HOCHSCHULE
MITTWEIDA
University of Applied Sciences

Fakultät Angewandte Computer- und Biowissenschaften

Professur Angewandte Informatik - Softwareentwicklung

Bachelorarbeit

Analyse und Erweiterung einer bestehenden Testumgebung für mobile Endgeräte im Kontext der Einbindung von iOS-Geräten.

Ansgar Dabow

Mittweida, den 20. Oktober 2021

Erstprüfer: Prof. M.Sc. Christian Roschke
Zweitprüfer: Dipl.–Ing. Thomas Lerche

Dabow, Ansgar
Analyse und Erweiterung einer bestehenden Testumgebung für mobile Endgeräte im Kontext der Einbindung von iOS-Geräten.
Bachelorarbeit, Fakultät Angewandte Computer- und Biowissenschaften
Hochschule Mittweida — University of Applied Sciences, Oktober 2021

Referat

Das Testen von Software ist unabdingbar und wird nicht nur für Desktop Applikationen, sondern auch für Applikationen auf mobilen Geräten benötigt. Der Prozess des Testens ist aufwändig. Es gibt durch Künstliche Intelligenz gestützte Lösungen, welche Systemtests zu großen Teilen eigenständig generieren und dem Entwickler das Testen erleichtern, zum Beispiel das AI4Test Projekt der Systems Multimedia Solutions GmbH.

Das Ziel dieser Bachelorarbeit besteht darin, das AI4Test Projekt zu analysieren und die bestehende Struktur so zu erweitern, dass iOS-Geräte neben Android und Desktop Geräten unterstützt werden. Dabei wird eine generalisierte Schnittstelle zur Interaktion mit Geräten geschaffen.

Das Ziel der Bachelorarbeit wird erreicht. Das AI4Test Projekt wird vorgestellt, die Schnittstelle generalisiert und die Plattform iOS wird prinzipiell unterstützt.

Name: Dabow, Ansgar
Studiengang: Angewandte Informatik - Softwareentwicklung
Seminargruppe: IF18wS-B
English title: Analysis and extension of a testenviroment of mobile devices to integrate iOS-devices.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung und Motivation

Zu Beginn wird eine Einführung in das grobe Themengebiet der Arbeit gegeben. Dazu wird das eigentliche Thema und Ziel der Arbeit beschrieben, sowie der Aufbau kurz erläutert.

1.1 Allgemeine Enführung

Das Testen von Software ist essenzieller Bestandteil in deren Entwicklung. Durch Testen kann Fehlverhalten entdeckt und verhindert werden. Dadurch kann Zeit und Geld gespart werden, in Extremfällen sogar das Ansehen oder Leben gerettet werden. [Cle10].

Dieser Prozess ist allerdings zeitintensiv, weshalb zum einen versucht wird, das Testen zu automatisieren und zum anderen, den Inhalt der Tests automatisch generieren zu lassen. Mittels künstlicher Intelligenz wird versucht, das automatische Generieren von Testfällen umzusetzen.

Gerade im Web-Umfeld gibt es bereits den vielversprechenden Ansatz, künstliche Intelligenz (KI) zu verwenden, um das Generieren von Testfällen zu übernehmen. Dazu wird einem neuronalen Netz das Erkennen von Basiselementen einer Webseite, wie zum Beispiel Buttons, Textfeldern, (Dropdown-)Menüs angelernt. Durch den Einsatz dieser neuronalen Netze und Cluster-Algorithmen können Testsysteme analysiert und modelliert werden. Das ermöglicht es bereits heute, Testfälle ohne menschliche Interaktion automatisiert zu generieren. Das AI4Test-Projekt von T-Systems Multimedia Solution GmbH setzt das bereits um.

Jedoch wird im Web-Umfeld nicht nur mit Desktop-Browser getestet, sondern auch mit mobilen Endgeräten, zum Beispiel auf Smartphones oder Tabletts. Das Testen auf mobilen Endgeräten erscheint sinnvoll, da ein Großteil der Medien auf dem Smartphone konsumiert werden. Die Abbildung 1.1 verdeutlicht diesen Umstand

mit Daten aus Deutschland. Auch außerhalb von Deutschland kann die Situation ähnlich bewertet werden nach statcounter. [Sta20]

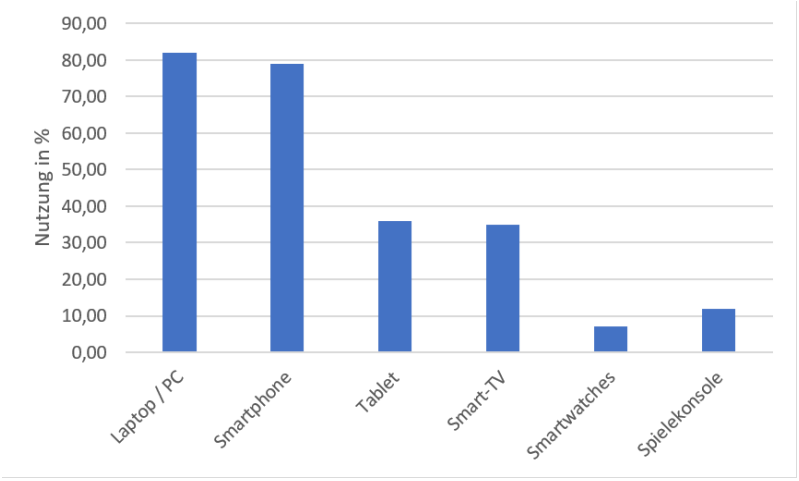

Abbildung 1.1: Gerätenutzung im Jahr 2019 nach Wahl [Wah19]

