordnete Thread im Leerlauf befindet.
WH_GETMESSAGE	Windows ruft diesen Hook immer dann, wenn ein Programm über die GetMessage() oder PeekMessage() Funktion eine Nachricht abholt bzw. sichtet.
WH_JOURNALRECORD	Hooks dieses Typs können dazu verwendet werden, Maus- und Tastatur-Ereignisse für einen Thread zu protokollieren und ggf. aufzuzeichenen.
WH_JOURNALPLAYBACK	Unter Verwendung dieses Hooks können mit WH_JOURNALRECORD aufgezeichnete Ereignisse ausgeführt werden.
WH_KEYBOARD	Filterfunktionen dieses Typs verarbeiten alle Tastatur-Ereignisse.
WH_MOUSE	Filterfunktionen dieses Typs verarbeiten alle Maus-Ereignisse.
WH_MSGFILTER	Hooks dieses Typs empfangen Nachrichten, die an ein Dialog-Fenster, ein Menü oder eine Scroll-Leiste eines Threads gehen oder wenn der Anwender die Tastenkombinationen ALT+TAB oder ALT+ESC drückt.
WH_SYSMSGFILTER	Dieser Typ ist identisch mit WH_MSGFILTER mit dem Unterschied, dass hier systemweit die entsprechenden Ereignisse empfangen werden.
WH_SHELL	Dieser Hook wird immer dann gerufen, wenn ein Top-Level Fenster erzeugt oder zerstört wird.

Tabelle 5.2.: Die Typen von Filterfunktionen für Windows-Hooks [20]

6. Realisierung

Dieses Kapitel beschreibt detailliert alle Aspekte der Entwicklung der Capturing-Erweiterung für die Testsuite ATOS. Es umfasst vor allen Dingen das Vorgehen bei der Auswertung der Eingaben für das Testobjekt sowie die Verbindung und Kommunikation zwischen den verschiedenen Programmteilen.

6.1. Anforderungen an die Capturing-Erweiterung für ATOS

Vor Beginn der Arbeit an der Realisierung wurden einige Anforderungen definiert, die von der Erweiterung zu erfüllen waren.

6.1.1. Saubere Integration in das bestehende System

ATOS sollte in seinem bisherigen Funktionsumfang uneingeschränkt weiter nutzbar bleiben. Das Ziel war außerdem nicht die Entwicklung eines eigenständigen Moduls (wie z.B. der *RCParser*), sondern die Eingliederung in das bestehende Hauptprogramm. Wichtig war zudem die Forderung, dass die Möglichkeit bestehen sollte, in einer Testsequenz gleichzeitig sowohl mit der manuellen Eingabe zu arbeiten, als auch die automatische Erzeugung von Testkommandos mittels Capturing zu nutzen.

6.1.2. Leistungsumfang

Die Capturing-Erweiterung sollte dem Tester erlauben, Testskripte für Windows-Programme aufzuzeichnen, die mit den Möglichkeiten von ATOS angesteuert werden können. Dabei war gefordert, dass neben Aktions-Kommandos auch Befehle für Status-Tests generiert werden können. Außerdem waren für das Capturing die über ATOS ansprechbaren Steuerelemente und die Verwendung von Menüs zu unterstützen.

6.2. Oberfläche und Bedienung

Das Capturing-Modul wird aus dem ATOS-Hauptprogramm im Kontext einer Testsequenz aufgerufen und präsentiert sich als übersichtliches Dialog-Fenster (siehe Abbildung 6.1), das mit wenigen selbsterklärenden Steuer-

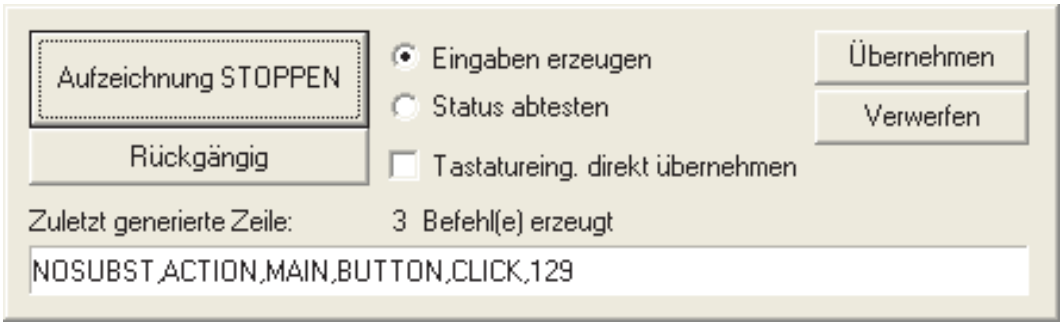

Abbildung 6.1.: Bildschirmansicht des Capturing-Dialogs

und Informationselementen auskommt und somit wenig Platz einnimmt. Das ATOS-Hauptfenster wird ausgeblendet, solange das Capturing-Modul aktiv ist. Sofern das Testobjekt noch nicht gestartet ist, wird es automatisch aufgerufen. Auch während der Eingabeaufzeichnung wird der Capturing-Dialog stets im Vordergrund gehalten, was jederzeit den Zugriff auf die Steuerung erlaubt. Bei aktivem Capturing signalisiert ein optisches Feedback in Form eines farblichen „Aufblitzens“ des Capturing-Dialogs, wenn durch eine Aktion ein Testskript-Befehl generiert wurde. Dabei können Aktionen zur Neuerzeugung eines Kommandos führen oder bewirken, dass die zuletzt generierte Befehlszeile modifiziert wird. Beide Fälle werden farblich unterschiedlich signalisiert.

Während des Capturings kann zwischen Aktions-Modus und Test-Modus (siehe Abschnitt 6.5) gewechselt werden, um Kommandos zu generieren, die entweder das Testobjekt steuern oder dessen Status auf den aktuellen Zustand prüfen. Alle Befehlszeilen, die das Capturing erzeugt, werden mit Kommentaren versehen, um eine leichtere Nachbearbeitung zu ermöglichen. Wenn die Aufzeichnung erfolgreich war, können die generierten Kommandos schließlich in die aktuelle Testsequenz übernommen werden.

6.3. Schnittstellen zwischen Hooks und Capturing-Modul

Das Capturing-Modul installiert Thread-spezifische Hooks für den Thread des aktiven Testobjekts. Wenn dann den verwendeten Hook-Typen entsprechende Ereignisse für das Testobjekt eintreffen, werden die definierten Filterfunktionen von Windows gerufen. Dabei findet die Ausführung im Kontext des Testobjekts statt. Damit dieser externe Zugriff möglich ist, wurden die Filterfunktionen in einer separaten DLL (Dynamic Link Library) untergebracht, während Daten, die

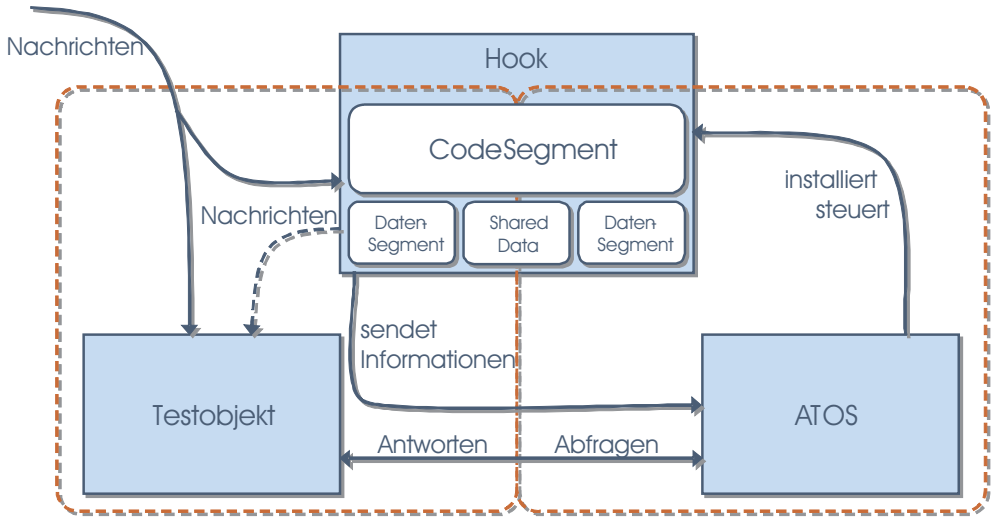

Abbildung 6.2.: Übersicht über das Capturing

im Kontext von ATOS und im Kontext des Testobjekts einheitlich zur Verfügung stehen müssen, in einem Shared Data Segment[1] abgelegt werden. Neben der Installation und Deinstallation nimmt das Capturing-Modul weitergehend steuernd Einfluss auf die Hooks, insbesondere durch Setzen des Capturing-Modus (ACTION oder TEST) über entsprechende Funktionsaufrufe.

Der Informationsfluss in die andere Richtung, d.h. von den Filterfunktionen zum Capturing-Modul, wird über das Windows-Nachrichtensystem realisiert. Die Filterfunktionen, im Kontext des Testobjekts gerufen, verarbeiten die eintreffenden Nachrichten. Für das Capturing sind dabei nur bestimmte Ereignisse von Interesse – diese werden dann zusammen mit weiteren benötigten Informationen in Form einer nutzerdefinierten Windows-Nachricht an das Capturing-Modul von ATOS übermittelt. Das Windows-Nachrichtensystem bietet eine sehr effiziente Möglichkeit, die Daten ereignisgesteuert zu senden. Die einzige Einschränkung besteht darin, dass für zusätzliche Informationen nur die zwei 32-Bit-Parameter zur Verfügung stehen, was sich jedoch als ausreichend herausgestellt hat. Zeiger auf Speicher mit zusätzlichen Daten über die Nachrichten mitzusenden, ist aufgrund der getrennten Speicherbereiche nicht möglich. Eine Nutzung des Shared Data Segments dagegen könnte Synchronisationsprobleme mit sich bringen und wäre weniger performant.

[1] **Shared Data Segment**: Speicher, der von mehreren Prozessen gleichzeitig benutzt wird. Normalerweise sind die Speicherbereiche aller Prozesse streng voneinander getrennt, was die Stabilität eines Betriebssystems stark positiv beeinflusst.

6.4. Capturing der Nutzereingaben

Für die Verwendung mit ATOS reicht es nicht, wenn der Capturing-Mechanismus nur reine Maus- und Tastatur-Ereignisse aufzeichnet, ohne Bezug zur konkreten grafischen Benutzungsoberfläche des Testobjekts. Statt dessen muss jede Aktion in Abhängigkeit des Steuerelements interpretiert werden, auf die sie sich tatsächlich bezieht.

Wie bereits in Abschnitt 5.1 dargestellt, läuft die Kommunikation eines Windows-Programms mit seinen Steuerelementen hauptsächlich über das wechselseitige Senden von Windows-Nachrichten ab. Möchte ein Programm beispielsweise darauf reagieren, dass eine Schaltfläche eines Dialogs angeklickt wurde, so verarbeitet es in seiner Window-Prozedur die Windows-Nachricht `WM_COMMAND`. Über eine Auswertung der beiden Parameter der Nachricht kann bestimmt werden, ob sie von der fraglichen Schaltfläche stammt und ob es sich um das gesuchte Ereignis (Klick auf die Schaltfläche) handelt. Auch im Falle von Menüereignissen (Auswahl eines Menüpunkts durch den Benutzer) erhält das Programm eine `WM_COMMAND` Nachricht.

Es liegt also nahe, genau diese Nachrichten in der Filterfunktion eines Hooks zu verarbeiten, um so direkt alle wichtigen Ereignisse für die Steuerelemente und das Menü des Testobjekts zu erhalten. Leider lässt sich jedoch schnell feststellen, dass diese Nachrichten offensichtlich nicht über die globale Windows-Nachrichtenverwaltung laufen, denn über Hooks gelingt der Zugriff darauf nicht. Es wäre auch ein eher unzuverlässiger Mechanismus, und er würde bestimmte Anforderungen an das Testobjekt stellen. Ob und über welche Ereignisse Steuerelemente ihr Eltern-Fenster informieren, kann nämlich individuell unterschiedlich eingerichtet sein.

Um eine höchstmögliche Unabhängigkeit vom Design des Testobjekts zu erreichen, wurde für ATOS ein Eingabe-Capturing auf höchster Ebene realisiert, dem sich eine Analyse der zu erwartenden Wirkung auf das Testobjekt anschließt. Detailliert wird dies in den folgenden Kapiteln dargestellt.

6.4.1. Capturing von Maus-Aktionen

Für das Capturing von Maus-Aktionen wird ein Hook vom Typ `WH_MOUSE` (vgl. Tabelle 5.2) installiert. Die Filterfunktion beschränkt sich auf die Verarbeitung aller `WM_LBUTTONDOWN`- (linke Maustaste gedrückt) und `WM_LBUTTONUP`-Ereignisse (linke Maustaste losgelassen) – Nachrichten über Maus-Bewegungen

```
#define WM_CAPTURE_LMOUSEDOWN ... SendMessage(
    hWnd,           // Dialog des Capturing-Moduls
    WM_CAPTURE_LMOUSEDOWN,
    wParam,         // LOWORD: x-Koordinate
                    // HIWORD: y-Koordinate
    lParam          // Handle des ermittelten Zielfensters
);
```

Abbildung 6.3.: Übermittlung des Ereignisses „linke Maustaste gedrückt“

oder das Drücken anderer Maustasten werden also ignoriert. Mit jeder Nachricht stellt Windows die Bildschirmkoordinaten zur Verfügung, die der Mauszeiger zum Zeitpunkt des Ereignisses hatte. Ausgehend davon bestimmt die Filterfunktion das unterste Fenster in der Fensterhierarchie[2] (also das Kindfenster der jüngsten Generation) des Testobjekts unter den gegebenen Koordinaten. Das Handle dieses Fensters wird zusammen mit den Bildschirmkoordinaten als Parameter einer nutzerdefinierten Nachricht vom Typ `WM_CAPTURE_LMOUSEDOWN` oder `WM_CAPTURE_LMOUSEUP` an das Capturing-Modul übermittelt (vgl. Abbildung 6.3).

6.4.2. Capturing von Tastatur-Eingaben

Um die Tastatur-Eingaben für das Testobjekt zu ermitteln, wird ein Hook vom Typ `WH_KEYBOARD` (vgl. Tabelle 5.2) verwendet. Die Filterfunktion wird jedes Mal gerufen, wenn eine Taste auf der Tastatur heruntergedrückt oder losgelassen wurde, während das Testobjekt den Eingabefokus besitzt. Verarbeitet werden aber nur die Ereignisse für „Taste gedrückt“. Für die spätere Auswertung der Eingaben ist es notwendig, dass bekannt ist, welches Steuerelement des Testobjekts den Eingabefokus zur Zeit des Tastendrucks hatte, denn in Abhängigkeit davon hat die Eingabe eine unterschiedliche Wirkung. An dieser Stelle ist es entscheidend, dass die Filterfunktionen von Hooks immer im Kontext der jeweiligen Zielanwendung gerufen werden, denn nur so besteht die Möglichkeit, das aktuelle Steuerelement mit dem Eingabefokus direkt zu ermitteln. Dieses aus einem an-

[2] **Fensterhierarchie**: Neben dem Hauptfenster (normalerweise mit Titelzeile, Menüzeile usw.) stellen auch die meisten kleineren „Bauteile“ eines Windows-Programms (z.B. alle Steuerelemente, wie Schaltflächen, Eingabefelder usw.) Fenster dar. Alle Fenster sind in einer Eltern-Kind-Fensterhierarchie angeordnet, d.h., ein Fenster kann mehrere Kindfenster haben und jedes Fenster kann ein Elternfenster haben.

```
#define WM_CAPTURE_KEY ... SendMessage(
    hWnd,          // Dialog des Capturing-Moduls
    WM_CAPTURE_KEY,
    wParam,        // LOWORD: 0
                   // HIWORD: virtueller Tastenkode
    lParam         // Handle des Fensters mit dem Fokus
);
```

Abbildung 6.4.: Übermittlung von Tastatur-Eingaben

deren Thread heraus zu bestimmen, kostet wesentlich mehr Aufwand und somit mehr Rechenzeit.

Das Capturing-Modul erhält die Informationen, welche Taste gedrückt wurde und welches Steuerelement den Eingabefokus hatte, mittels einer nutzerdefinierten Windows-Nachricht vom Typ `WM_CAPTURE_KEY` (vgl. Abbildung 6.4).

6.4.3. Capturing von Menü-Aktionen

Das Capturing von Menü-Aktionen ist nicht auf so direkte Weise möglich wie das Mitlesen der bisher behandelten Nutzereingaben. Ein Windows-Programm wird über ein Menü-Ereignis mittels einer `WM_COMMAND` Nachricht informiert, die als Parameter die Kennziffer des Menü-Befehls enthält. Diese Nachrichten lassen sich auch über einen Hook mitschneiden, da sie über die normale Nachrichtenverarbeitung laufen – Menüs gehören nämlich nicht zum Client-Bereich der grafischen Oberfläche eines Windows-Programms. Allerdings ist dieser Mechanismus nicht zuverlässig: So werden z.B. teilweise auch bei Klicks auf Schaltflächen als Menü-Aktionen gekennzeichnete `WM_COMMAND` Nachrichten empfangen. Ein weiteres Problem besteht darin, dass diese Nachrichten manchmal (aber eben nicht immer) doppelt auftauchen, wobei aber eine Entscheidung, ob der Menüpunkt zweimal hintereinander angewählt wurde oder nicht, möglich wäre. Allerdings zeigen diese Beispiele, dass ein Mechanismus des Capturings von Menü-Aktionen über das Auswerten der `WM_COMMAND` Nachrichten nicht sehr robust sein kann.

Die realisierte Methode setzt wieder eine Schicht höher an und nimmt eine eigene Interpretation der Eingaben vor. Über einen Hook vom Typ `WH_CALLWNDPROC` (vgl. Tabelle 5.2) wird auf Nachrichten vom Typ `WM_INITMENU` und `WM_MENUSELECT` reagiert. Windows sichert zu, dass immer wenn ein Menü

aktiviert wird, die Windows-Nachricht `WM_INITMENU` gesendet wird. Beim Auftreten dieser Nachricht ermittelt die Filterfunktion das (Dialog-)Fenster, zu dem das Menü gehört, und sendet diese Information über eine nutzerdefinierte Windows-Nachricht vom Typ `WM_CAPTURE_MENU` (vgl. Abbildung 6.5) an das ATOS-Capturing-Modul.

Eine Nachricht vom Type `WM_MENUSELECT` wird immer dann gesendet, wenn ein neuer Menüpunkt aktiv wird, d.h. wenn der Mauszeiger über einen neuen Menüpunkt geführt wird oder über die Pfeiltasten der Tastatur eine neue Auswahl erfolgt. Wichtig dabei ist, dass immer, bevor ein Menübefehl ausgeführt wird, der Menüpunkt zuvor zunächst markiert wird. Diese Tatsache erlaubt es, folgende Maus- oder Tastatur-Ereignisse auszuwerten, um zu bestimmen, ob der zuletzt markierte Menüpunkt ausgeführt wurde oder nicht und ob das Menü geschlossen wurde oder noch aktiv ist. Das geschieht jedoch erst im Capturing-Modul selbst. Dieses bekommt durch die Filterfunktion eine nutzerdefinierte Windows-Nachricht vom Typ `WM_CAPTURE_MENU` übermittelt, zusammen mit der Kennziffer des aktuell markierten Menüpunkts und dem Handle des aktiven Menüs (vgl. Abbildung 6.5). Wenn es sich um einen deaktivierten oder ausgegrauten Menüpunkt handelt, um eine Trennlinie oder um ein Popup-Menü (d.h. einen Menüpunkt, der ein Untermenü aufklappt), dann wird 0 anstelle der Kennziffer übertragen. Klicks auf diese Menüelemente haben üblicherweise keine Auswirkung.

6.5. Interpretation und Auswertung der Nutzereingaben

Dem Tester stehen während der Arbeit mit dem Capturing-Modul zwei Modi zur Verfügung. Im Modus *ACTION* werden aus den Eingaben für das Testobjekt solche Testskript-Befehle generiert, die durch Simulation von Nutzeraktionen das Testobjekt steuern. Die Eingaben während des Capturings werden dabei unverändert an das Testobjekt weitergeleitet, so dass es sich normal bedienen lässt. Der Modus *TEST* dagegen dient zum Erfassen des aktuellen Status des Testobjekts. Alle Eingaben über Maus und Tastatur, die das Testobjekt betreffen, werden in diesem Modus blockiert – eine Steuerung wird also unterbunden – und nur durch das Capturing-Modul ausgewertet. Dieses erzeugt dann Testskript-Kommandos, die die Elemente des Testobjekts auf verschiedene Weise abfragen.

6.5.1. Identifikation von Steuerelementtypen

Für das ATOS-Capturing-Modul ist es für die Interpretation der Maus- und Tastatur-Eingaben in Abhängigkeit des jeweils betreffenden Steuerelements von zentraler Bedeutung, dessen Typ zu ermitteln. Dieses gelingt, indem der

```
#define WM_CAPTURE_MENU

#define CAPTURE_MENU_INIT ... SendMessage(
    hWnd,           // Dialog des Capturing-Moduls
    WM_CAPTURE_MENU,
    wParam,         // LOWORD: CAPTURE_MENU_INIT
                    // HIWORD: 0
    lParam          // Handle des Fensters, zu dem
                    // das Menü gehört
);
```

```
#define WM_CAPTURE_MENU

#define CAPTURE_MENU_SELECT ... SendMessage(
    hWnd,           // Dialog des Capturing-Moduls
    WM_CAPTURE_MENU,
    wParam,         // LOWORD: CAPTURE_MENU_SELECT
                    // HIWORD: Kennziffer des Menüpunkts oder 0
    lParam          // Handle des Menüs
);
```

Abbildung 6.5.: Übermittlung von Menü-Aktionen

Klassenname des jeweiligen Fensters bestimmt wird, wobei sich jedoch besonders für die Klasse `BUTTON` ein Problem ergibt. Wie in Tabelle 6.5.1 dargestellt, vereint diese Klasse eine Anzahl unterschiedlichster Steuerelemente. Zusätzlich zu den in der Tabelle aufgelisteten (von ATOS unterstützten) werden auch noch die Typen `GROUPBOX` (Rahmen) und `OWNERDRAW` (selbst gezeichnete Schaltfläche) von `BUTTON` realisiert. Eine Unterscheidung ist jedoch zwingend erforderlich, da jedes dieser Steuerelemente unterschiedlich bedient und abgefragt werden muss. Eine Möglichkeit dafür ergibt sich über die Auswertung des Fensterstils[3] eines betreffenden Elements. Fensterstile werden jedoch nicht direkt durch jeweils ein gesetztes bzw. nicht gesetztes Bit des Stil-Eigenschaftsparameters des Fensters repräsentiert, sondern oft durch Kombinationen, wobei sich verschiedene Stile auch gegenseitig beeinflussen oder implizit enthalten. Im Resultat muss bei der Prüfung des Fensterstils auf gesetzte Bits eines BUTTON-Stils mit dem Auftreten von falschen Ergebnissen gerechnet werden. Die sequenzielle Abtestung aller BUTTON-Stile in einer bestimmten Reihenfolge führt aber zu einer sicheren Unterscheidung.

Eine weitere Besonderheit stellen Steuerelemente dar, die zwar auf den Standard-Typen basieren, aber in einer erweiterten Form im Testobjekt eingesetzt werden. Oft finden sich dann abgeleitete Typen, die über die üblichen Mechanismen abgefragt und gesteuert werden können, sich aber über einen anderen Klassennamen identifizieren. Um auch diese potenziell kompatiblen Steuerelemente unterstützen zu können, stellt ATOS die Möglichkeit bereit, in einer Konfigurationsdatei für das Capturing-Modul Klassennamen-Synonyme anzugeben.

6.5.2. Modus ACTION

Im *ACTION*-Modus wird das Testobjekt durch den Testskript-Designer normal gesteuert. Das Capturing-Modul erzeugt dabei automatisch die nötigen Testskript-Kommandos, um dasselbe Verhalten in Form einer Testsequenz reproduzieren zu können. Dieser Modus dient vorwiegend dazu, das Testobjekt in den Zustand zu bringen, in dem es anschließend überprüft werden soll, bzw. es kann dann getestet werden, ob dieser Zustand tatsächlich erreicht wurde. Aber natürlich können unter Umständen auch schon während der Abarbeitung der

[3] **Fensterstil**: Jedem Fenster ist in Windows ein 32-Bit-Wert zugeordnet, der für Flags verwendet wird, die Stilparameter festlegen. Dazu zählen allgemeine Angaben, wie z.B. die Sichtbarkeit (`WS_VISIBLE`) oder ob das Fenster eine Titelzeile besitzt (`WS_CAPTION`), und spezielle Stilparameter, die sich auf bestimmte Fensterarten beziehen, z.B. für Steuerelemente der Klasse `BUTTON` die Entscheidung über den Typ `BS_PUSHBUTTON` (Schaltfläche), `BS_CHECKBOX` (Kontrollkästchen), `BS_RADIOBUTTON` (Optionsfeld) usw.

Steuerelement		**Fensterklasse**
HTS-Typ	Bezeichnung	
BUTTON	Schaltfläche	BUTTON
CHECKBOX	Kontrollkästchen	
RADIOBUTTON	Optionsfeld	
STATIC	Statisches Element	STATIC
EDITBOX	Eingabefeld	EDIT
LISTBOX	Listenfeld	LISTBOX
COMBOBOX	Kombinationsfeld	COMBOBOX
TAB	Registerfenster	SYSTABCONTROL

Tabelle 6.1.: Übersicht Steuerelemente und ihre Fensterklassen

Steuerkommandos Fehler erkannt werden, wenn deren Ausführung fehlschlägt.

Eingaben über die Maus

Im Fall von Mausklicks erhält das Capturing-Modul die entsprechenden Bildschirmkoordinaten, ein Handle zu dem Fenster des Testobjekts, das durch den Klick beeinflusst wird, und die Information, ob die Maustaste gedrückt oder losgelassen wurde. Ein Befehl dabei wird erst dann generiert, wenn die gedrückte Maustaste wieder losgelassen wurde, und auch nur dann, wenn das Drücken und Loslassen der Taste demselben Zielfenster (Steuerelement) zuzuordnen ist. Das simuliert das übliche Verhalten von vielen Steuerelementen unter Windows. So ist es möglich, beim versehentlichen Anklicken eines Steuerelements dessen Auslösung zu vermeiden, indem die Maus vor dem Loslassen der Taste aus dem Bereich des Elements hinausbewegt wird.

Handelt es sich bei dem Zielfenster um einen kompatiblen Typ (also ein unterstütztes Steuerelement), muss dessen Status noch dahingehend geprüft werden, ob sich das Steuerelement im aktivierten Zustand befindet oder die Eingabe gar nicht verarbeiten würde, bevor schließlich in Abhängigkeit von Typ und ggf. Status des Steuerelements ein entsprechender HTS-Testskript-Befehl generiert wird.

Es gibt Fälle, in denen neue Testskript-Kommandos vorhergehende Kommandozeilen überflüssig machen. Diese Zeilen sollten aus Gründen der Übersichtlichkeit und Performance einer Testsequenz unbedingt vermieden wer-

den. Deshalb erkennt das ATOS-Capturing-Modul solche Fälle und modifiziert dann die bestehenden Zeilen anstelle der Generierung eines neuen Kommandos. Das betrifft z.B. die Behandlung von Listen- und Kombinationsfeldern.

Eingaben über die Tastatur

Zur Auswertung von Tastatureingaben steht dem Capturing-Modul neben dem virtuellen Tastenkode auch das Handle des Fensters (Steuerelements) mit dem Eingabefokus zur Verfügung. Abhängig von dessen Typ und der gedrückten Taste können zunächst zwei grundsätzlich unterschiedliche Behandlungsmethoden differenziert werden. Bestimmte Steuerelemente ändern ihren Zustand oder werden ausgelöst, wenn bestimmte Tastatureingaben erfolgen, während sie den Eingabefokus besitzen. So werden z.B. Schaltflächen getriggert und in Kontrollkästchen die Marker gesetzt bzw. gelöscht, wenn der Nutzer die Leertaste drückt. Alle Tasten für alphanumerische Zeichen werden jedoch z.B. ignoriert. Anders sieht das bei Eingabefeldern aus, wo jedes alphanumerische Zeichen den Inhalt des Steuerelements ändert. In diesen Fällen generiert das Capturing-Modul die entsprechenden, für die jeweiligen Steuerelemente spezifischen Aktions-Kommandos (z.B. HTS-Befehl `ACTION, ..., EDITBOX, EDIT, "Inhalt", ...`). Wenn aber für das Steuerelement mit dem Eingabefokus auf eine Tastatureingabe keine spezifische Reaktion definiert ist, wird der Input meist in einen Testskript-Befehl umgesetzt, der den konkreten Tastendruck direkt simuliert (z.B. HTS-Befehl `KEYBOARD, KEYCODE, 102, ...`).

Wenn Aktions-Kommandos für Steuerelemente generiert werden, kommt auch hier, wie im Fall von Eingaben über die Maus, der Mechanismus zur Vermeidung überflüssiger Befehlszeilen zum Tragen. Das betrifft ganz im speziellen Eingabefelder, wo aufeinander folgende Eingaben immer wieder den Inhalt ändern.

Viele Windows-Programme sollen zusätzlich zur Bedienung mit Maus und Tastatur auch ausschließlich über die Tastatur steuerbar sein. Nachdem der Anwender Routine im Umgang mit dem Programm erlangt hat, ist eine reine Tastaturbedienung oftmals wesentlich performanter. Um das Verhalten eines Testobjekts für diese Anwendung in einem Funktionstest zu erfassen, ist die Interpretation der Eingaben und Umsetzung auf Aktions-Kommandos für Steuerelemente nicht sinnvoll. Deshalb bietet das Capturing-Modul die Option, sämtliche Tastatureingaben ohne Auswertung direkt in Testskript-Befehle zu übernehmen, die lediglich den jeweiligen Tastendruck simulieren (vgl. Abbildung 6.1). Ein solcher reiner Tastatur-Funktionstest war mit dem ATOS-System in seiner bisherigen

Form mit vertretbarem Aufwand nicht realisierbar.

Menü-Aktionen

Das Capturing von Menü-Aktionen kann nicht so unmittelbar erfolgen, wie das bei Maus- und Tastatureingaben für Steuerelemente der Fall ist, da die Ereignisse in Reaktion auf Menükommandos nicht zuverlässig verarbeitet werden können, wie in Abschnitt 6.4.3 bereits dargestellt wurde. Statt dessen werden verschiedene Ereignisse im Zusammenhang mit Menüs verwendet, um die benötigten Informationen zu erhalten.

Immer, wenn ein Menü aktiv wird, z.B. durch Öffnen eines Eintrags der Menüleiste des Testobjekts, wird das ATOS-Capturing-Modul darüber informiert und erhält in diesem Zusammenhang ein Handle des Fensters, zu dem das Menü gehört. Dieses Handle wird für eine eventuelle spätere Kommandogenerierung benötigt und bis dahin gespeichert. Im Folgenden erhält das Capturing-Modul immer dann eine Nachricht, wenn ein Menüelement markiert wurde. Das passiert, wenn der Nutzer den Mauszeiger über das Menü bewegt oder mit den Pfeiltasten der Tastatur zum nächsten Eintrag wechselt. Die Nachricht enthält als Parameter ein Handle des Menüs und außerdem die Kennziffer des Menübefehls. Wurde ein Menüelement markiert, das kein Kommando auslöst (ein deaktivierter Menüpunkt, eine Trennlinie oder ein Untermenü), wird statt der Kennziffer der Wert 0 übergeben.

Der Benutzer kann einen Menübefehl auf zwei Arten ausführen: entweder unter Verwendung der Tastatur, indem er mit den Pfeiltasten zum gewünschten Menüpunkt springt und diesen dann mit der `ENTER`-Taste auslöst, oder mit der Maus, indem er den Menüpunkt mit der linken Taste anklickt. Diese Ereignisse müssen somit speziell überwacht werden, solange ein Menü aktiv ist. Die Auswertung der Tastatur-Eingaben erfolgt dabei ohne besondere Einschränkungen. Wird die `ENTER`-Taste gedrückt, nachdem ein Menü initialisiert wurde, und war das zuletzt markierte Menüelement ein ausführbarer Menüpunkt, dann kann direkt ein Testskript-Kommando für die Menü-Aktion generiert werden. Durch Drücken der `ESC`-Taste wird das zuletzt geöffnete Menü der Menühierarchie geschlossen (das letzte ausgeklappte Untermenü), wodurch ein zuvor als markiert gespeicherter Menüpunkt seine Gültigkeit verliert.