Das Testen mit mobilen Geräten ist besonders aufwändig, da hier eine sehr große Diversität an Geräten gegeben ist, die sich zum Beispiel stark in der Displaygröße unterscheiden. Das hat zur Folge, dass Tests in sehr vielen unterschiedlichen Varianten ausgeführt werden müssen. Dabei müssen verschiedenste Displaygrößen, Betriebssystem oder Betriebssystemversionen beachtet werden, um möglichst viele Geräte mit den Tests abzudecken und Kompatibilität für diese zu gewährleisten. Die grundlegend andere Bedienung von mobilen Endgeräten erschwert das Testen im Vergleich zu Desktop-Systemen außerdem. Es gibt Elemente welche anders dargestellt oder verwendet werden und andere Bedienmuster wie zum Beispiel das Swipen.

Das AI4Test-Projekt funktioniert nicht nur für Desktop-Systemen, sondern auch für mobile Geräten, welche mit dem Betriebssystem Android laufen. Jedoch existieren weitere Geräte, mit anderen Betriebssystemen, welche noch nicht unterstützt werden. Dazu zählen iOS-Geräte.

Für jede Geräteklasse ist eine gesonderte Schnittstelle zum Interagieren mit den Geräten notwendig, welche jeweils individuell konfiguriert werden muss, damit sie korrekt funktioniert.

1.2 Aufgabenbeschreibung und Zielstellung

Aktuell verwendet das AI4Test-Projekt verschiedene Schnittstellen zu den unterschiedlichen Plattformen, die unterstützt werden. Das ist zum einen ein *Selenium*-Treiber für Desktop-Browser und zum anderen ein *Appium*-Treiber für Android-Browser.

Diese Treiber müssen jeweils um ihre Besonderheiten, wie zum Beispiel PopUp-Verhalten konfiguriert und angepasst werden.

Ziel dieser Arbeit soll der Entwurf einer generalisierten Schnittstelle zum Ansteuern von Endgeräten sein, welcher von dem AI4Test-*Crawler* verwendet werden kann. Der Fokus liegt hierbei zunächst darauf, die Schnittstelle von Android- und iOS-Geräten zu generalisieren und teilweise zu implementieren. Darüber hinaus soll auch die Generalisierung bei anderen Komponenten des AI4Test-Projektes geprüft werden.

1.3 Aufbau der Arbeit

Inhalt dieser Arbeit wird die Analyse des AI4Test-Projekts sein. Es werden als Überblick die Einzelkomponenten vorgestellt und beschrieben. Dabei wird auf die Probleme beim mobilen Testen eingegangen.

Außerdem soll die generelle Funktion eines *Appcrawlers* untersucht werden, um daraufhin die Schnittstelle bei AI4Test zwischen *Crawler* und mobilem Gerät zu beschreiben. Hierbei sollen Möglichkeiten zur Interaktion mit dem mobilen Endgerät gezeigt werden. Dabei wird gesondert auf iOS-Endgeräte eingegangen.

Der IST-Stand der aktuellen Schnittstelle bei AI4Test zu mobilen Android-Geräten soll analysiert und darauffolgend dessen Umsetzbarkeit für iOS-Geräte ausgewertet werden. Aus den gewonnenen Informationen wird ein generalisiertes Model abgeleitet und durch teilweise Implementierung getestet.

2 Grundlagen

In den nachfolgenden Abschnitten wird der IST-Stand des AI4Test-Projektes im Detail beschrieben und auf die einzelnen Komponenten des Gesamtsystems eingegangen. Damit soll das grundlegende System mit seinen Schnittstellen, die Funktion und die Abläufe beschrieben und erklärt werden und ein genereller Überblick geschaffen werden.

2.1 Systemtests

> „Die Aufgabe des Systemtests besteht darin, das Ganze zu testen und zu bestätigen, dass das gewünschte Ergebnis tatsächlich erreicht wird.“ [SBS12, S. 2]

Ein Systemtest unterscheidet sich zum Beispiel von einem Unittest darin, dass das Gesamtsystem mit allen beteiligen Komponenten in ihrem Zusammenwirken getestet wird und nicht modulare Einzelkomponenten für sich.

Das Testen von Software ist wie das Entwickeln von Software im stetigen Wandel. In der Vergangenheit wurde dabei viel auf das Wasserfallmodell bei der Entwicklung gesetzt, welches heute allerdings immer mehr durch die Agile Software Entwicklung ersetzt wird [Sha17, S. 84f]. Darauf muss auch bei dem Testen von Software reagiert werden. Während in der Vergangenheit vermehrt auf manuelle Tests gesetzt wurde, werden diese mittlerweile immer weiter durch Testautomatisierungen verdrängt [Sha17, S. XII]. Auch wird das Testen vermehrt nicht mehr vom Entwickler durchgeführt, sondern von Personen, welche nur für das Testen zuständig sind [Sha17, S. 20]. Genauso entwickeln sich die Spezialisierungen der einzelnen Tester weiter, sodass neue Rollen bei der Softwareentwicklung entstehen. Solche Rollen sind zum Beispiel: Funktions-, Automatisations-, Performance-, Sicherheits- oder Benutzerfreundlichkeitstester [Sha17, S. 9]. Die Testmethoden und Technologien sind von dem

Wandel nicht ausgeschlossen. Zuvor bestand der Wandel darin, dass manuelle Tests durch Testautomatisierung abgelöst wurden. Dafür sind verschiedene Testautomatisierungsframeworks entstanden, welche es ermöglichen, Testskripte zu erstellen. Es wird jedoch weiterhin nach Wegen gesucht, effizienter zu testen. [Sha17, S. 23] Einer dieser Wege besteht darin, die Tests durch den Einsatz von KI zu unterstützen. Dabei ist das Ziel, den Aufwand durch KI zu reduzieren oder die Qualität der Tests zu erhöhen. Eine Möglichkeit dafür ist zum Beispiel das Testen durch ein neuronales Netz, welches Teile des Testens übernimmt oder die Automatisierung generiert. Dadurch wird der meist aufwändige und kostspielige manuelle Testaufwand minimiert und durch Testautomatisierung ersetzt.