Die Behandlung der Mausbedienung gestaltet sich etwas aufwändiger. Wichtig ist aber, dass, egal wie schnell der Nutzer einen Menüpunkt auswählt, dieser immer zuerst markiert und danach ausgeführt wird. Das Capturing-Modul kann sich

also auf die Nachricht im Zusammenhang mit der Markierung des Menüpunkts verlassen.

Im Falle eines Mausklicks zum Zeitpunkt eines aktiven Menüs, muss entschieden werden können, ob dieser Klick einen Menübefehl ausgeführt hat. Das ist immer dann gegeben, wenn die Koordinaten des Mausklicks im Bereich des zuletzt aktiven Menüpunkts liegen und dieser ausführbar war. Wenn das Menüelement kein aktiver Menüpunkt gewesen ist, hat der Mausklick üblicherweise keine Auswirkungen. Befinden sich die Koordinaten außerhalb dieser Fläche (und somit zwangsläufig außerhalb des Menüs, da sonst zuvor die Markierung auf ein anderes Menüelement gewechselt hätte), hat dies das Schließen des Menüs ohne Ausführung eines Befehls zur Folge. Die Fläche des zuletzt markierten Menüelements muss also zum Zeitpunkt eines Mausklicks bereits bekannt sein und wird deshalb im Zusammenhang mit der Verarbeitung der Selektion bestimmt und gespeichert.

Für die Erzeugung eines Testskript-Befehls für ein Menü-Kommando müssen dann die bei früheren Ereignissen gespeicherten Daten zur Verfügung stehen: das Fenster, dessen Menü aufgerufen wurde, und die Kennziffer des Menüeintrags.

6.5.3. Modus TEST

Der *TEST*-Modus dient dazu, den aktuellen Status des Testobjekts mit Testskript-Befehlen zur Zustandsprüfung zu erfassen. Während dieser Modus aktiv ist, kann das Testobjekt nicht gesteuert werden, da alle Eingaben durch die installierten Windows-Hooks blockiert werden. Es ist entscheidend, die Steuerung zu unterbinden, da die Steuerelemente und Fenster, deren Zustände für den Test von Bedeutung sind, durch Anklicken mit der Maus spezifiziert werden. Würden diese Mauseingaben nicht abgefangen, hätten sie Einfluss auf den Zustand des Testobjekts, was nicht erwünscht ist. Als Reaktion auf die Auswahl eines Objekts (z.B. eines Steuerelements) wird jeweils eine Sammlung von möglichen Prüfungen angeboten. Bei einem Kontrollkästchen wären das z.B. der Status (markiert oder nicht), die Beschriftung, die Verfügbarkeit (aktiviert oder deaktiviert) und die Sichtbarkeit. Der zu prüfende Wert wird vom Objekt aktuell ausgelesen und im Testskript-Kommando verwendet, so dass keine weitere Interaktion mit dem Testskript-Designer nötig ist.

Wie bereits in Abschnitt 6.5.2 dargelegt, realisiert das ATOS-Capturing-Modul im Modus *ACTION* einen Mechanismus, um überflüssige Kommandozeilen zu vermeiden, wenn ein Steuerelement erst über mehrere Zwischenschritte seinen

beabsichtigten Zustand erreicht (z.B. den Inhalt eines Eingabefeldes, der Zeichen für Zeichen eingegeben wird). Es kann jedoch auch Fälle geben, in denen auch diese Zwischenschritte für einen Test von Interesse sind, da die Zustandsänderungen des Steuerelements z.B. an anderer Stelle im Testobjekt einen Einfluss haben. Dieses ist aber nur dann zu prüfen, wenn zwischen den Eingaben Test-Kommandos eingefügt werden, wodurch die Zwischenstadien dann auch erhalten bleiben.

6.6. Beziehung zwischen Capturing-Modul und ATOS-Grundsystem

Die Anforderungen an die Entwicklung einer Erweiterung für die Testsuite ATOS umfassten die bruchlose Integration in das bestehende System (vgl. Abschnitt 6.1). Der streng modulare Aufbau war dafür die bestehende Voraussetzung. Auf Programmkode-Ebene teilt sich das Capturing-Modul in die Methoden für die grafische Dialog-Oberfläche und die Funktionalität, die in Form einer Klasse `CapturingProcessor` realisiert ist. Die Klasse stellt eine Reihe öffentlicher Funktionen als Schnittstelle zur Verfügung, die von der Window-Prozedur des Dialogs genutzt wird. Ein Objekt der Klasse `CapturingProcessor` wird dabei von der Hauptklasse `ATOS` des Systems verwaltet (vgl. Abbildung 6.6). Die Funktionalität der Klase umfasst die Hook-Installation und -Steuerung sowie Verarbeitung und Interpretation der Informationen des Capturings und anschließende Kommandogenerierung.

Alle Methoden, die in den Bereich der Hook-Funktionalität fallen, sind in einer DLL (Dynamic Link Library) untergebracht. Das ist auch deshalb sinnvoll, weil so der Zugriff anderer Prozesse auf diese Funktionen direkt gewährleistet ist. Für die Steuerung durch den ATOS-Prozess stellt die Bibliothek Methoden zur Verfügung, die extern gerufen werden können.

Die vom Capturing-Modul generierten Kommandozeilen werden in die aktuelle Testsequenz an der gewünschten Stelle übernommen. Dafür wird auf bereits vorhandene Methoden der Klasse `ATOS` zurückgegriffen.

6.7. Integration des Capturing-Moduls in das ATOS-Grundsystem

6.7.1. Erweiterung von HTS

Eine wichtige Forderung für die Realisierung der Capturing-Erweiterung war, dass die erzeugten Testskript-Kommandos gültige HTS-Befehle darstellen, die sich auch möglichst im ATOS-Befehlseditor bearbeiten lassen (siehe Abschnitt

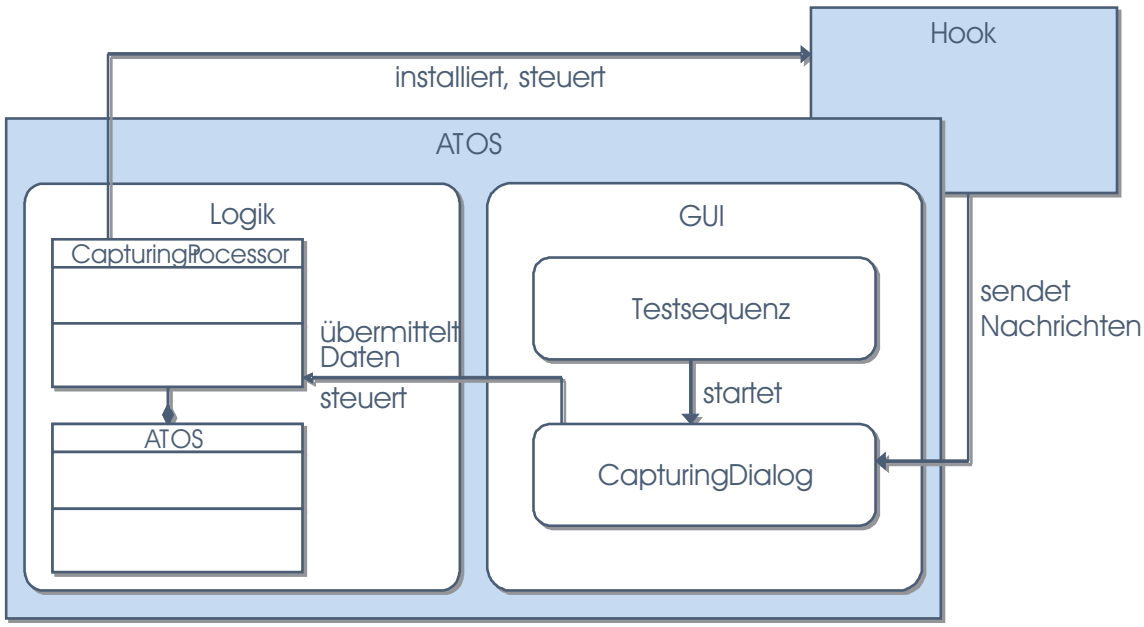

Abbildung 6.6.: Trennung von GUI und Logik

6.1). Hierbei trat jedoch ein Problem im Zusammenhang mit dem Design der HTS-Sprache zutage. Diese war entwickelt worden mit der Vorgabe, dass stets die unter Verwendung des Zusatzprogramms *RCParser* aufgearbeiteten Daten der Ressourcen-Dateien aus dem Quelltext des Testobjekts im URF-Format (Uniform Resource File) zur Verfügung stehen würden. Mit der Ergänzung um die Capturing-Funktionalität erlangt ATOS jedoch das Potenzial, auch solche Programme zu testen, für die keine Quellen verfügbar sind, bzw. kann damit auf das Einlesen und Aufarbeiten der Quellen verzichtet werden. Im Hinblick darauf wäre es nicht wünschenswert, aufgrund der Einschränkungen von HTS doch wieder das Vorhandensein von URF-Daten zu fordern. Somit blieb die Möglichkeit, die HTS-Sprache zu modifizieren, um den neuen Anforderungen zu genügen.

Alle HTS-Befehle, die mit Steuerelementen oder Menüs arbeiten, erwarten dafür die verbalen Bezeichner bzw. Menüpfade (d.h. die Abfolge in der Menühierarchie, die gewählt werden muss, um den entsprechenden Menüpunkt aufzurufen). Während eines Testdurchlaufs wird jede HTS-Befehlszeile zunächst in ATS-Befehle (Atomare Testskript Sprache [18]) übersetzt, wobei diese Parameter mit den jeweiligen Kennziffern (der Steuerelemente oder Menükommandos) substituiert werden.

```
anweisung = aktionsschritt | auswertungsschritt | interaktion |
            dateioperation | loopstruktur | kommentar |
            deakt_kommando | nosubst_kommando
            ...
            nosubst_kommando = NOSUBST,anweisung
```

Abbildung 6.7.: Definition des neuen HTS-Kommandos NOSUBST

Über das Capturing werden jedoch genau diese Kennziffern bereits bestimmt, wobei eine Rückwärtsübersetzung nur bei vorhandenen URF-Daten möglich wäre. Diese findet aber momentan generell nicht statt. Als Modifikation wäre denkbar, für alle Befehle, die mit Steuerelementen oder Menüs arbeiten, in den entsprechenden Parametern alternativ zu den verbalen Bezeichnern oder Menüpfaden auch numerische Kennziffern zu erlauben. Leider hätte das weitreichende Änderungen an der Editor-Komponente von ATOS nach sich gezogen, die im Rahmen dieser Diplomarbeit nicht realisiert werden konnten, da viele grundsätzliche Konzeptänderungen nötig geworden wären, die wiederum weitere Modifikationen bedingt hätten. So wurde als Alternative ein neues HTS-Kommando `NOSUBST` eingeführt (vlg. Listing 6.7). Dieses Kommando dient als Präfix vor einem beliebigen anderen HTS-Befehl und bewirkt bei der Übersetzung nach ATS eine Deaktivierung der Substituierungen. Im ATOS-Befehlseditor besitzt `NOSUBST` einen Parameter vom Typ `string`, der beliebig belegt werden kann. Auf diese Weise findet die automatische Semantik-Prüfung, die auf URF-Daten basiert, nicht statt. Auch die Syntaktik-Prüfung wird so verhindert. Eine Nachbearbeitung der meisten durch das Capturing automatisch erzeugten Befehlszeilen durch den Tester ist also möglich, gestaltet sich jedoch nicht so komfortabel, wie bei Zeilen, die unter Verwendung von URF-Daten manuell erstellt wurden. Die `KEYBOARD`-Kommandos zur Simulation von Tastatur-Eingaben sind unabhängig von URF-Daten und werden somit vom Capturing-Modul nicht mit dem Präfix `NOSUBST` versehen.

Im Rahmen dieser Diplomarbeit wurden noch weitere Ergänzungen an der HTS-Sprache realisiert, um das Potential des Capturings noch besser nutzen zu können. So erlaubte es das Kommando `KEYBOARD` bisher lediglich, eine begrenze Auswahl von Tasten zu simulieren. Ergänzend ist nun auch die Angabe eines beliebigen virtuellen Tastenkodes möglich, wodurch jetzt praktisch jede Tastatureingabe erzeugbar ist. Weitere Ergänzungen betreffen die Möglichkeiten der Kommandos

TEST und READ für die Steuerelemente COMBOBOX, LISTBOX, RADIOBUTTON und CHECKBOX. Außerdem wurde die Unterstützung für Steuerung und Test von Registerfenstern (TAB) hinzugefügt.

6.7.2. Modifikationen am ATOS-Grundsystem

Die bisherige starre Auslegung der Testsuite ATOS auf das Vorhandensein von aufgearbeiteten Ressourcen-Daten des Testobjekts in Form von einer oder mehreren URF-Dateien musste gelockert werden, um die mit Einführung des Capturings erlangten neuen Einsatzmöglichkeiten vollständig nutzen zu können. Um den Test von Applikationen zu unterstützen, für die keine Quellen vorliegen, ist es nun möglich, Projekte ohne die Angabe einer URF-Datei anzulegen. Zudem ist der Titel des Hauptfensters jetzt frei wählbar, während bisher nur eine Liste der Titel der in den URF-Daten als Hauptfenster definierten Fenster zur Auswahl angeboten wurde. Besonders die erste Änderung erforderte eine umfassende Revision der ATOS-Quelltexte, um Probleme an den Stellen zu vermeiden, wo das Vorhandensein von URF-Daten vorausgesetzt wurde.

7. Praxistest des Capturing-Moduls

In diesem Kapitel werden die Ergebnisse des Einsatzes von ATOS unter Verwendung der neuen Capturing-Erweiterung vorgestellt. Neben der Definition und Durchführung von Tests für Softwareprodukte, die ausschließlich in Binärform vorliegen, konnte die Testskript-Erstellung unter Nutzung des Capturings besonders auch im Vergleich zur bisherigen Methode der Testprogrammierung überzeugen.

Die Testsuite ATOS wurde ursprünglich mit dem Ziel entwickelt, die Arbeit am Lehrstuhl Softwaretechnik im Projekt „Software-Sanierung“ ([5], [6]) zu unterstützen. In diesem Projekt wurde und wird ein großes Software-System einem Reengineering[1] unterzogen. Es handelt sich dabei um ein Programm, das am Institut für Physik der Humboldt-Universität zu Berlin in der Arbeitsgruppe „Röntgenbeugung an Schichtsystemen“ unter Prof. Köhler eingesetzt wird, um eine Messanlage zur Untersuchung kristalliner Strukturen von Halbleitern anzusteuern. Die Anwendung wurde ursprünglich von Mitarbeitern der Physik unter Verwendung der Programmiersprache C++ implementiert. Dabei entstand aber über den Verlauf der Entwicklung eine so ungünstige Software-Architektur, dass schließlich eine Weiterentwicklung und Wartung nicht mehr möglich war.

Ziel des Projekts „Software-Sanierung“ war es somit, die Steuerungssoftware aufzuarbeiten und umfangreiche Veränderungen an der Programmstruktur vorzunehmen. Weiterhin wurde die Funktionalität an vielen Stellen verbessert, ergänzt und überarbeitet. Die Aufgaben wurden von vielen Studenten im Rahmen von Studien- oder Diplomarbeiten durchgeführt – besonders in der Anfangsphase auch zeitlich parallel. Der Bedarf nach regelmäßigen Regressionstests im Rahmen des Projekts ergab sich somit schnell und auch der Wunsch nach einer Möglichkeit der Automatisierung, was den Anstoß zur Entwicklung der Testsuite ATOS gab.

Für jeden Programmteil der Steuerungssoftware XCtl wurden dann von den jeweiligen Entwicklergruppen Testfälle aufgestellt, um die korrekte Funktiona-

[1] **Software-Reengineering**: Untersuchung und Modifikation eines Systems, um es in einer neuen Form wiederherzustellen und diese Form nachfolgend zu implementieren (nach Chikofsky/Cross).

lität im Rahmen von Regressionstests jederzeit prüfen zu können. Fehler durch Seiteneffekte von Änderungen an anderen Programmteilen konnten so schnell gefunden und ausgebessert werden.