Eine weitere Herausforderung für den Test und ein fundamentaler Wandel ist die vermehrte Nutzung von mobilen Endgeräten neben den herkömmlichen Desktop-Geräten. So gut wie jede Person besitzt mindestens ein mobiles Endgerät. Sei es Smartphone, Tablet oder gar Smartwatch. Abbildung 1.1 aus Kapitel 1 zeigte bereits, dass Medien im Jahr 2019 zu fast gleichen Anteilen auf dem Laptop beziehungsweise Desktop-PC und dem Smartphone konsumiert werden.

Diesen Trend bemerkte zum Beispiel auch die Firma Google. Sie passte ihre Webseitenanalyse so an, dass präferiert und später ausschließlich die mobile Ansicht einer Webseite genutzt wird. Dadurch erhalten Webseiten, welche für die mobile Ansicht optimiert sind oder zumindest eine mobile Ansicht bereitstellen, einen Vorteil in der Bewertung. [Lam19, S. 216f]

Für das Testen von Software ist dieser Trend von großer Bedeutung. Tests auf mobilen Endgeräten werden immer wichtiger. Der nächste Abschnitt beschäftigt sich mit den Besonderheiten und Herausforderungen beim Testen auf mobilen Endgeräten.

2.1.1 Herausforderungen beim Testen auf mobilen Endgeräten

Das Testen auf mobilen Endgeräten ist eine besondere Herausforderung. Während im Desktopbereich eine überschaubare Menge an Betriebssystemen und Bildschirmauflösungen existieren, wird es in der mobilen Welt sehr unübersichtlich.

Mobile Anwendungen werden genau auf eine Betriebssystemversion (Software Development Kit (SDK)) kompiliert und können sich auf den verschiedenen Systemen unterschiedlich verhalten. Auch gibt es eine große Variation von unterschiedlichen

Displaygrößen und -formaten, welche von denen der Desktop-Geräte abweichen. Die Abbildung 2.1 zeigt eine Übersicht der verwendetet Bildschirmgrößen von Smartphones. Auffallend ist, dass so viele verschiedene Größen existieren, dass diese als „Others“ als größte Menge zusammengefasst werden. Die Verteilung von iOS und Android Betriebssystemversionen verhält sich ähnlich. [Sta20]

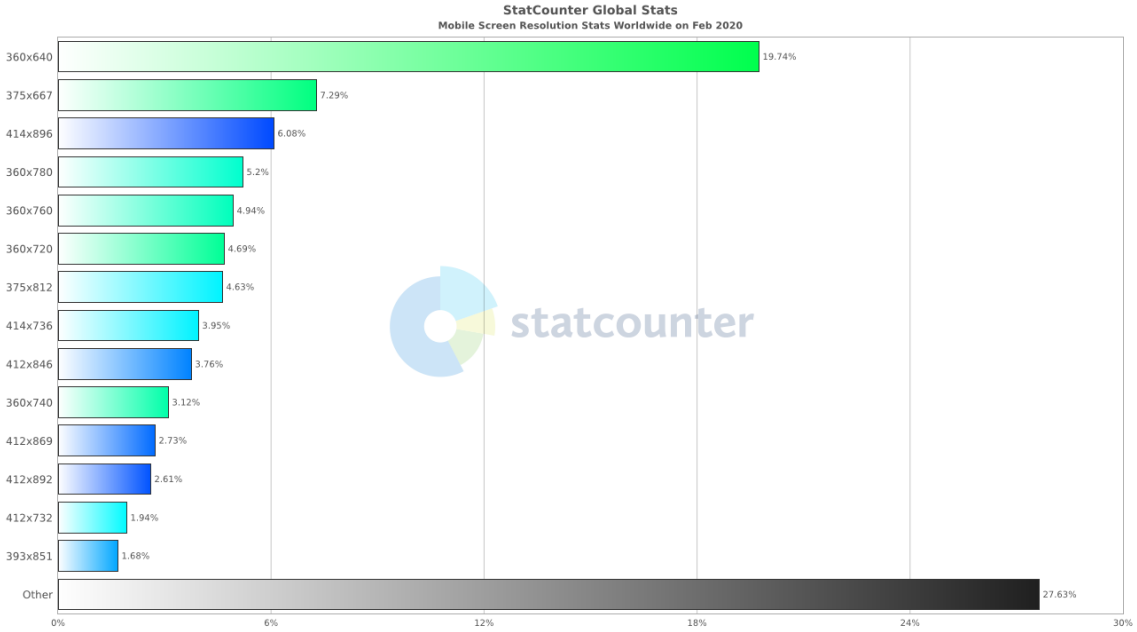

Abbildung 2.1: Smartphone Displaygrößen nach statcounter [Sta20]

Es muss also eine größere Menge an Geräten getestet werden, um wirklich sicher gehen zu können, dass eine Anwendung auf allen gängigen Geräten funktioniert. Wobei die Bezeichnung „gängige Geräte“ mit Absicht schwammig formuliert ist, da sich diese Gruppe an Geräten sehr schnell verändert. Zu der Vielfalt der Geräte kommen andere User Interface (UI)-Elemente hinzu, welche sich anders als auf Desktop-Geräten verhalten. Beispiele hierfür sind „Sandwich Menüs“ oder „Slider“.

Diese Elemente sind für die Bedienung der mobilen Endgeräte angepasst, welche sich von der Bedienung von Desktop-Geräten unterscheidet. Während ein Desktop-Gerät hauptsächlich mit Klicks, Scrollen und Texteingaben gesteuert wird, kommen bei mobilen Endgeräten noch Touchgesten hinzu. Darunter zählen beispielsweise „wischen“ mit einem oder mehreren Fingern, „drehen“, „zwicken“ oder auch „zoomen“, sowie „lange halten“. Diese Optionen können außerdem noch kombiniert werden zu Gesten wie „halten und ziehen“ was eine Zusammensetzung aus „lange halten“ und

anschließendem „wischen“ ist. Generell ändert sich das Design auf mobilen Ansichten, da ein mobiles Display in der Regel vertikal ausgerichtet ist, während ein Desktop Display hauptsächlich horizontal vorzufinden ist. Elemente werden größer als auf Desktop-Geräten dargestellt, damit sie auf dem kleineren mobilen Display erkennbar sind.

2.2 AI4Test

AI4Test ist ein Projekt der T-Systems Multimedia Solutions GmbH und beschäftigt sich mit der KI-gestützten Testfallentwicklung. Das Projekt soll es Softwareentwicklern erleichtern, ihre Software zu testen. Dies soll zum einen dadurch bewerkstelligt werden, dass Standard-Systemtests automatisiert und effizient erstellt und zum anderen, dass manuelle Tests, welche teurer und zeitaufwändiger sind, durch automatisierte Tests ersetzt werden. Kunden sollen ihre Software zur Verfügung stellen können und im Anschluss eine Reihe bereits automatisierter Testfälle für die Software erhalten, welche auf verschiedenen Plattformen wie Desktop, Android oder iOS ausführbar ist. Das Projekt automatisiert die Erstellung der automatischen Tests und reduziert dadurch den Aufwand der eigentlichen Testautomatisierung erheblich. Ebenfalls möglich ist ein Abgleich zwischen zwei verschiedenen Softwareständen, also ob und was sich zwischen zwei Versionen geändert hat.

An dem Projekt wird seit dem Jahr 2017 gearbeitet. Es startete mit dem Zusammenfassen von Seiten einer Webseite und dem *Crawlen* durch selbige.[Her18] Die Funktionalität für Desktop-Browser, sowie Android-Browser existiert bereits. Für iOS-Browser und native Anwendungen auf mobilen Endgeräten wird bereits an einer Integrierung gearbeitet. Somit können Webanwendungen bereits mit dem AI4Test-Projekt getestet werden, teils mit mobilen Tests.

AI4Test unterteilt sich in verschiedene Komponenten, welche als Schema in Abbildung 2.2 dargestellt sind. Auffallend ist, dass das AI4Test-Projekt nicht nur ein Programm ist, sondern aus mehreren unabhängigen Programmen besteht. Externe Programme sind durch die „Programmgrenze“ gekennzeichnet.