Für den Praxistest des neu entwickelten Capturing-Moduls für ATOS bot sich die Verwendung des XCtl Systems an. ATOS ist dabei mit einem Testobjekt hoher Komplexität konfrontiert, das sich aber auf die Nutzung von Standard-Steuerelementen stützt und somit in großem Umfang testbar ist. Außerdem konnte durch ein Re-Design von einigen Testfällen ein guter Vergleich zwischen der Erstellung von Testskripts mit und ohne Verwendung der Capturing-Funktionalität gezogen werden. Abgesehen davon wurden einige Standard-Programme des Microsoft® Windows® Systems zum Testen eingesetzt.

7.1. Einsatz mit Standard-Programmen des Betriebssystems

Die Anwendung von ATOS für die Steuerung und Abfrage von Programmen, die zum Lieferumfang von Microsoft® Windows® gehören, ist nur unter Verwendung des Capturing-Moduls möglich, da für diese Applikationen natürlich keine Quelltexte vorliegen.

Als erstes Testobjekt diente das Programm *Rechner*, das stets im Windows-Systemverzeichnis zu finden ist, als `calc.exe`. Die Anwendung realisiert einen „Taschenrechner“ mit zwei Modi. In der Standard-Ansicht beschränkt sich die Funktionalität dabei auf die Grundrechenarten, einen Zwischenspeicher und Funktionen für die Quadratwurzel, Prozentrechnung und das Reziproke. In der wissenschaftlichen Ansicht stehen dagegen weit mehr Funktionen zur Verfügung. Das Programm kann vollständig unter Verwendung der Maus oder via Tastatur bedient werden.

Im Test wurde festgestellt, dass ATOS in der Lage ist, das Programm *Rechner* in vollem Funktionsumfang zu steuern und abzufragen. Neben der Steuerelement-spezifischen Bedienung, die die Verwendung mit der Maus simuliert, ist auch die Überprüfung der Steuerfähigkeit unter Beschränkung auf die Tastatur möglich, indem im Dialog des Capturing-Moduls die Übernahme der reinen Tastatureingaben aktiviert wird.

Das zweite Testobjekt dieser Kategorie stellte das Programm *Editor* dar. Die Binary `notepad.exe` dieses Programms befindet sich direkt im Windows-Stammverzeichnis. Es dient zur Bearbeitung von Textdateien ohne Verwendung von Formatierungen. Eine geladene Datei kann bearbeitet, gespeichert und ge-

druckt werden. Für die Ansicht ist eine Anpassung der verwendeten Schriftart möglich.

Auch für diese Applikation ist ATOS unter Verwendung der Capturing-Erweiterung zur Steuerung und Prüfung einsetzbar. Das Editorfenster besteht hauptsächlich aus einem Standard-Eingabefeld für mehrzeilige Texteingabe – lediglich die Behandlung der Tabulatortaste ist abweichend implementiert – und ist somit zu allen entsprechenden Testskript-Kommandos kompatibel. Über das Menü lassen sich Dialogfenster zur Einstellung verschiedener Optionen aufrufen. Als einzige Einschränkung stellte sich heraus, dass sich die Kontextmenüs[2] dieses Programms nicht erfassen lassen und somit in Testabläufe nicht aufgenommen werden können.

7.2. Einsatz mit der Steuerungssoftware XCtl

Die in Abschnitt 7.1 beschriebenen Tests wurden im Verlauf der Entwicklung des Capturing-Moduls für ATOS immer wieder durchgeführt, um dessen Funktionstüchtigkeit zu verifizieren. Nach Abschluss der Arbeiten sollte das Resultat nun an einem völlig neuen Testobjekt erprobt werden, wofür sich aufgrund der bereits beschriebenen Komplexität und Kompatibilität die Steuerungssoftware XCtl anbot. Hilfreich waren zudem die bereits detailliert spezifiziert vorliegenden Testfälle, von denen einige im Rahmen des Praxistests neu implementiert wurden, einerseits unter Verwendung der Capturing-Funktionalität, andererseits mit Beschränkung auf die bisherige Methode der Testskript-Erstellung mit Hilfe des integrierten Editors. So war neben einer Einschätzung der erreichten Abdeckung von benötigten Testaufgaben mit der Capturing-Erweiterung auch ein Vergleich der Performance beim Einsatz beider Vorgehensweisen möglich.

Ausgewählt wurden fünf Testfälle, die sich mit jeweils anderen Programmteilen des XCtl-Systems beschäftigen. So konnte die Kompatibilität des Capturing-Moduls mit dem Testobjekt umfassend getestet werden. Die verbalen Spezifikationen der verwendeten Testfälle finden sich im Anhang (A.1.1 bis A.1.5).

Das System wird hauptsächlich über unterschiedlich komplexe Dialog-Fenster gesteuert, die vorwiegend über die Menüleiste aufgerufen werden. Daneben

[2] **Kontextmenü**: Es handelt es sich um ein Menü, das zum gerade gewählten Objekt die im jeweiligen Zusammenhang relevanten Befehle enthält. Im Unterschied zu einem gewöhnlichen Menü aus der Menüzeile, wird das Kontextmenü an der Stelle angezeigt, an der sich das Objekt befindet. Unter Microsoft® Windows® wird das Kontextmenü üblicherweise mit einem Rechtsklick aufgerufen.

Testfall	**Zeitaufwand für die Erstellung**		**Zeitersparnis**
	ohne Capturing	mit Capturing	
AE.1	9:10	3:09	66%
AS.1	26:11	12:56	51%
ARS.2	16:48	7:39	54%
LS.2	10:04	4:24	56%
MS.1	13:21	5:20	61%
		Durchschnittliche Zeitersparnis:	57,6%

Tabelle 7.1.: Performance bei der Testskript-Erstellung mit Capturing

erforderten die Testfälle den Umgang mit Öffnen- und Speichern-Dialogen der Windows-Standard-Bibliothek (Common Dialog Box Library). Außerdem wurden einige Dialoge über Kontextmenüs aufgerufen, was hier auch problemlos vom Capturing-Modul ausgewertet werden konnte.

Es ergab sich ein durchweg positives Bild bzgl. des Einsatzes der Capturing-Erweiterung für den Regressionstest des XCtl-Systems. Die Testanforderungen konnten vollständig abgedeckt werden, d.h. die Erstellung der Testfälle wäre auch möglich gewesen, wenn keine Quelltexte des Programms vorgelegen hätten. Manuell eingefügte Testskript-Kommandos waren lediglich für die folgenden Anforderungen nötig.

Jeder Testfall wird eingeleitet durch die Herstellung eines definierten Anfangszustands des XCtl-Systems, indem speziell vorbereitete Konfigurationsdateien in das Programmverzeichnis kopiert werden. Nach dem Ende des Testdurchlaufs wird der alte Zustand wiederhergestellt durch Rückkopieren der gesicherten Originaldateien. Diese Dateioperationen müssen in jedem Fall manuell eingegeben werden.

An verschiedenen Stellen der Testfälle sind Wartezeiten spezifiziert, z.B. um einen Initialisierungsvorgang oder eine Messung abzuwarten. In der aktuellen Version des Capturing-Moduls wird dafür die Aufzeichnung unterbrochen, das Kommando für die Pause der erforderlichen Dauer manuell eingefügt und anschließend das Capturing an der alten Stelle fortgesetzt. Die Implementation einer Funktion zur Aufnahme von Wartezeiten erscheint vor diesem Hintergrund sinnvoll, allerdings steht die Frage, ob sich die gehäufte Verwendung von Wartezeiten u.U. nur speziell bei dem XCtl-System findet.

Bei einigen Testfällen wird am Ende ein externes Programm für den Dateivergleich ausgeführt, um während des Testlaufs erzeugte Ausgaben gegen Referenzdateien zu prüfen. Dies und der Startbefehl für das Testobjekt selbst müssen per Hand programmiert werden.

Im Ergebnis kann aber der weitaus größte Teil der erforderlichen Testskript-Kommandos automatisch erzeugt werden, was eine erhebliche Zeitersparnis von deutlich über 50% erbrachte (vgl. Tabelle 7.1). Dazu kommt, dass durch die direkte Arbeit am Testobjekt Fehler vermieden werden, wie beispielsweise vergessene Steuerbefehle. Dies kann erfahrungsgemäß durch die eingeschränkte Übersicht bei der manuellen Eingabe leicht passieren und trat bei der Erstellung dieser Testskripte in zwei Fällen auf. Bei länger andauernder Arbeit ist jedoch mit einem Anstieg der Fehlerrate zu rechnen, während das Capturing die Aufmerksamkeit des Testskript-Designers weit weniger beansprucht.

8. Fazit und Ausblick

In diesem Kapitel sollen die erreichten Ergebnisse im Rahmen dieser Diplomarbeit zusammengefasst werden. Außerdem werden Unterschiede und Gemeinsamkeiten von ATOS und dem kommerziellen Produkt *WinRunner*® [21] gegenübergestellt. Abschließend findet eine Betrachtung von noch vorhandenen Einschränkungen und möglichen zukünftigen Weiterentwicklungen statt.

8.1. Erreichte Ziele

Die in Abschnitt 6.1 definierten Anforderungen an die Capturing-Erweiterung für die ATOS-Testsuite konnten im Rahmen der Arbeiten für diese Diplomarbeit vollständig erfüllt werden.

Die Verwendung von ATOS im Zusammenhang mit bestehenden Projekten ist weiterhin ohne Einschränkungen oder Modifikationen möglich. Zusätzlich lässt sich ab sofort auch in diesen Projekten das Capturing für die Testskript-Erstellung eingesetzen. Das Capturing-Modul konnte bruchlos in das bestehende Rahmenwerk des Systems eingegliedert werden und ist auch deshalb intuitiv benutzbar. Jede Testsequenz kann beliebig aus manuell eingegebenen und automatisch über das Capturing generierten Kommandos aufgebaut sein, wobei auch eine manuelle Nachbearbeitung der automatisch erzeugten Zeilen möglich ist. Die Funktionalität umfasst die Anforderungen häufiger Testaufgaben, so dass die Erstellung von Standard-Testszenarien großflächig abgedeckt wird.

8.2. Vergleich ATOS und WinRunner

8.2.1. Aufbau einer Datenbasis über die GUI des Testobjekts

WinRunner: Es erfolgt ein weitgehend automatisches Aufrufen und Einlesen aller Fenster des laufenden Testobjekts. Probleme werden vom Anwender direkt mit Hilfe eines Wizards gelöst. Die Datenbasis findet für die automatische und manuelle Testskript-Erzeugung Verwendung.

ATOS: Wenn vorhanden, können die Ressourcen-Dateien der Quelltexte des Testobjekts automatisch geparst werden. Inkonsistenzen der erstellten Datenbasis sind nur manuell zu beseitigen, werden jedoch angezeigt. Die Datenbasis wird für die manuelle Testskript-Programmierung verwendet.

WinRunner gestaltet die Erzeugung der Datenbasis benutzerfreundlicher und intuitiver. Außerdem ist WinRunner dabei unabhängig vom Quellcode des Testobjekts. ATOS ist allerdings im Gegensatz zu WinRunner auch ohne Datenbasis einsatzfähig, wenn die Testskript-Programmierung automatisch vorgenommen wird.

8.2.2. Testskript-Aufzeichnung

WinRunner: Es stehen zwei Modi zur Verfügung zwischen denen vor Beginn gewählt werden muss. Die kontextsensitive Aufzeichnung generiert Steuerelement-spezifische Testskript-Befehle, im analogen Modus werden reine Maus- und Tastatureingaben übernommen, ohne Bezug auf die GUI des Testobjekts.

ATOS: Das Capturing-Modul generiert aus den mitgelesenen Nutzereingaben (Maus und Tastatur) Steuerelement-spezifische Testskript-Kommandos. Zusätzlich ist jederzeit ein Wechsel in einen Modus möglich, in dem die Tastatureingaben direkt, ohne Berücksichtigung der GUI des Testobjekts, übernommen werden.

WinRunner ist im Gegensatz zu ATOS auch in der Lage, Mauseingaben analog aufzuzeichnen, d.h. es werden für jeden Mausklick die absoluten Bildschirmkoordinaten und die genaue Dauer des Tastendrucks gespeichert. Anwendungsszenarien für diese Funktion sind allerdings sehr selten. Der Vorteil von ATOS besteht darin, dass der Aufzeichnungsmodus jederzeit während der Aufnahme gewechselt werden kann.

8.2.3. Manuelle Testskript-Programmierung

WinRunner: Es steht eine hochentwickelte Testskript-Sprache zur Verfügung, die alle üblichen Konstrukte von höheren Programmiersprachen (Variablen, Schleifen, Bedingungsanweisungen, Funktionen usw.) unterstützt. Die verwendete Syntax ist der Programmiersprache C nachempfunden. Als Variablentypen stehen Zeichenketten, Zahlen verschiedener Genauigkeit und Felder zur Verfügung. Neben speziellen Funktionen für die Testdefinition kann eine umfassende Bi-

bliothek von Standardfunktionen (z.B. Arithmetik, Zeichenkettenmanipulation, Ein-/Ausgabe) verwendet werden.

ATOS: Testskripte werden unter Verwendung einer einfachen Skriptsprache angelegt. Es stehen 21 Kommandos zur Verfügung, um das Testobjekt über die unterstützten Steuerelemente, Fenster und Menüs zu steuern und dessen Status zu testen. Daneben sind u.a. Dateioperationen, einfache Schleifen und Interaktionen mit dem Nutzer möglich.

Die Testskript-Sprache von WinRunner erlaubt es, hoch komplexe Testskripte zu entwickeln und den Testablauf in jeder Hinsicht zu beeinflussen. Für diese Arbeiten sind dann allerdings auch fortgeschrittene Programmierkenntnisse erforderlich und mit steigender Komplexität nimmt die Wartbarkeit der Testskripte ab. Die deutliche Mehrheit der Testszenarien wird aber auch schon von dem Befehlssatz, den ATOS anbietet, voll abgedeckt.

8.2.4. Testobjektstart und –vorbereitung

WinRunner: Vor einem Testdurchlauf muss der Anwender das Testobjekt starten und in den gewünschten Anfangszustand bringen.

ATOS: Vorbereitung und Start des Testobjekts sind Teil des Testskripts und müssen nicht vom Anwender durchgeführt werden.

Die hochentwickelte Testskript-Sprache TSL des WinRunners erlaubt es erfahrenen Programmierern, alle vorbereitenden Schritte für einen Testdurchlauf und auch das Starten des Testobjekts automatisiert durchzuführen. Diese Aufgaben lassen sich mit den HTS-Befehlen von ATOS jedoch wesentlich unkomplizierter realisieren.

8.2.5. Vermeidung von Synchronisationsproblemen

WinRunner: An so genannten Checkpunkten und bei kontextsensitiven Aktionen wird in der Testdurchführung bei Bedarf bis zu einem festgelegten Timeout (Standard: 10s) gewartet. Der Wert kann global angepasst werden, beeinflusst dann aber auch die gesamte Ablaufgeschwindigkeit. Alternativ lässt sich an Stellen im Testskript, wo das normale Timeout nicht ausreicht, manuell ein Synchronisationspunkt einfügen, der die Abarbeitung anhält, bis im Testobjekt ein bestimmter Zustand eingetreten ist.

ATOS: Nach der Ausführung jeden Testschritts wird immer eine bestimmte Anzahl von Millisekunden gewartet, bevor mit dem nächsten Schritt fortgefahren wird. Diese Dauer kann bei jedem Testdurchlauf individuell eingestellt werden (Standard: 100ms). Zusätzlich steht ein Testskript-Kommando zur Verfügung, um die Abarbeitung eines Tests an ausgewählten Stellen für eine beliebige Zeit zu unterbrechen.

Beide Programme verfolgen bei der Behandlung von Synchronisationsproblemen einen ähnlichen Ansatz, wobei ATOS die Möglichkeit fehlt, auf das Eintreten eines bestimmten Zustands des Testobjekts zu warten. Diese Funktionalität ist jedoch in vielen Fällen besser geeignet als das Warten für eine bestimmte Zeitspanne, z.B. wenn die Reaktionszeit des Testobjekts von der Leistungsfähigkeit des verwendeten Rechners abhängt und die Tests auf verschiedenen Maschinen durchgeführt werden sollen.

8.2.6. Abtesten von Steuerelementen

WinRunner: Während der Aufzeichnung eines Testfalls können GUI Checkpunkte eingefügt werden. Dazu wählt der Benutzer die Funktion aus dem Menü oder der Toolbar und klickt anschließend das gewünschte Steuerelement mit der Maus an (einfach oder doppelt). Daraufhin wird entweder direkt ein für das Objekt üblicher Statustest in das Skript eingefügt oder eine Liste möglicher Tests angeboten. Um weitere Steuerelemente zu prüfen, muss die Funktion erneut ausgewählt werden.

ATOS: Das Capturing kann in zwei verschiedenen Aufzeichnungsmodi durchgeführt werden. Im ersten Fall erzeugt das Capturing-Modul Testskript-Kommandos zur Steuerung des Testobjekts. Jederzeit während des Capturings (oder auch vor Beginn) kann der Anwender in den zweiten Modus umgeschalten. Dann werden alle Eingaben für das Testobjekt blockiert, um es in dem aktuellen Status zu halten. Die Steuerelemente und Dialogfenster lassen sich jetzt anklicken, um eine Liste möglicher Prüfungen für das spezifische Objekt zu erhalten. Der am häufigsten verwendete Test steht dabei vorselektiert am Kopf der Zusammenstellung.

Das Einfügen von Kommandos für die Überprüfung des Zustands der Steuerelemente des Testobjekts ist in ATOS intuitiver und einfacher möglich. Das ist ein wichtiger Vorteil von ATOS, da diese Testskript-Befehle stets einen großen Teil eines GUI-Testskripts ausmachen.

8.2.7. Bildvergleich

WinRunner: Während der Testskript-Aufzeichnung können Bild-Checkpunkte definiert werden. An diesen Stellen werden dann Fenster, Steuerelemente oder andere Bereiche des Testobjekts abfotografiert und diese Bilder mit gespeicherten Referenzgrafiken während des Testdurchlaufs verglichen.

ATOS: Diese Funktionalität wird nicht unterstützt.

Pixelweise Bildvergleiche sind nur in einer geringen Anzahl von Test-Szenarien sinnvoll einsetzbar. In den Fällen, wo Grafiken verglichen werden müssen (z.B. Verlaufkurven), sind meist bestimmte Abweichungen von der Vorlage erlaubt, wodurch ein trivialer Test auf Identität zweier Bilder, wie von WinRunner unterstützt, nicht ausreicht.

8.2.8. Kombinierte Testdurchläufe

WinRunner: Um mehrere Tests sequenziell durchzuführen, können Batch-Tests (Stapelverarbeitung) definiert werden.

ATOS: Testsequenzen können in Testpaketen zusammengefasst und auf diese Weise ohne Nutzereingriff nacheinander abgearbeitet werden. Eine Testsequenz kann dabei gleichzeitig unterschiedlichen Testpaketen zugeordnet sein.

Beide Programme bieten in dieser Hinsicht etwa gleichwertige Funktionalität.

8.2.9. Testlaufauswertung

WinRunner: Die Ergebnisse einer Testdurchführung stehen direkt im Programm zur Verfügung und können über einen Dialog eingesehen werden. Im Fehlerfall wird, soweit möglich, der aktuelle Testdurchlauf fortgesetzt.

ATOS: Jeder Testschritt wird in einer Logdatei dokumentiert – im Fehlerfall mit der entsprechenden Meldung. Am Ende jedes Testdurchlaufs erhält der Anwender einen Hinweis, ob die Abarbeitung erfolgreich war. Sollte während eines Tests ein Testskript-Kommando nicht erfolgreich sein, so wird der Ablauf abgebrochen. Ggf. wird mit der nächsten Testsequenz fortgefahren, wenn ein Testpaket ausgeführt wird.

Die Analyse der Testergebnisse gestaltet sich bei der Verwendung von WinRunner

komfortabler und direkter, so ist z.B. auch ein Vergleich verschiedener Testdurchläufe übersichtlich möglich. Zudem kann die Möglichkeit, den Testdurchlauf im Fehlerfall mit dem nächsten Schritt fortzusetzen, von Vorteil sein, wenn es sich um einen zeitaufwändigen Testfall handelt, in dem relativ spät ein geringfügiger Fehler auftritt.

8.2.10. Wartung der Testskripte

WinRunner: Nach Änderungen an der GUI des Testobjekts kann die Datenbasis automatisch oder manuell angepasst werden. Die Überarbeitung der Testskripte erfolgt manuell.

ATOS: Wenn eine GUI-Datenbasis verwendet wird, können Modifikationen an der grafischen Benutzungsoberfläche des Testobjekts durch erneutes Einlesen der Ressourcen-Dateien übernommen werden. Dadurch ungültig gewordene Testskript-Zeilen werden als fehlerhaft markiert und sind manuell nachzubearbeiten. Testskripte, die unter Verwendung des Capturings erstellt wurden, müssen manuell angepasst oder (teilweise) neu aufgezeichnet werden.

Für die Übernahme von Änderungen am Testobjekt, die eine Anpassung der Testskripte nötig machen, bietet WinRunner eine deutlich komfortablere Funktionalität als ATOS. Besonders nachteilig wirkt sich aus, dass ATOS für via Capturing aufgezeichnete Testskripte keine Unterstützung bei der Überarbeitung anbieten kann. Es genügt dann jedoch die erneute Aufzeichnung der Stellen des Testfalls, die von den Änderungen betroffen sind.

8.2.11. Zusammenfassung und Schlussfolgerungen

Das kommerzielle Testtool WinRunner® [21] der Firma Mercury™ kann sich gegenüber der im universitären Rahmen entwickelten Lösung ATOS besonders im Hinblick auf die Benutzerfreundlichkeit absetzen. Auch die Einsatzmöglichkeiten sind erwartungsgemäß deutlich höher, während ATOS bislang nur die grundlegendsten Windows-Steuerelemente unterstützt. Mit seiner hoch entwickelten Skriptsprache TSL bietet WinRunner dem Testskript-Programmierer auch die Möglichkeit, sehr spezialisierte Lösungen für den Einsatz in komplexen Szenarien zu erstellen. Die komfortablere Wartung der Testskripte bei WinRunner kann ihre Vorteile aber erst beim Einsatz für solche Testobjekte zeigen, die durch eine sich häufig stark ändernde grafische Benutzungsoberfläche gekennzeichnet sind.

Im Vergleich der technischen Herangehensweise zur Realisierung von GUI-Tests und des angebotenen Umfangs der Steuerungs- und Testmöglichkeiten der

unterstützten Steuerelemente kann ATOS mit dem kommerziellen Werkzeug allerdings ohne Abstriche mithalten. Die automatische Testskript-Erstellung zeigt sich an einigen Stellen sogar effizienter und intuitiver. Auch die generelle Ausrichtung auf den Einsatz für unbeaufsichtigte Testläufe ist bei ATOS besser realisiert, wobei allerdings WinRunner im Bereich der Behandlung und Auswertung von Fehlerfällen deutlich mehr Stärken hat, was für die spätere Auswertung der Tests von Bedeutung ist.

Abschließend lässt sich sagen, dass ATOS für die unterstützten Testszenarien im Allgemeinen durchaus gleichwertig einsetzbar ist wie das kommerzielle Konkurrenzprodukt WinRunner. Hier kann sich ATOS dann durch seinen Preis von WinRunner mit seinen außerordentlich hohen Lizenzkosten absetzen.

8.3. Ausblick

Für die Testsuite ATOS im Allgemeinen und das Capturing-Modul im Speziellen zeigt sich Bedarf an weiterführenden Arbeiten, um die Funktionalität auszubauen und zu verbessern. Einige wichtige Punkte dafür sollen im Folgenden angesprochen werden.

8.3.1. Unterstützung weiterer Steuerelemente

Bislang arbeitet ATOS mit allen gängigen Steuerelementen des Windows-API, die sich über Windows-Nachrichten unter Nutzung der Nachrichtenparameter mit Standard-Datentypen ansteuern und abfragen lassen (siehe Tabelle 6.5.1). Das Windows-API stellt jedoch noch eine Reihe weiterer Steuerelemente zur Verfügung, die ebenfalls häufig eingesetzt werden, die jedoch wesentlich komplexer anzusprechen sind. Meist muss den Windows-Nachrichten ein mit bestimmten Werten vorbelegter Parameter eines für jedes Steuerelement spezifischen `struct`-Datentyps mitgegeben werden, was durch die Art und Weise, wie ATOS HTS-Kommandos umsetzt, Probleme bereitet. Die HTS-Kommandoabarbeitung läuft über eine Zwischenübersetzung in Befehle einer weiteren systemnäheren Testskript-Sprache ATS [18], deren Befehlszeilen dann schließlich interpretiert und ausgeführt werden. Die Leistungsfähigkeit dieser niederen Sprache müsste für die neuen Anforderungen erweitert werden und dementsprechend alle Komponenten, die mit ATS arbeiten. Somit wäre eine gesamte Schicht des ATOS-Systems zu überarbeiten.

8.3.2. Nutzung von vorhandenen URF-Daten durch das Capturing-Modul

In den meisten Fällen, in denen ATOS Einsatz findet, wird es um das Testen von Software-Produkten gehen, die sich in der Entwicklung befinden und für die somit die Quelltexte zur Verfügung stehen. Der Aufbau einer GUI-Datenbasis mit Hilfe des Zusatzprogramms *RCParser* ist dann also in der Regel möglich. Wünschenswert wäre es deshalb, wenn das Capturing-Modul vor der Generierung von Kommandos eventuell vorhandene URF Daten abfragt, um gegebenenfalls den verbalen Bezeichner bzw. Menüpfad für das zu behandelnde Steuerelement bzw. Menükommando zu ermitteln. Wenn dies gelingt, könnte eine HTS-Befehlszeile ohne `NOSUBST` Präfix erstellt werden, die für den Anwender später leichter les- und editierbar wäre.

8.3.3. Übertragung von ATOS und der Capturing-Funktionalität auf Java

Java erfährt inzwischen auch als Implementierungssprache von Anwendungen immer stärkere Verbreitung und konnte an vielen Stellen C++ schon ablösen. Somit steigt auch der Bedarf an Werkzeugen für den Software-Test, die Java unterstützen. ATOS könnte deshalb stark davon profitieren, die Möglichkeit von GUI-Tests von Java-Anwendungen anzubieten.

Java-GUIs

Für die Realisierung von grafischen Benutzungsoberflächen von Java-Programmen kommen verschiedene Bibliotheken zum Einsatz, wobei nahezu alle grafischen Java-Anwendungen auf einer von den drei wichtigsten Bibliotheken basieren.

Seit seiner ersten Version steht in Java das plattformunabhängige Abstract Window Toolkit (AWT) zur Verfügung. Unterstützt werden damit nur die grundlegendsten Steuerelemente, wie sie etwa auch durch HTML auf Internetseiten Verwendung finden. Zum Zeichnen dieser Steuerelemente werden immer die Funktionen des gerade verwendeten Betriebssystems bzw. Fenstermanagers genutzt, wodurch das Java-Programm stets das typische Aussehen, passend zur jeweiligen Zielplattform, erhält. Die Funktionalität jedes Steuerelements ist in einer Klasse gekapselt, damit unabhängig von der Zielplattform mit einheitlichen Interfaces gearbeitet werden kann. Das AWT war allerdings für die Realisierung der einfachen Oberflächen von Applets entwickelt worden und ist mit der Darstellung von komplexen GUIs für Desktop-Anwendungen überfordert, auch wenn es inzwischen noch etwas erweitert wurde und jetzt z.B. immerhin Ereignisverarbeitung unterstützt [7].

Für Java-Programme, die grafische Benutzungsoberflächen realisieren sollen, die über die Möglichkeiten von AWT hinausgehen, wurde von SUN deshalb ein neues Framework namens Swing entwickelt. Hierüber wird eine breite Palette von Steuerelementen bereitgestellt, angefangen bei den bereits im AWT enthaltenen Basistypen bis hin zu sehr komplexen Objekten, wie etwa einem Feld zur Texteingabe mit Formatierungen. Ein wichtiger Unterschied zu AWT ist aber auch, dass Swing das Zeichnen der Steuerelemente nicht dem jeweiligen Betriebssystem bzw. Fenstermanager überlässt, sondern die Darstellung nur unter Verwendung grundlegender Zeichenfunktionen selbst realisiert. Aussehen und Bedienung sind deshalb variabel und werden über ein Look&Feel-Plug-In bestimmt, wodurch das Erscheinungsbild der Anwendung passend zur Zielplattform oder in einem anderen Stil angepasst werden kann. Das Aussehen der GUI des Betriebssystems oder Fenstermanagers kann natürlich nicht perfekt nachgeahmt werden und anwenderspezifische Anpassungen haben deshalb auf das Erscheinungsbild von Swing-Anwendungen meist keinen Einfluss [7].

Die jüngste der drei wichtigsten GUI-Bibliotheken für Java ist das Standard Widget Toolkit (SWT), das von IBM für seine Entwicklungsumgebung Eclipse entwickelt wurde. Das Toolkit ist allerdings auch für Anwendungen außerhalb von Eclipse einsetzbar. Ebenso wie das AWT überlässt das SWT der Zielplattform das Zeichnen der grafischen Oberfläche, wann immer das möglich ist. Anders als das AWT realisiert es aber auch komplexe Steuerelemente, die nicht von allen Betriebssystemen bzw. Fenstermanagern unterstützt werden. Dort, wo diese nicht verfügbar sind, werden sie von SWT wie bei Swing unter Verwendung von einfachen Grafikfunktionen gezeichnet. Zusätzlich verwendet SWT die integrierten GUI-Funktionen des jeweiligen Zielsystems und erlaubt den Zugriff darauf, wodurch eine Java-Anwendung mit Nutzung des SWT nicht nur immer das passende Erscheinungsbild zeigt, sondern sich auch wie ein normales Programm der Zielplattform verhält und entsprechend mit anderen Anwendungen interagieren kann [7].

Technische Realisierung von GUI-basierten Tests für Swing und SWT

Java-Programme, die das SWT für ihre GUI verwenden, sollten sich wie Programme, die speziell für die jeweilige Zielplattform entwickelt wurden, verhalten. Insofern ist die bestehende ATOS-Version möglicherweise schon mit wenigen Modifikationen für den Test von SWT-basierten Java-Anwendungen unter Windows einsetzbar.

Für die Unterstützung von Java-Programmen, die mit Swing arbeiten, ist davon auszugehen, dass in ATOS wesentlich mehr Anpassungen vorgenommen werden müssen. So ist bereits die ATS-Schicht stark Windows-spezifisch, ebenso wie alle tieferen Schichten. Somit ist es nötig, die Umsetzung der HTS-Kommandos für Swing von Grund auf neu zu entwickeln. Auch die Capturing-Erweiterung ist für Swing-basierte Java-Programme nicht verwendbar und müsste neu implementiert werden. Grundsätzlich sind Oberflächentests nach dem Vorbild für Windows-Programme auch für Java-Anwendungen mit Swing-GUIs möglich, wie z.B. das Programm Marathon [8] beweist. Allerdings kann es sein, dass die verwendete Technik nur in einem Java-Programm selbst realisierbar ist. Wenn das zutrifft, müsste ATOS entweder unter Verwendung von Java neu komplett implementiert werden oder es wird ein kleines, externes Java-Programm als Controller verwendet, das von ATOS unter Nutzung einer Schnittstelle, wie CORBA, gesteuert werden könnte.

Aufgrund der seltenen Verwendung des AWT bei Desktop-Anwendungen, lohnt es sich nicht, diese Technik bei der Anpassung von ATOS zu berücksichtigen.

ATOS-Weiterentwicklung in Java oder C++

Trotz der umfangreichen Neuimplementierungen von einigen Teilen des ATOS-Systems, die zur Realisierung einer Version mit Unterstützung für GUI-Tests von Java-Applikationen nötig sind, könnten auch viele Komponenten weiter genutzt werden. Solche Bestandteile sind z.B. das gesamte Projekt-Management, der HTS-Editor, die Dateiformate und deren Verwendung sowie die grafische Benutzungsoberfläche. In verschiedenen Hilfsklassen ist zudem Funktionalität implementiert, die ebenfalls für die Weiterentwicklung nützlich sein kann. Die Möglichkeit der Nachnutzung setzt aber voraus, dass C++ als Programmiersprache für ATOS beibehalten wird.