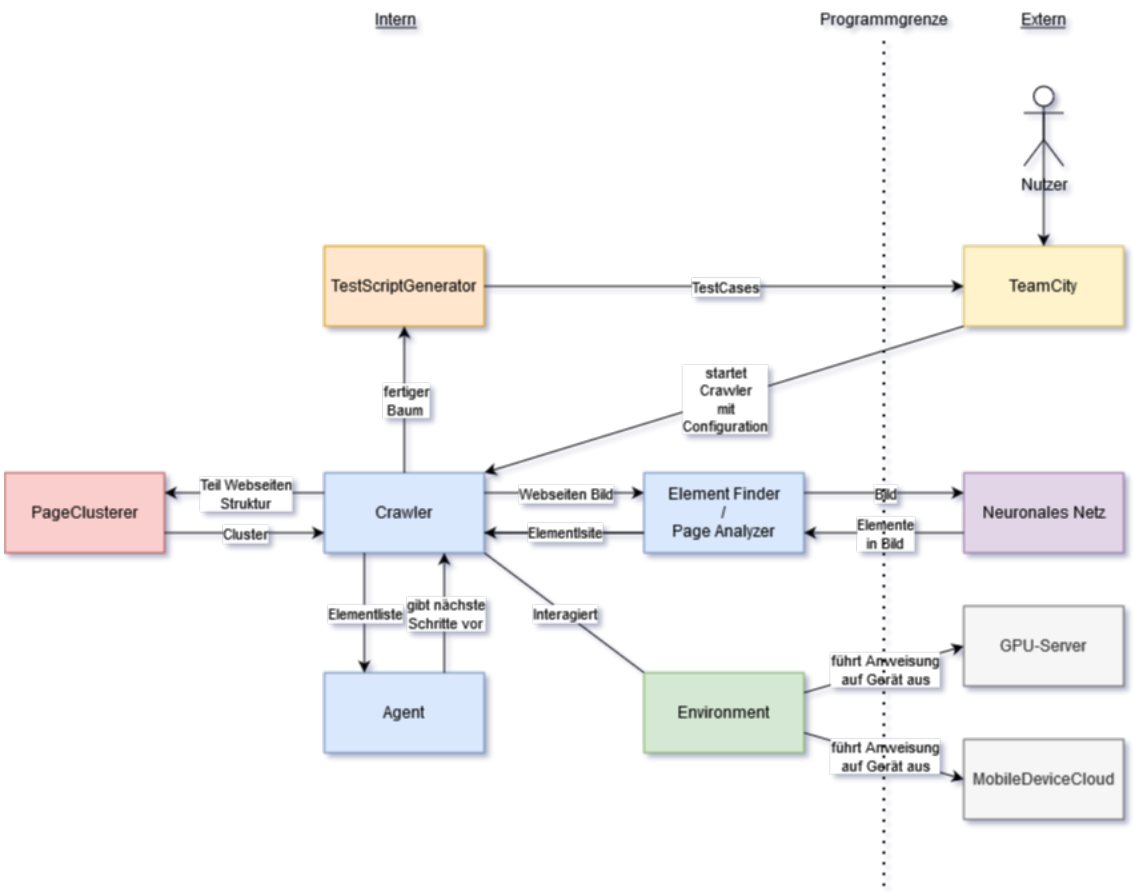

Abbildung 2.2: Komponenten des AI4Test-Projektes

Der Nutzer interagiert über die TeamCity-Komponente mit dem Projekt. Von dieser Komponente wird der gesamte Ablauf gestartet und die Ergebnisse werden auch hier für den Nutzer sichtbar. Von der TeamCity-Komponente wird der *Crawler* gestartet. Die Agent-Komponente bekommt vom *Crawler* eine Elementliste. Anhand dieser gibt die Agent-Komponente dem *Crawler* die nächsten abzuarbeitenden Schritte vor. Diese werden über die Schnittstelle der *Environment*-Komponente auf den Endgeräten ausgeführt. Anschließend wird der Stand auf dem Endgerät in Form eines Bildes und der Webseitenstruktur festgehalten. Während die *PageClusterer*-Komponente die Struktur analysiert und versucht Seitengruppen zu erkennen, nutzt die *ElementFinder*-Komponente das Bild, um visuelle Elemente zu finden. Anhand dieser Daten generiert der *Crawler* einen Graphen der Webseite. Dieser wird abschließen an die *TestScriptGenerator*-Komponente übergeben, welche daraus Testfälle und entsprechend ausführbaren Code für die Webseite generiert, welche dem

Nutzer zur Verfügung gestellt werden. Die Komponenten werden im Detail in den nächsten Abschnitten vorgestellt.

2.2.1 PageClusterer

Der im Abschnitt 2.2.4 beschriebene *Crawler* versucht alle Seiten einer Webseite zu finden. Gerade in ContentManagementSystemen (CMS) oder Online Shopsystemen haben viele Seiten die gleiche Funktion und unterscheiden sich nicht von der Struktur. Als Beispiel hierfür kann in einem Onlineshop eine Produktseite gesehen werden. In der Regel wird innerhalb eines Shops immer die gleiche Vorlage benutzt und lediglich die Produktinformationen angepasst. Das heißt, überall existiert eine Überschrift, ein Beschreibungstext, ein Kaufen-Button und so weiter. Diese Seiten müssen nicht alle einzeln auf ihre Funktionalität geprüft werden, da sie sich nicht unterscheiden und davon ausgegangen werden kann, dass wenn eine Seite funktioniert, auch die anderen Seiten funktionieren. Es bietet sich daher an, diese Seitengruppen zu finden und somit den Testumfang deutlich zu reduzieren. Die Aufgabe des *PageClusterers* ist also: Seitengruppen finden, welche gleiche Funktionalität besitzen und diese gruppieren. Dafür wird das Gerüst der jeweiligen Seite (Document Object Model (DOM)-Baum) analysiert, welches vom *Crawler* übergeben wird. Anhand eines Grenzwertes wird festgestellt, ob eine Seite zu einer Seitengruppe gehört oder nicht.

Das Clustering soll auch für die Durchführung auf iOS-Geräten funktionieren. Dabei sollten die gleichen Cluster wie auf anderen mobilen Endgeräten gefunden und erstellt werden.

2.2.2 ElementFinder

Die Aufgabe des *ElementFinders* im AI4Test-Projekt ist das Identifizieren von interaktiven Elementen auf einem Bild, einer Webseite oder eines Bildschirms. Er wird genutzt, um die Nutzersicht so nahe wie möglich nachzuempfinden, welcher die visuellen Elemente sieht, nicht jedoch das darunter liegende Gerüst einer Webseite.

Diese Komponente beinhaltet auch die „AI“ (im Deutschen „KI“), auf welche vom Projektnamen „AI4Test“ zu schließen ist. Sie ist dafür zuständig, Elemente in einem Bild zu identifizieren. Das neuronale Netz besteht aus einer YOLO v3 Bilderkennung

(You Only Look Once)[RF18] in DarkNet (einem Framework für neuronale Netze) [Red16] implementiert. Das Ergebnis der Bilderkennung sind Kästen, welche das Element umschließen („BoundingBoxes“) und einige Zusatzinformationen beinhalten. Aus den BoundingBoxes kann die *ElementFinder*-Komponente relevante Elemente der Webseite ableiten und an den *Crawler* zurückgeben, welcher diese für weitere Aktionen verwendet. In Abbildung 2.3 ist ein Ergebnis des *ElementFinders* zu sehen. Es wurde aus einem Testlauf auf einem iOS-Gerät entnommen. Die roten Kästen umschließen die gefundenen Elemente und zeigen Ihre erkannte Größe. Die grünen Kreise markieren das Zentrum des gefundenen Elementes.

Abbildung 2.3: Visualisiertes Ergebnis des Elementfinders

Das neuronale Netz erwartet sehr konkrete Attributeigenschaften, damit sinnvolle Ergebnisse geliefert werden können. Dazu gehören unter anderem: Bildauflösung oder Seitenverhältnis. Das Netz wurde nicht speziell für die mobile Anwendung trainiert und kann auch nicht zwischen Webseitenelementen und Systemelementen unterscheiden. Die aufrufende Komponente muss dafür sorgen, dass die Attributeigenschaften gegeben sind und keine ungewollten Elemente auf dem Bild zu sehen sind, wie zum Beispiel Systemmenüs des Browsers wie es in Abbildung 2.3 der Fall ist. Es wurden Systemelemente wie der vor und zurück Button gefunden oder Uhrzeit und Signalqualität.

Für die geplante Einbindung von iOS-Geräten muss evaluiert werden, wie die Bilder der Webseite angepasst werden müssen, damit diese sinnvoll vom *ElementFinder* verwendet werden können.

2.2.3 TestscriptGenerator

Wie der Name dieser Komponente vermuten lässt, generiert sie die ausführbaren Testskripte. Diese sind das Ergebnis des AI4Test-Projektes und schlussendlich das Produkt, welches der Nutzer erhält. Diese Komponente agiert eigenständig und ist nicht direkt in den *Crawler* integriert. Sie wird extern (über das TeamCity-Interface) ausgeführt.

Als Übergabeparameter verwendet der Testskriptgenerator den vom *Crawler* generierten Graphen, welcher die Webseitenstruktur mit Interaktionselementen beinhaltet.

Die TestSkripte selbst sind Java Anwendungen welche Testfälle des Frameworks TestNG beinhalten. TestNG ist ein Java Testframework, welches im Vergleich zu anderen Frameworks, zum Beispiel JUnit, für mehr als nur UnitTests konzipiert ist. [Beu07, S. XVII f]

Die Java Anwendung wird bei der Generierung vorkonfiguriert, damit die Testausführung möglichst einfach gestartet werden kann. Die Testskripte sollen auch für iOS-Geräte generiert werden können. Ebenfalls soll die Konfigurierung für iOS-Geräte möglich sein.

2.2.4 Crawler

Ein *Web Crawler* kurz *Crawler* ist ein Programm, welches das Internet nach Webseiten durchsucht. Die Technologie ist ein wesentlicher Bestandteil von Suchmaschinen, wird aber auch in dem AI4Test-Projekt verwendet. Hier findet sie Anwendung, um alle möglichen Seiten, Ansichten und deren Elemente einer vorgegebenen Webseite zu finden. Die *CrawlerKlasse* beinhaltet außerdem die Startmethode, welche den gesamten Ablauf startet.

Der *Crawler* generiert einen Graphen, welcher alle gefundenen Seiten und den Pfad zu diesen speichert. Dieser Graph wird von anderen Komponenten wie den TestscripGenerator weiterverwendet.

Der Ablauf des *Crawlers* beginnt zunächst mit dem Initialisieren des *Environments*, welches zur Interaktion mit einem Browser verwendet wird. Folgend wird eine Liste von verbotenen und erlaubten Uniform Resource Identifier (URI) generiert, welche dafür sorgt, dass der *Crawler* nicht die zu untersuchende Webseite verlässt. Ohne diese Liste würde der *Crawler* den Fokus verlieren und alle denkbaren Webseiten finden und untersuchen. Bei dieser Initialisierung werden die *Desired Capabilities* generiert und die Schnittstelle zum eigentlichen Gerät hergestellt. Die *Desired Capabilities* werden im späteren Abschnitt 2.2.6 weiter erklärt.

Nach dieser Initialisierung startet die Hauptschleife, welche so lange durchlaufen wird, bis die Warteschlange, welche die nächstens Schritte beinhaltet, leer ist oder eine maximale Anzahl an Schritten getätigt wurden. Ein Schleifendurchlauf startet damit, dass der Cookie-Stand auf einen definierten Stand zurückgesetzt wird, damit jeder Durchlauf die gleichen Voraussetzungen hat. Ein veränderter Cookie-Stand kann das Ergebnis verfälschen, da die Webseite anders reagieren könnte. Als Beispiel kann ein Session-Cookie gesehen werden, welcher einen Nutzer authentifiziert. Ist ein solcher Cookie vorhanden, ist der Ablauf zum Anmelden ein anderer als ohne Cookie, da der Nutzer erst abgemeldet werden muss.

Anschließend wird vom Agenten der nächste Schritt für den *Crawler* abgefragt. Diese Anweisung beinhaltet eine URI, zu welcher dann navigiert wird. Die Navigation wird gerätespezifisch ausgeführt und kann sich zwischen den verschiedenen Architekturen unterscheiden.

Nachfolgend wird der anstehende Schritt vorbereitet und anschließend auch ausgeführt. Die Vorbereitung besteht darin, störende Elemente, wie zum Beispiel PopUps, zu entfernen. PopUps auf Webseiten könnten zum Beispiel Newsletteraufforderungen sein, Rabatt Erinnerungen oder ein Chatbot, welcher seine Hilfe anbietet. Um diese Elemente zu erkennen und zu entfernen, wird auf eine JavaScript-Schnittstelle zurückgegriffen. Genauer gesagt, es wird ein JavaScript-Programm in die Webseite injiziert und anschließend ausgeführt. Sobald die Vorbereitung abgeschlossen ist, wird die eigentliche Anweisung ausgeführt. Es wird also gescrollt, geklickt oder getippt. Danach Erfolg eine Überprüfung, ob der *Crawler* sich noch im definierten Fokus befindet. Ist das der Fall, wird der neue Stand in Form eines Bildschirmfotos und der aktuellen Webseitenstruktur festgehalten und der Graph erweitert, welcher die Ergebnisse des *Crawlers* speichert. Das erstellte Abbild wird an weiteren Kom-

ponenten übergeben, welche daraus weitere Aktionen für den *Crawler* ableiten und in die Warteschlange einreihen.