Alternativ wäre denkbar, ATOS unter Verwendung von Java vollständig neu zu implementieren. Erreichbar wäre damit eine Plattformunabhängigkeit, was die generelle Lauffähigkeit von ATOS betrifft. Für den erfolgreichen Einsatz auf einer bestimmten Zielplattform werden allerdings jeweils Anpassungen nötig sein, die im ungünstigsten Fall die Neuimplementierung aller Schichten unterhalb von HTS bedeuten. Lediglich Java-Anwendungen, die auf das Swing-Toolkit für ihre GUI zurückgreifen, sollten sich auf jeder Plattform erfolgreich testen lassen. Für alle anderen Testobjekte wird die Nutzung von integrierten Systemfunktionen des jeweiligen Betriebssystems oder Fenstermanagers unerlässlich sein. Können diese Funktionen, für Microsoft Windows z.B. die Funktionen des Windows-API,

nicht direkt aus einem Java-Programm heraus aufrufbar sein, müsste für jede Zielplattform jeweils ein Controller entwickelt werden, der z.B. über CORBA angesteuert werden kann. Natürlich bedeutet eine solche Lösung einen hohen Performanceverlust gegenüber direkten Funktionsaufrufen.

8.3.4. Einschätzung der in ATOS realisierten Konzepte

Für den Fall einer Neuimplementation des ATOS-Systems, aber auch generell, lohnt sich eine kritische Betrachtung der Konzepte, mit denen ATOS arbeitet, im Hinblick auf bisher erlangte Erfahrungswerte. Während hier bei einer Neuentwicklung für die Fälle, die sich nicht bewährt haben, sicher neue Wege beschritten werden sollten, kann vielleicht sogar eine Überarbeitung des bestehenden Systems an diesen Stellen sinnvoll sein.

Projektmanagement

Die in ATOS realisierten Möglichkeiten der Projektdefinition, –verwaltung und –organisation haben sich bewährt und sollten grundsätzlich in der vorhandenen Form beibehalten werden.

Testskript-Erstellung und –Verarbeitung

ATOS definiert eine eigene Skriptsprache HTS [15] zur Testfall-Beschreibung und enthält entsprechend einen Parser und Interpreter, um die Skript-Kommandos auszuführen. Dieser Ansatz ist bei ähnlichen Programmen sehr weit verbreitet und kann somit als Standard-Konzept angesehen und beibehalten werden. Eine alternative Herangehensweise wird später noch dargestellt.

Zweischichtige Testskript-Verarbeitung Zusätzlich zu der Skriptsprache für die Programmierung der Testfälle, definiert ATOS noch eine weitere Sprache ATS [18] auf einem niedrigeren Level. Die Befehle auf dieser Ebene sind schon sehr systemnah und damit plattformabhängig. Jedes Kommando der HTS wird mittels einer definierten Übersetzungsregel auf mehrere ATS-Befehle abgebildet, dabei existieren eigene Übersetzungsregeln für alle Steuerelemente, da diese jeweils unterschiedlich angesteuert werden müssen.

Die Abarbeitung eines HTS-Testskripts geschieht somit in mehreren Stufen. Eine HTS-Kommandozeile wird zunächst eingelesen und geparst. Anschließend wird eine passende Übersetzungsregel gesucht, mit Hilfe derer ein kurzes ATS-Skript generiert wird. Dieses ATS-Skript wird nun wiederum geparst und schließ-

lich interpretiert, womit das HTS-Kommando ausgeführt wird. Diese mehrstufige Abarbeitung ist natürlich wesentlich teurer, in Hinsicht auf benötigte Rechenzeit und Speicher, als wenn der Zwischenschritt über die niederlevelige Sprache eingespart würde.

Mit Hilfe der ATS-Schicht sollte eine Möglichkeit geschaffen werden, nachträglich die Unterstützung für weitere Steuerelemente einzurichten, ohne den Quellcode von ATOS bearbeiten zu müssen. So ist es dem Anwender möglich, die Behandlung eines Steuerelements, das ATOS von Haus aus nicht kennt, selbständig zu implementieren. Er muss dazu zunächst die HTS erweitern, indem er Regeln für gültige Befehle mit dem neuen Steuerelement erstellt. Danach müssen diese neuen Befehle mit ATS implementiert werden. Diese Aufgaben erfordern allerdings ein fortgeschrittenes Verständnis von Software- und speziell systemnaher Windows-Programmierung sowie der Funktionsweise eines Parsers nach dem Prinzip einer Zustandsmaschine.

Leider können mittels ATS komplexere Steuerelemente nicht unterstützt werden. Das ist darin begründet, dass in diesen Fällen für den Datenaustausch zwei 32Bit-Werte (als Parameter von SendMessage) nicht ausreichen. Als Konsequenz werden Daten-Strukturen definiert, in denen sich beliebig komplexe Informationen darstellen lassen. Übertragen werden Zeiger auf Objekte des jeweiligen Struktur-Datentyps, wofür die 32Bit ausreichen und wodurch sowohl Informationsübertragung als auch -empfang möglich sind.

Da die Struktur-Datentypen für jedes Steuerelement unterschiedlich sind, kann ATS nicht ohne weiteres entsprechend erweitert werden, um komplexere Steuerelemente zu unterstützen. Die Realisierung eines generischen Ansatzes für das Problem wird einerseits mit relativ großem Aufwand verbunden sein und andererseits die Anwendung von ATS noch anspruchsvoller und weniger intuitiv machen. Insofern stellt sich die Frage, ob die Zwischenschicht ATS nicht generell aufgegeben und statt dessen eine andere Möglichkeit entwickelt werden sollte, um dem Anwender die Anpassung von ATOS für neue Steuerelemente zu ermöglichen. Denkbar wäre z.B. der Ansatz, ein Interface für Steuerelement-Handler zu definieren, das von ActiveX/COM-Komponenten implementiert werden kann. Anwender von ATOS könnten dann ihre eigenen Handler als Dynamic Link Libraries (DLLs) erstellen und bei ATOS registrieren.

Nutzung einer externen Skriptsprache statt Eigenentwicklung (HTS) Einen ganz neuen Ansatz, mit verschiedenen Vorteilen, möchte ich im Folgenden vorstellen. Als Alternative dazu, eine eigene Testskript-Sprache zu definieren, kann auch eine bereits etablierte Skriptsprache zu diesem Zweck eingesetzt werden.

Verwendbar ist dabei jede Sprache, die Module enthält, um die Systemfunktionen eines Betriebssystems auch direkt ansprechen zu können. Im Hinblick auf Windows kämen z.B. Perl [28] oder Python [26] in Frage, die beide Zugriff auf das Win32 API bieten.

Die Idee besteht darin, alle bisherigen Testskript-Kommandos in Form von Funktionen in der Skriptsprache zu implementieren. Als Resultat könnte ein Testskript dann auch wie gehabt, einfach aus einer Sequenz von Kommandos – also Funktionsaufrufen – bestehen. Hinzu kommt aber, dass der Anwender jetzt neben den speziellen Testskript-Kommandos auch die ganze Mächtigkeit der verwendeten Skriptsprache zur Verfügung hat, wie die Möglichkeit der Verwendung von Variablen, die üblichen Kontrollstrukturen (Schleifen, Bedingungen), eigene Funktionen usw. Zudem sind die Testskript-Funktionen, da sie als Quelltext vorliegen und nicht kompiliert werden müssen, durch den Anwender einsehbar, aber vor allem auch ohne weiteres modifizier- und erweiterbar. Die Unterstützung für ein neues (z.B. eigenes ActiveX-) Steuerelement kann vom Nutzer somit selbständig programmiert werden, wofür ihm beispielsweise von ATOS zur Unterstützung eine Vorlage bereit gestellt werden kann.

Jedes Testskript muss dabei natürlich syntaktisch korrekt für die verwendete Skriptsprache sein. Für einen Testdurchlauf wird dann der entsprechende Interpreter mit dem Testskript gestartet und dieses somit abgearbeitet. Der Return-Wert kann danach zur Auswertung dienen, ob der Test erfolgreich war, oder welcher Fehler aufgetreten ist. Natürlich werden die Testskript-Funktionen auch wieder Ausgaben in eine Protokoll-Datei schreiben.

Die Umsetzung dieses Konzepts würde in ATOS die HTS-, die ATS- und die Interpretationsschicht überflüssig werden lassen. Zu überarbeiten wären jedoch die Editor-Komponente und das Capturing-Modul. Auch in Hinsicht auf eine angestrebte Plattformunabhängigkeit von ATOS wäre der Umstieg auf eine unabhängige, etablierte Skriptsprache für die Testskripte sehr vorteilhaft. Mit der aktuellen Technik müssten für eine Anpassung von ATOS für den Einsatz unter einem anderen Betriebssystem – nur in Hinsicht auf die Funktionsschicht – mindestens die ATS, der ATS-Interpreter und das Übersetzungsmodul von HTS nach ATS ausgetauscht werden, was auch kompilierten Code betrifft. Leicht kann es passieren, dass bestimmte HTS-Kommandos unter dem neuen Betriebssystem nicht realisierbar sind oder neue Kommandos erforderlich werden. Dann ziehen sich die Änderungen noch tiefer ins System hinein. Für den Fall, dass keine eigene Testskript-Sprache mehr verwendet wird, würden sich die nötigen Modifikationen auf eine Überarbeitung der vorimplementierten Testskript-Funktionen beschränken. Dabei ist es dann weniger problematisch, wenn bestimmte Funktio-

nen betriebssystemspezifisch sind, da es sich ja nicht mehr um eine eigene Skriptsprache handelt, sondern um Bibliotheken von Funktionen zur Realisierung von GUI-Tests für verschiedene Zielplattformen.

Trennung von Komponenten

Das ATOS-Programmpaket besteht aus verschiedenen funktionell eigenständigen Teilen, wie das Projektmanagement, die Editor-Komponente, der HTS- und der ATS-Interpreter. Diese Komponenten sind zwar im Hinblick auf die objektorientierte Modellierung voneinander separiert, werden jedoch zusammen in eine Binärdatei kompiliert. Das wirkt sich nachteilig auf die mögliche Nachnutzung sowie die Wart- und Erweiterbarkeit aus. Besser sollten abgrenzbare Programmteile auch in separaten Bibliotheken (Dynamic Link Libraries – DLLs) untergebracht werden. Dann wäre es nicht mehr nötig, immer das gesamte Programmpaket zu übersetzen, wenn an einer Komponente Änderungen gemacht wurden. Auch lassen sich Komponenten einfach austauschen, wenn sie ein einheitliches Interface implementieren, so dass ein flexibles Programmpaket entsteht, das sich z.B. besser an verschiedene Zielplattformen anpassen lässt.

Genau das andere Extrem stellt sich bei der *RCParser*-Komponente dar, die vollständig getrennt vom ATOS-Hauptprogramm als eigenes Projekt existiert und als autarke, ausführbare Binärdatei kompiliert wird. In Hinsicht auf die starke Zusammengehörigkeit des *RCParsers* und des ATOS-Hauptprogramms in der Praxis, ist es ungünstig, dass beide Programme auf keine Weise verknüpft sind. So ist es aufwändig, den *RCParser* separat zu starten, die GUI-Daten zu bearbeiten und schließlich in eine URF-Datei zu exportieren, um danach zum ATOS-Hauptprogramm zu wechseln und die Exportdatei dort zu importieren. Statt dessen sollte es möglich sein, den *RCParser* direkt aus ATOS heraus zu starten und die vorgenommenen Änderungen direkt zu übernehmen.

A. Anhang

A.1. Testfälle des XCtl-Systems für den Praxistest der Capturing-Erweiterung

A.1.1. Allgemeine Einstellungen (AE.1)

Kurzbeschreibung

Die Dialogbox „Allgemeine Einstellungen“ wird aufgerufen. Benutzerangaben werden festgelegt, verworfen, überschrieben und schließlich durch Beendigung des XCTL-Programmes gesichert. Das XCTL-Programm wird erneut gestartet und das Fenster „Allgemeine Einstellungen“ aufgerufen, um die aktuellen Benutzerangaben zu vergleichen.

Vorbereitung

Schritt	Aktionen	Erklärung
1.	Existenz aller Umgebungsdateien des XCTL-Systems im Programmverzeichnis überprüfen	Gültigen und startfähigen Ausgangszustand des XCTL-Systems sicherstellen
2.	1. Umbenennen der Datei `XCONTROL.INI` in `XCONTROL.BAK` 2. Kopieren der Datei `.\ENV\TEST_XCONTROL.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `XCONTROL.INI`	Sicherung der originalen Datei `XCONTROL.INI` und Ersetzung durch eine präparierte Konfiguration

Testsequenz

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
1.	Starten des XCTL-Systems (Ausführen der `Xcontrol.exe`)	Hauptfenster des XCTL-Systems öffnet sich

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
2.	20 Sekunden warten (Initialisierung des XCTL-Systems abwarten)	
3.	Hauptmenü: Einstellungen → Einstellungen...	Dialogbox „Allgemeine Einstellungen“ öffnet sich, alle aktuellen Benutzerangaben werden angezeigt
4.	1. Strom = **20** mA 2. Hochspannung = **10** kV 3. Wellenlänge = **1.123** 4. Nutzer = **Testnutzer** 5. Besondere Bedingungen = **Test!!** 6. Name = **Testname** 7. Substrat = **TestSub** 8. Orientierung = [**123**] 9. Untersuchter Reflex = [**456**] 10. Button **Ok** anklicken	Dialogbox „Allgemeine Einstellungen“ schließt sich
5.	Hauptmenü: Einstellungen → Einstellungen...	Dialogbox „Allgemeine Einstellungen“ öffnet sich
6.	1. Hochspannung auf **50** kV setzen 2. Button **Ok** anklicken	Dialogbox „Allgemeine Einstellungen“ schließt sich
7.	Hauptmenü: Datei → Beenden	Das Hauptfenster des XCTL-Systems schließt sich
8.	10 Sekunden warten (Beendigungsvorgang des XCTL-Systems abwarten)	
9.	Starten des XCTL-Systems (Ausführen der Xcontrol.exe)	Das Hauptfenster des XCTL-Systems öffnet sich
10.	20 Sekunden warten (Initialisierung des XCTL-Systems abwarten)	

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
11.	Hauptmenü: Einstellungen → Einstellungen...	1. Dialogbox „Allgemeine Einstellungen“ öffnet sich 2. Alle neuen Benutzerangaben werden angezeigt und müssen verglichen werden
12.	Button **Ok** anklicken	Dialogbox „Allgemeine Einstellungen“ schließt sich
13.	Hauptmenü: Datei → Beenden	Das Hauptfenster des XCTL-Systems schließt sich
14.	10 Sekunden warten (Beendigungsvorgang des XCTL-Systems abwarten)	
15.	Anwenden von `.\BIN\DataDiff.exe` auf die Dateien `XCONTROL.INI` und `.\REF\XCONTROL.INI.REF`	Vergleich der Konfigurationsdatei mit ihrer zugehörigen Solldatei muss erfolgreich sein

Nachbereitung

Schritt	Aktionen	Erklärung
1.	Umbenennen der Datei `DEVELOP.BAK` in `DEVELOP.INI`	Wiederherstellung der originalen Datei `DEVELOP.INI`

A.1.2. Ablaufsteuerung (AS.1)

Kurzbeschreibung

Die Dialogbox zur Ausführung von Makros wird aufgerufen. Zusätzliche Makros werden aus der Datei `SCAN.MAK` nachgeladen. Das Makro „AreaScanJob“ wird daraufhin gleich wieder aus der Liste entfernt. Das Makro „ScanJob“ wird gestartet, abgebrochen und schließlich beendet (Fehlender Antrieb Theta in der Konfiguration `OTOPO_HARDWARE.INI` führt zum Verharren während Makroausführung). Das präparierte Makro „Test“ wird ausgewählt und ausgeführt, kurz darauf unterbrochen und wieder fortgesetzt. Während der Ausführung der Befehle werden die Antriebe auf bestimmte Positionen bewegt, um von dort aus die Halbwertsbreite zu bestimmen. Die weitere Ausführung bewegt den Antrieb DF relativ zur aktuellen Position, setzt den Zähler, StepScan-Parameter und Filenamen und speichert schließlich eine datenlose `*.crv` Datei in das Arbeitsverzeichnis. Da für diese Kommandos Line- und AreaScan-Fenster vorhanden sein müssen, werden sie vor Ausführung der Makrodatei geöffnet. Alle gemessenen Intensitäten des 0-dimensionalen Testdetektors werden auf dem Bildschirm und in der Protokolldatei `MACRO.LOG` protokolliert.