Auf diese Weise wird die Webseite indiziert und das Ergebnis als Graph gespeichert.

Der *Crawler* soll zusätzlich zu den aktuell unterstützen Plattformen, Desktop und Android, auch Webseiten auf iOS-Browsern durchsuchen können. Dabei sollte er optimalerweise fehlertolerant für typische Unterbrechungen wie PopUps sowie Cookie Banner sein.

2.2.5 MobileDeviceCloud

Die *MobileDeviceCloud* ist ein T-Systems Multimedia Solutions Projekt, welches reale mobile Endgeräte für Softwaretests zur Verfügung stellt. Dies hat den Vorteil, dass die Software auf echten Geräten getestet wird und nicht auf Emulatoren, welche das echte Gerät nicht vollständig nachstellen können. Die *MobileDeviceCloud* wird zum Beispiel von Kunden wie der CoronaWarnApp, Henkel oder Zeiss genutzt.[Sol] Das AI4Test-Projekt nutzt ebenfalls die *MobileDeviceCloud* zur Interaktion mit mobilen Geräten.

Als Ausführungsumgebung nutzt die *MobileDeviceCloud* die Software „Experitest“ der Firma Digital.ai Software Inc, welche wiederum einen eigenen Treiber „SeeTest“ zur Interaktion mit den Mobilen Endgeräten bereitstellt. Dieser Treiber übersetzt die Anweisungen in einen eigenen nativen Code, was eine zusätzliche Umwandlung ist, im Vergleich zu *Appium*. Diese Umwandlung ermöglicht es zusätzliche Funktionalität zu gewährleisten.

Die *MobileDeviceCloud* soll auch für iOS-Geräte die Ausführungsplattform sein. iOS-Geräte sollen ähnlich wie Android-Geräte angesprochen werden können.

2.2.6 Treiber

Treiber erleichtern es, vereinfacht gesagt, dem Programmierer, mit einem Gerät über eine generalisierte Schnittstelle zu interagieren. Dies wird ermöglicht, indem eine standardisierte Gruppe an Anweisungen definiert wird, welche von einem in sich geschlossenem System (dem Treiber) umgesetzt werden, um auf einem speziellen System zu funktionieren. Somit muss der Entwickler nur noch die Gruppe an Anweisungen kennen und kann mehrere Systeme ansprechen, solange für diese ein Treiber existiert, welcher die Anweisungen für das jeweilige System implementiert. [CRKH05, S. 1f]

Im AI4Test-Projekt wird auch auf Gerätetreiber zurückgegriffen. Die nachfolgenden Abschnitte sollen diese kurz erläutern.

Das Selenium Framework ist die Grundlage des im vorherigen Abschnitt 2.2.5 genannten „SeeTest“ Treiber. Dieser erweitert lediglich den Funktionsumfang des *Selenium* Treibers.

Selenium ist Teil des „World Wide Web Consortium“ (W3C) und in der Gesamtheit ein Testframework, welches wiederum aus vielen Komponenten besteht. Für diese Arbeit besonders wichtig ist die „WebDriver“-Komponente, welche die Schnittstelle zwischen Programm und nativen Browser-Kommando darstellt. Für alle aktuellen Browser (Chrome, Firefox, Edge und Safari) existiert eine Browserspezifische Implementierung des WebDrivers, sodass diese vom *Selenium* Framework angesprochen werden können. [Gun18, S. 7f]

Appium ist ähnlich wie der „SeeTest“ Treiber eine Erweiterung des *Selenium*-Frameworks, welche auf die Ansteuerung von mobilen Geräten und deren Anwendungen, hybriden Anwendungen und Browser spezialisiert ist. Hybride Anwendungen unterscheiden sich von nicht hybriden Anwendungen darin, dass sie im Grunde nur eine Web-Applikation darstellen. [Gun18, S. 10f]

Der RemoteDriver ist eine gesonderte Form eines Treibers. Er interagiert nicht direkt mit dem anzusteuernden Gerät, sondern mit einem entfernten Server. Der Server wiederum interagiert mit dem Gerät.[Gun18, S. 149ff] Für den mobilen Teil von AI4Test ist der Server die *MobileDeviceCloud*, welche im Abschnitt 2.2.5 vorgestellt wird. Die Geräte werden also nicht direkt angesprochen, sondern über einen Server.

Abbildung 2.4 visualisiert noch einmal den Aufbau des Frameworks. Von rechts nach links gelesen: Test-Skripte oder auch der *Crawler* sind das ausführende Programm und geben vor, was getan wird. *Appium* stellt Funktionen zur Interaktion mit Geräten zur Verfügung, welche von den Test-Skripten oder dem *Crawler* über den RemoteDriver genutzt werden (In der Abbildung ausgegraut als Client und Server dargestellt). *Appium* selbst gibt diese Anweisungen an die gerätespezifischen Treiber weiter, welche dann das eigentliche Gerät steuern. Die Abbildung verdeutlicht außerdem den Vorteil dieses Aufbaus. Jede dargestellte Komponente ist austauschbar (In der Abbildung durch „+“ dargestellt). Dadurch ist der Aufbau sehr flexibel und kann einfach angepasst werden.