Vorbereitung

Schritt	Aktionen	Erklärung
1.	Existenz aller Umgebungsdateien des XCTL-Systems im Programmverzeichnis überprüfen	Gültigen und startfähigen Ausgangszustand des XCTL-Systems sicherstellen
2.	1. Umbenennen der Datei `XCONTROL.INI` in `XCONTROL.BAK` 2. Kopieren der Datei `.\ENV\TEST_XCONTROL.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `XCONTROL.INI`	Sicherung der originalen Datei `XCONTROL.INI` und Ersetzung durch eine präparierte Konfiguration

Schritt	Aktionen	Erklärung
3.	1. Umbenennen der Datei `HARDWARE.INI` in `HARDWARE.BAK` 2. Kopieren der Datei `.\ENV\OTOPO_HARDWARE.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `HARDWARE.INI`	Sicherung der originalen Datei `HARDWARE.INI` und Ersetzung durch eine präparierte Konfiguration
4.	1. Umbenennen der Datei `STANDARD.MAK` in `STANDARD.BAK` 2. Kopieren der Datei `.\ENV\TEST_STANDARD.MAK` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `STANDARD.MAK`	Sicherung der originalen Datei `STANDARD.MAK` und Ersetzung durch eine präparierte Konfiguration

Testsequenz

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
1.	Starten des XCTL-Systems (Ausführen der `Xcontrol.exe`)	Das Hauptfenster des XCTL-Systems öffnet sich
2.	20 Sekunden warten (Initialisierung des XCTL-Systems abwarten)	
3.	Hauptmenü: Ausführen → Makro...	Dialogbox „Makro ausführen“ öffnet sich
4.	Button **Laden** anklicken	Liste der Makros verlängert sich um die Einträge „ScanJob“ und „AreaScanJob“
5.	1. Ablauf auf **AreaScanJob** setzen 2. Button **Löschen** anklicken	Liste der Makros verkürzt sich um den Eintrag „AreaScanJob“

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
6.	1. **Bei Verlassen sichern** deaktivieren 2. **Ausführlicher Report** aktivieren 3. Ablauf auf **ScanJob** setzen 4. Button **Starten** anklicken	1. Protokollierung im Feld unter „Report“ beginnt 2. Button „Start“ ist ausgegraut 3. Button „Anhalten“ ist aktivierbar
7.	Button **Anhalten** anklicken	1. Button „Start“ wird zu „Weiter“ 2. Button „Anhalten“ wird zu „Beenden“
8.	Button **Beenden** anklicken	1. Button „Weiter“ wird zu „Start“ 2. Button „Beenden“ wird zu „Anhalten“ (ausgegraut)
9.	1. **Bei Verlassen sichern** aktivieren 2. **Ausführlicher Report** aktivieren 3. Ablauf auf **Test** setzen 4. Button **Start** anklicken	1. Protokollierung im Feld unter „Report“ beginnt 2. Button „Start“ ist ausgegraut 3. Button „Anhalten“ ist aktivierbar
10.	1 Sekunde warten (Positionierung läuft)	
11.	Button **Anhalten** anklicken	1. Button „Start“ wird zu „Weiter“ 2. Button „Anhalten“ wird zu „Beenden“
12.	2 Sekunden warten (Positionierung unterbrochen)	
13.	Button **Weiter** anklicken	1. Button „Beenden“ wird zu „Anhalten“ 2. Button „Weiter“ wird zu „Start“ (ausgegraut)

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
14.	1 Minute warten (Messung abwarten)	Messagebox „Information“ erscheint: Halbwertsbreite: 21.67 arcsec
15.	Button **Ok** anklicken	Protokollierung im Feld unter „Report“ wird fortgesetzt
16.	10 Sekunden warten (Positionierung abwarten)	1. Button „Start“ wird aktivierbar 2. Button „Anhalten“ wird ausgegraut
17.	Button **Abbrechen** anklicken	1. Dialogbox „Makro ausführen“ schließt sich 2. Ausgabe des Makroreports in die Datei `MACRO.LOG` im Programmverzeichnis des XCTL-System
18.	Hauptmenü: Datei → Beenden	Das Hauptfenster des XCTL-Systems schließt sich
19.	10 Sekunden warten (Beendigungsvorgang des XCTL-Systems abwarten)	
20.	Anwenden von `.\bin\DataDiff.exe` auf die Dateien `MACRO.LOG` und `.\ref\MACRO.LOG.REF`, sowie `MCROSCAN.CRV` und `.\ref\MCROSCAN.CRV.REF`	Vergleich des Makroreports und der Datendatei mit den zugehörigen Solldateien muss erfolgreich sein

Nachbereitung

Schritt	Aktionen	Erklärung
1.	Umbenennen der Datei `XCONTROL.BAK` in `XCONTROL.INI`	Wiederherstellung der originalen Datei `XCONTROL.INI`
2.	Umbenennen der Datei `HARDWARE.BAK` in `HARDWARE.INI`	Wiederherstellung der originalen Datei `HARDWARE.INI`
3.	Umbenennen der Datei `STANDARD.BAK` in `STANDARD.MAK`	Wiederherstellung der originalen Datei `STANDARD.MAK`
4.	Löschen der Dateien `MACRO.LOG` und `MCROSCAN.CRV` aus dem Programmverzeichnis des XCTL-Systems	Ausgangszustand im Programmverzeichnis des XCTL-Systems wieder herstellen

A.1.3. AreaScan (ARS.2)

Kurzbeschreibung

Das AreaScan-Fenster wird geöffnet. PSD-Messungen mit Parametereinstellungen für den 0-dimensionalen Testdetektor werden vorgenommen. Für Omega und Theta werden dabei Offsets festgelegt. Die erhaltenen Messwerte werden in `*.crv` Dateien zerlegt und zu einer kürzeren Datei zusammengefügt, so dass wiederum `*.psd` bzw. `*.rep` Dateien entstehen.

Vorbereitung

Schritt	Aktionen	Erklärung
1.	Existenz aller Umgebungsdateien des XCTL-Systems im Programmverzeichnis überprüfen	Gültigen und startfähigen Ausgangszustand des XCTL-Systems sicherstellen
2.	1. Umbenennen der Datei `XCONTROL.INI` in `XCONTROL.BAK` 2. Kopieren der Datei `.\ENV\TEST_XCONTROL.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `XCONTROL.INI`	Sicherung der originalen Datei `XCONTROL.INI` und Ersetzung durch eine präparierte Konfiguration
3.	1. Umbenennen der Datei `HARDWARE.INI` in `HARDWARE.BAK` 2. Kopieren der Datei `.\ENV\1TOPO_HARDWARE.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `HARDWARE.INI`	Sicherung der originalen Datei `HARDWARE.INI` und Ersetzung durch eine präparierte Konfiguration
4.	Löschen der evt. vorhandenen Dateien `TEST3.PSD`, `TEST4.PSD`, `TEST3.REP`, `TEST4.REP` sowie alle `TEST000[0-8].CRV` auf `C:\`	Dateien `TEST3.PSD`, `TEST4.PSD`, `TEST3.REP`, `TEST4.REP` sowie alle `TEST000[0-8].CRV` sollen durch den Testfall erzeugt werden

Testsequenz

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
1.	Starten des XCTL-Systems (Ausführen der `Xcontrol.exe`)	Hauptfenster des XCTL-Systems öffnet sich
2.	20 Sekunden warten (Initialisierung des XCTL-Systems abwarten)	
3.	Hauptmenü: Öffnen → AreaScan-Fenster	„AreaScan“-Fenster öffnet sich
4.	Fenster-Menüpunkt: „Setup zum AreaScan. . . “ aktivieren	Dialogbox „Einstellungen AreaScan“ öffnet sich
5.	Einstellungen Omega: 1. Minimum = **-1** 2. Maximum = **3** 3. Schritt = **0.5** Detektor: 4. Counter auswählen 5. Meßzeit = **1** sec 6. Impulse = **30000** Einstellungen 2Theta: 7. Minimum = **0** 8. Schritt = **0.2** 9. Relation = **3** 10. Bereich = **1** Speicher-Optionen: 11. **Kontinuierlich sichern** deaktivieren 12. **Bei Beenden speichern** deaktivieren 13. Sicherungs-Verzeichnis = **C:** 14. **Offset verwenden** aktivieren	Dialogbox „Einstellungen Offset“ öffnet sich

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
6.	Omega indirekt: 1. „Winkel“ auf **3** setzen 2. „entspricht Winkel“ auf **11** setzen 3. Button **Offset berechnen** anklicken	Omega direkt: „Winkel-Offset“ steht auf 8.000000
7.	Theta direkt: 1. „Winkel-Offset“ auf **-5** setzen 2. Button **Winkel berechnen** anklicken	Theta indirekt: 1. „Winkel“ steht auf 0.000000 2. „entspricht Winkel“ steht auf -5.000000
8.	Button **OK** anklicken	Dialogbox „Einstellungen Offset“ schließt sich
9.	In Dialogbox „Einstellungen AreaScan“ Button **Ok** anklicken	Dialogbox „Einstellungen AreaScan“ schließt sich
10.	Fenster-Menüpunkt: „AreaScan starten“ aktivieren	Messagebox „Start Scan“ erscheint: Starte Omega/2Theta-Scan(SLD) mit Omega von -1.0000 bis 3.0000?
11.	Button **Ja** anklicken	Eine Messagebox „Meldung“ erscheint mit dem Text: Soll die Messung im Protokollbuch gespeichert werden?
12.	Button **Nein** anklicken	In Statuszeile erscheint: Anfahren der Startposition ... Scanvorgang beginnt mit Ausgabe in die Statuszeile
13.	1 Minute warten (Messung abwarten)	1. In der Statuszeile erscheint die Nachricht: 9 Scan’s wurden gemessen 2. Eine Messagebox „Meldung“ mit folgendem Text erscheint: Die Messung wurde abgeschlossen

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
14.	Button **Ok** anklicken	Kurvengrafik entspricht ungefähr einer Vorlage
15.	Hauptmenü: Datei → Sichern unter... aktivieren	File-Dialogbox „Sichern unter...“ erscheint
16.	Unter „Dateiname“ **TEST3.PSD** eintragen Dialog mit **Ok** Button beenden	1. In Statuszeile erscheint: Scan 9 von 9 schreiben 2. Ausgabe in die Dateien `TEST3.PSD` und `TEST3.REP`
17.	Fenster-Menüpunkt: „Datenbasis zerlegen...“ aktivieren	Dialogbox „Datenbasis zerlegen“ erscheint
18.	1. Dateiname = **TEST** 2. Datenpfad = **C:** 3. Button **OK** anklicken	1. Dialogbox „Datenbasis zerlegen“ schließt sich 2. In Statuszeile erscheint: `c:\test0008.crv` 3. Dateien `TEST000[0-8].CRV` werden auf `C:\` erstellt
19.	Hauptmenü: Datei → Neu	Kurve im „AreaScan“-Fenster verschwindet
20.	Fenster-Menüpunkt: „Datenbasis zusammensetzen...“ aktivieren	Dialogbox „Datenbasis zusammensetzen“ erscheint
21.	Button **Kurve wählen!** anklicken	File-Dialogbox „Datenbasis-File wählen“ erscheint
22.	1. Unter „Dateiname“ **TEST0000.CRV** eintragen 2. Button **Ok** anklicken	1. File-Dialogbox „Datenbasis-File wählen“ schließt sich 2. „Beginnen bei“ steht auf `TEST0000.CRV` (ehemaliger Button „Kurve wählen!“) 3. „Beenden bei“ steht auf `TEST0008.CRV` 4. „Anzahl Spektren“ steht auf 9
23.	1. „Anzahl Spektren“ auf **4** setzen 2. Button **OK** anklicken	„Beenden bei“ steht auf `TEST0003.crv`

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
24.	Button **Ok** anklicken	1. Dialogbox „Datenbasis zusammensetzen“ schließt sich 2. In Statuszeile erscheint: `TEST0003.crv`
25.	Hauptmenü: Datei → Sichern unter. . . aktivieren	File-Dialogbox „Sichern unter. . . “ erscheint
26.	1. Unter „Dateiname“ **TEST4.PSD** eintragen, 2. Dialog mit **Ok** Button beenden	1. In Statuszeile erscheint: Scan 4 von 4 schreiben 2. Ausgabe in die Dateien `TEST4.PSD` und `TEST4.REP`
27.	Kreuzchen im „AreaScan“-Fenster oben rechts anklicken	„AreasScan“-Fenster schließt sich
28.	Hauptmenü: Datei → Beenden	Hauptfenster des XCTL-Systems schließt sich
29.	10 Sekunden warten (Beendigungsvorgang des XCTL-Systems abwarten)	
30.	Anwenden von `.\bin\DataDiff.exe` auf die Dateien `TEST3.PSD`, `TEST4.PSD`, `TEST3.REP`, `TEST4.REP` `TEST000[0-3].CRV` und `.\ref\TEST3.PSD.REF`, `.\ref\TEST4.PSD.REF`, `.\ref\TEST3.REP.REF`, `.\ref\TEST4.REP.REF`, `.\ref\TEST000[0-8].CRV.REF`	Vergleiche der Messwerte- und Reportdateien mit ihren zugehörigen Solldateien müssen erfolgreich sein

Nachbereitung

Schritt	Aktionen	Erklärung
1.	Umbenennen der Datei `XCONTROL.BAK` in `XCONTROL.INI`	Wiederherstellung der originalen Datei `XCONTROL.INI`
2.	Umbenennen der Datei `HARDWARE.BAK` in `HARDWARE.INI`	Wiederherstellung der originalen Datei `HARDWARE.INI`
3.	Löschen der Dateien `TEST3.PSD`, `TEST4.PSD`, `TEST3.REP`, `TEST4.REP` sowie alle `TEST000[0-8].CRV` auf `C:\`	Ausgangszustand auf `C:\` wieder herstellen

A.1.4. LineScan (LS.2)

Kurzbeschreibung

Das LineScan-Fenster wird aufgerufen. Ein ContinuousScan wird mit dem 0-dimensionalen Testdetektor von Herrn Damerow gestartet, angehalten, weitergeführt, abgebrochen und erneut gestartet. Die über den jeweiligen Winkeln gemessenen Intensitäten werden in Form einer Kurve dargestellt und als `*.crv` Dateien abgespeichert.

Vorbereitung

Schritt	Aktionen	Erklärung
1.	Existenz aller Umgebungsdateien des XCTL-Systems im Programmverzeichnis überprüfen	Gültigen und startfähigen Ausgangszustand des XCTL-Systems sicherstellen
2.	1. Umbenennen der Datei `XCONTROL.INI` in `XCONTROL.BAK` 2. Kopieren der Datei `.\ENV\TEST_XCONTROL.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `XCONTROL.INI`	Sicherung der originalen Datei `XCONTROL.INI` und Ersetzung durch eine präparierte Konfiguration
3.	1. Umbenennen der Datei `HARDWARE.INI` in `HARDWARE.BAK` 2. Kopieren der Datei `.\ENV\1TOPO_HARDWARE.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `HARDWARE.INI`	Sicherung der originalen Datei `HARDWARE.INI` und Ersetzung durch eine präparierte Konfiguration
4.	Löschen der evt. vorhandenen Datei `TEST3.CRV` auf `C:\`	Datei `TEST3.CRV` soll durch den Testfall erzeugt werden

Testsequenz

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
1.	Starten des XCTL-Systems (Ausführen der `Xcontrol.exe`)	Hauptfenster des XCTL-Systems öffnet sich
2.	20 Sekunden warten (Initialisierung des XCTL-Systems abwarten)	
3.	Hauptmenü: Öffnen → LineScan-Fenster	„Diffraktometrie“-Fenster öffnet sich
4.	Fenster-Menüpunkt: „Setup ContinuousScan. . . “ aktivieren	Dialogbox „Einstellungen ContinuousScan“ öffnet sich
5.	1. ScanAchse = **Omega** 2. Startpunkt = **-4** 3. Endpunkt = **6** 4. Bereichsgröße = **0.05** 5. Sensor = **Simulant** 6. Meßzeit = **0.2** sec 7. **Bei Beenden speichern** aktivieren 8. Sicherungs-Verzeichnis = **C:** 9. Button **Parameter aktualisieren** anklicken	1. „Geschwindigkeit“ steht auf 0,2500 2. „Meßpunkte“ steht auf 200
6.	Button **Ok** anklicken	Dialogbox „Einstellungen ContinuousScan“ schließt sich
7.	Fenster-Menüpunkt: „kontinuierlichen Scan starten“ aktivieren	Messagebox „Start Scan“ mit folgender Frage erscheint: Einen kontinuierlichen Scan mit Omega von -4.0000 bis 6.0000 starten?
8.	Button **Ja** anklicken	File-Dialogbox „Sichern unter ...“ erscheint

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
9.	1. Unter „Dateiname“ **TEST3.CRV** eintragen 2. Dialog mit **Ok** Button beenden	Eine Messagebox „Meldung“ erscheint mit dem Text: Soll die Messung im Protokollbuch gespeichert werden?
10.	Button **Nein** anklicken	In Statuszeile erscheint: Anfahren der Startposition ... Scanvorgang beginnt mit Ausgabe in die Statuszeile
11.	5 Sekunden warten (Messung abwarten)	
12.	Hauptmenü: Ausführen → Messung unterbrechen	In Statuszeile erscheint: Messung wurde unterbrochen ! Scanvorgang unterbricht Ausgabe in die Statuszeile und in die Datei `TEST3.CRV`
13.	5 Sekunden warten (Messung unterbrochen)	
14.	**ESC**-Taste drücken	Scanvorgang setzt Ausgabe in die Statuszeile und in die Datei `TEST3.CRV` fort
15.	5 Sekunden warten (Messung weiterführen)	
16.	**ESC**-Taste drücken	In Statuszeile erscheint: Messung wurde unterbrochen ! Scanvorgang unterbricht Ausgabe in die Statuszeile und in die Datei `TEST3.CRV`
17.	Fenster-Menüpunkt: „kontinuierlichen Scan starten“ aktivieren	Messagebox „Start Scan“ mit folgender Frage erscheint: Einen kontinuierlichen Scan mit Omega von -4.0000 bis 6.0000 starten?

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
18.	Button **Ja** anklicken	File-Dialogbox „Sichern unter ...“ erscheint
19.	1. Unter „Dateiname“ **TEST3.CRV** eintragen 2. Dialog mit **Ok** Button beenden	Eine Messagebox „Sichern unter ...“ erscheint: `C:\TEST3.CRV` besteht bereits. Möchten Sie sie ersetzen?
20.	Button **Ja** anklicken	Eine Messagebox „Meldung“ erscheint mit dem Text: Soll die Messung im Protokollbuch gespeichert werden?
21.	Button **Nein** anklicken	In Statuszeile erscheint: Anfahren der Startposition ... Scanvorgang beginnt mit Ausgabe in die Statuszeile
22.	40 Sekunden warten (Messung abwarten)	Eine Messagebox „Meldung“ erscheint: Die Messung wurde abgeschlossen und gespeichert.
23.	Button **Ok** anklicken	Kurvengrafik entspricht ungefähr einer Vorlage
24.	Kreuzchen im LineScan-Fenster oben rechts anklicken	„LineScan“-Fenster schließt sich
25.	Hauptmenü: Datei → Beenden	Das Hauptfenster des XCTL-Systems schließt sich
26.	10 Sekunden warten (Beendigungsvorgang des XCTL-Systems abwarten)	
27.	Anwenden von `.\bin\DataDiff.exe` auf die Dateien `TEST3.CRV` und `TEST3.CRV.REF`	Vergleich der Messwertedatei mit ihrer zugehörigen Solldatei muss erfolgreich sein

Nachbereitung

Schritt	Aktionen	Erklärung
1.	Umbenennen der Datei `XCONTROL.BAK` in `XCONTROL.INI`	Wiederherstellung der originalen Datei `XCONTROL.INI`
2.	Umbenennen der Datei `HARDWARE.BAK` in `HARDWARE.INI`	Wiederherstellung der originalen Datei `HARDWARE.INI`
3.	Löschen der Datei `TEST3.CRV` auf `C:\`	Ausgangszustand auf `C:\` wieder herstellen

A.1.5. Motorsteuerung (MS.1)

Kurzbeschreibung

Die Dialogboxen „Grundstellung anfahren“ und „Verfahren nach Encoder-Position“ werden benutzt. Der Antrieb DF wird auf seinen absoluten Nullpunkt zurück gesetzt, auf seine Indexposition und schließlich auf seine Grundstellung gefahren. Über die „Direkte Steuerung“ wird schließlich der Antrieb DF wieder auf 0 gefahren. Hier wird außerdem ein Testmotor vom Typ TMotor ausgewählt und bewegt. Der Antrieb Tilt wird neu kalibriert und seine Position über die „Direkte Steuerung“ ausgelesen.

Vorbereitung

Schritt	Aktionen	Erklärung
1.	Existenz aller Umgebungsdateien des XCTL-Systems im Programmverzeichnis überprüfen	Gültigen und startfähigen Ausgangszustand des XCTL-Systems sicherstellen
2.	1. Umbenennen der Datei `XCONTROL.INI` in `XCONTROL.BAK` 2. Kopieren der Datei `.\ENV\TEST_XCONTROL.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `XCONTROL.INI`	Sicherung der originalen Datei `XCONTROL.INI` und Ersetzung durch eine präparierte Konfiguration
3.	1. Umbenennen der Datei `HARDWARE.INI` in `HARDWARE.BAK` 2. Kopieren der Datei `.\ENV\1TOPO_HARDWARE.INI` in das Programmverzeichnis des XCTL-Systems und Umbenennen in `HARDWARE.INI`	Sicherung der originalen Datei `HARDWARE.INI` und Ersetzung durch eine präparierte Konfiguration

Testsequenz

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
1.	Starten des XCTL-Systems (Ausführen der `Xcontrol.exe`)	Das Hauptfenster des XCTL-Systems öffnet sich
2.	20 Sekunden warten (Initialisierung des XCTL-Systems abwarten)	
3.	Hauptmenü: Einstellungen → Antriebe → Grundstellung...	Dialogbox "Grundstellung anfahren"öffnet sich
4.	1. Antrieb auf **DF** setzen 2. Button **Absoluter Nullpunkt** anklicken	In der Dialogbox „Grundstellung anfahren“ erscheint: **Absolute Null für Motor DF**
5.	1. Antrieb auf **DF** setzen 2. **Position beibehalten** deaktivieren 3. **Grundstellung für alle Antriebe** deaktivieren 4. **Gültige Kalibrierungsdaten** aktivieren 5. Button **Referenzpunktlauf** anklicken	1. Button „Absoluter Nullpunkt“ ist ausgegraut 2. Button „Referenzpunktlauf“ ist ausgegraut 3. Combobox „Antriebsauswahl“ ist ausgegraut 4. Text „Absolute Null für Motor DF“ verschwindet
6.	30 Sekunden warten (Kalibrierung abwarten)	In der Dialogbox „Grundstellung anfahren“ erscheint der Text: Index DF
7.	35 Sekunden warten (Positionierung abwarten)	1. In der Dialogbox „Grundstellung anfahren“ erscheint: Target Pos DF 2. Eine Messagebox „Meldung“ erscheint: Kalibrierung beendet !

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
8.	Button **Ok** in der Messagebox anklicken	1. Messagebox „Meldung“ schließt sich 2. Button „Absoluter Nullpunkt“ ist aktivierbar 3. Button „Referenzpunktlauf“ ist aktivierbar 4. Combobox „Antriebsauswahl“ ist aktivierbar
9.	Button **Ok** anklicken	Dialogbox „Grundstellung anfahren“ schließt sich
10.	Hauptmenü: Einstellungen → Antriebe → Direkte Steuerung...	Dialogbox „Verfahren nach Encoder-Position“ öffnet sich
11.	„Motor:“ auf **DF** setzen	„Position“ und „Neue Position“ stehen auf 10247
12.	1. „Neue Position“ auf **0** setzen 2. **Enter** drücken	Schieberegler bewegt sich sichtbar nach links
13.	5 Sekunden warten (Motorbewegung abwarten)	„Position“ und „Neue Position“ stehen wieder auf 0
14.	„Motor:“ auf **Test** setzen	„Position“ und „Neue Position“ stehen auf 0
15.	1. „Neue Position“ auf **50** setzen 2. **Enter** drücken	Schieberegler bewegt sich sichtbar nach rechts
16.	1 Sekunden warten (Motoreinstellung abwarten)	„Position“ und „Neue Position“ stehen auf 50
17.	1. „Neue Position“ auf **-50** setzen 2. **Enter** drücken	Schieberegler bewegt sich sichtbar nach links
18.	1 Sekunden warten (Motoreinstellung abwarten)	„Position“ und „Neue Position“ stehen auf -50
19.	Button **Abbrechen** anklicken	Dialogbox „Verfahren nach Encoder-Position“ schließt sich
20.	Hauptmenü: Einstellungen → Antriebe → Grundstellung...	Dialogbox „Grundstellung anfahren“ öffnet sich

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
21.	1. Antrieb auf **Tilt** setzen 2. **Position beibehalten** deaktivieren 3. **Grundstellung für alle Antriebe** deaktivieren 4. **Gültige Kalibrierungsdaten** deaktivieren	1. Checkbox „Position beibehalten“ ist ausgegraut 2. Checkbox „Grundstellung für alle Antriebe“ ist ausgegraut
22.	Button **Referenzpunktlauf** anklicken	Messagebox „Meldung“ erscheint: Es wird davon ausgegangen, dass sich der Antrieb in der Position der physikalischen Null befindet. Es werden Kalibrierungsdaten geändert !
23.	Button **Ja** anklicken	1. Button „Referenzpunktlauf“ ist ausgegraut 2. Combobox „Antriebsauswahl“ ist ausgegraut
24.	15 Sekunden warten (Kalibrierung abwarten)	In der Dialogbox „Grundstellung anfahren“ erscheint der Text: Index Tilt
25.	15 Sekunden warten (Positionierung abwarten)	1. In der Dialogbox „Grundstellung anfahren“ erscheint: Target Pos Tilt 2. Eine Messagebox „Meldung“ erscheint: Kalibrierung beendet !
26.	Button **Ok** in der Messagebox anklicken	1. Messagebox „Meldung“ schließt sich 2. Button „Absoluter Nullpunkt“ ist aktivierbar 3. Button „Referenzpunktlauf“ ist aktivierbar 4. Combobox „Antriebsauswahl“ ist aktivierbar

Schritt	Aktionen und Eingaben	Ergebnisse und Ausgaben
27.	Button **Ok** anklicken	Dialogbox „Grundstellung anfahren“ schließt sich
28.	Hauptmenü: Einstellungen → Antriebe → Direkte Steuerung...	Dialogbox „Verfahren nach Encoder-Position“ öffnet sich
29.	„Motor:“ auf **Tilt** setzen	„Position“ und „Neue Position“ stehen auf 0
30.	Button **Abbrechen** anklicken	Dialogbox „Verfahren nach Encoder-Position“ schließt sich
31.	Hauptmenü: Datei → Beenden	Das Hauptfenster des XCTL-Systems schließt sich
32.	10 Sekunden warten (Beendigungsvorgang des XCTL-Systems abwarten)	

Nachbereitung

Schritt	Aktionen	Erklärung
1.	Umbenennen der Datei `XCONTROL.BAK` in `XCONTROL.INI`	Wiederherstellung der originalen Datei `XCONTROL.INI`
2.	Umbenennen der Datei `HARDWARE.BAK` in `HARDWARE.INI`	Wiederherstellung der originalen Datei `HARDWARE.INI`

Literaturverzeichnis

[1] Ames, Alexander K. and Haward Jie: *Critical Paths for GUI Regression Testing*, Univ. of California, Santa Cruz, http://www.soe.ucsc.edu/ sasha/proj/gui_testing.pdf, 2004

[2] AutoTester, Inc.: *AutoTester®*, http://www.autotester.com, 2005

[3] Beier, Michael: *Automatische Tests Grafischer Benutzungsoberflächen*, OBJEKTspektrum 3/2004, S. 53-57

[4] Bergmann, Sebastian: *PHPUnit*, http://www.phpunit.de, 2005

[5] Bothe, Klaus: *Reverse Engineering: the Challenge of Large-Scale Real-World Educational Projects*, CSEE&T 2001, 14th Conference on Software Engineering Education and Trainig, Charlotte, USA, http://www.informatik.hu-berlin.de/swt/projekt98/, Febr. 2001

[6] Bothe, Klaus und Ulrich Sacklowski: *Praxisnähe durch Reverse Engineering-Projekte: Erfahrungen und Verallgemeinerungen*, 7. Workshop SEUH, Zürich, CH, http://www.informatik.hu-berlin.de/swt/projekt98/, Febr. 2001

[7] Cioroianu, Andrei: *Java Desktop Development*, O'Reilly Media, Inc., ONJava.com, http://www.onjava.com/pub/a/onjava/2004/02/18/desktop.html, 2004

[8] Dakshinamurthy, K.: *Marathon*, http://marathonman.sourceforge.net, 2004

[9] Feathers, Michael: *CppUnit – C++ port of JUnit*, http://sourceforge.net/projects/cppunit, 2004

[10] Gamma, Erich: *JUnit*, http://www.junit.org, 2002

[11] Gerrard, Paul: *Testing GUI Applications*, EuroSTAR '97, 24-28. November 1997, Edinburgh UK, Systeme Evolutif Limited, 1997

[12] Hanisch, Jens, Johann Letzel und Klaus Bothe: *Ein Werkzeugsystem zur Automatisierung von GUI-Tests*, Gesellschaft für Informatik e.V., Software-Trends Band, 23 Heft 1, Februar 2003

[13] Hanisch, Jens und Johann Letzel: *Automatisierung von Regressionstests eines Programms zur Halbleiter-Strukturanalyse*, Humboldt-Universität zu Berlin, 2003

[14] Hanisch, Jens, Johann Letzel und Andreas Hirth: *ATOS-Benutzerleitfaden*, Interne Dokumentation, Humboldt-Universität zu Berlin, 2005

[15] Hanisch, Jens, Johann Letzel und Andreas Hirth: *HTS-Sprachspezifikation*, Interne Dokumentation, Humboldt-Universität zu Berlin, 2005

[16] IBM Corporation: *Rational Functional Tester*, `http://www.ibm.com/software/awdtools/tester/functional/`, 2005

[17] IBM Corporation: *Rational® Software*, `http://www.ibm.com/software/rational/`, 2005

[18] Letzel, Johann: *ATS-Sprachspezifikation*, Interne Dokumentation, Humboldt-Universität zu Berlin, 2003

[19] Liggesmeyer, Peter: *Software-Qualität*, Spektrum Akademischer Verlag, 2002

[20] Marsh, Kyle: *Win32 Hooks*, Microsoft Developer Network Technology Group, `http://msdn.microsoft.com/library/en-us/dnwui/html/msdn_hooks32.asp`, 1994

[21] Mercury Interactive Corporation: *WinRunner®*, `http://www.mercury.com`, 2005

[22] Microsoft® Corporation: *MSDN Library (Microsoft Developer Network)*, `http://msdn.microsoft.com/library`, 2005

[23] Petzold, Charles: *Windows-Programmierung*, Microsoft Press, 2000

[24] Compuware Corporation: *QAHiperstation*, `http://www.compuware.com`, 2005

[25] QES Testing b.v.: *QES/Architect*, `http://www.qestest.com`, 2005

[26] Rossum, Guido van: *Python*, http://www.python.org, 2005

[27] Two, Michael C., Charlie Poole, Jamie Cansdale and Gary Feldman: *NUnit*, http://www.nunit.org, 2005

[28] Wall, Larry, *Perl*, http://www.perl.com, 2005

[29] Warner, Bruce: *Introduction to Regression Test Automation Software*, http://www.operativesoft.com/html/introregress.htm, 2004