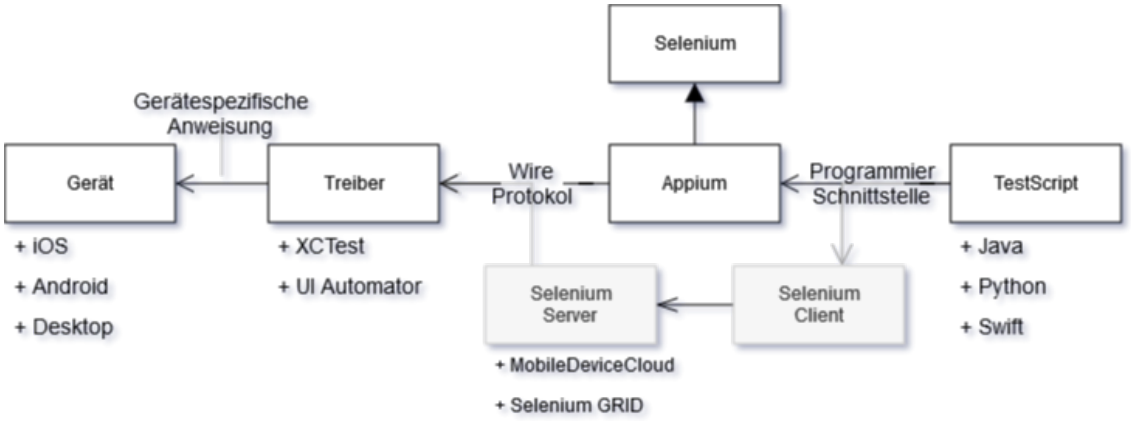

Abbildung 2.4: Schema vom Aufbau des Selenium Frameworks

XCTest ist der von Apple zur Verfügung gestellte Treiber zur Interaktion mit Apple Geräten. Zur Nutzung dieses Treibers ist ein anderes Applegerät notwendig, da das zu automatisierende Gerät vorbereitet werden muss, bevor die Automatisierung angewandt werden kann. Dafür ist eine weitere Anwendung („XTest“) notwendig, welche von Apple zur Verfügung gestellt wird und nur auf Apple Geräten ausgeführt werden kann. *Appium* ist in der Lage, *XCTest* zur Interaktion mit iOS-Geräten zu verwenden.[Gun18, S.219f]

Der Treiber Kontext ist bei der Interaktion mit Geräten zu beachten. Er bestimmt, in welchem Fenster oder welcher Anwendung gearbeitet wird. Im AI4Test-Projekt wird der native Kontext, sowie der WebView-Kontext genutzt.

In der WebView kann nur mit dem Inhalt des Browsers interagiert werden und der Browser gesteuert werden. Klicks auf Buttons einer Webseite sind in diesem Kontext zum Beispiel möglich, aber auch das ausführen von JavaScript. Das Einstellungsmenü gehört beispielsweise nicht zu diesem Kontext, da es nicht Teil der Webseite ist, sondern Element des Browserprogrammes. Um auf das Einstellungsmenü zu klicken, muss zu dem nativen Kontext gewechselt werden. Auch zum Ausführen von Gesten ist der native Kontext notwendig. Im nativen Kontext ist dafür der Inhalt der Webseite nicht sichtbar. Während des Crwlens wird hauptsächlich auf der WebView-Kontext genutzt.

Das Environment ist die Schnittstelle zwischen dem AI4Test-Projekt und der zu testenden Software. Es beinhaltet den Treiber und über diesen Grundfunktionalitäten zur Interaktion mit der Software.

Die aktuelle Struktur des Enviroments ist in Abbildung 2.5 dargestellt. Eine Elternklasse stellt alle Funktionalität zur Verfügung, die für die Verwendung von Desktop-Browsern notwendig ist. Das heißt, die Logginkonfiguration, Browserkonfiguration sowie Umgebungskonfiguration wird geladen und alle Funktionen zur Interaktion mit dem Desktop-Browser werden zur Verfügung gestellt.

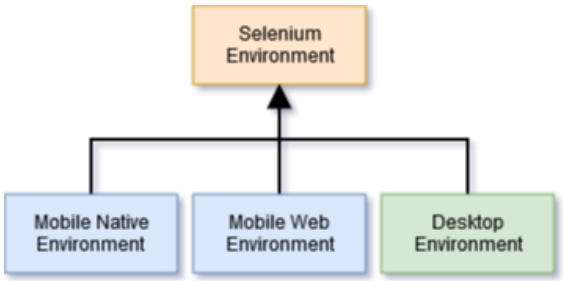

Abbildung 2.5: Strukturdiagramm initiales Environment

Eine wichtige Konfiguration hierbei ist die des Browsers. Sie beinhaltet sogenannte *Desired Capabilities*, welche zum einen den angesteuerten Browser definieren, als auch Einstellungen im Browser festlegen können. Das heißt, Werte wie Browser, Browserversion oder Standardverhalten werden hier definiert. [src21] Darüber hinaus werden über sie Eigenschaften für die Interaktion mit der *MobileDeviceCloud* definiert [Inc20].

Von der Elternklasse erben 3 weitere Klassen, welche jeweils zur Fernsteuerung spezialisiert sind, genauer zur Fernsteuerung von Desktop-Browser, mobilen Browsern und nativen mobilen Anwendungen. Die Fernsteuerung von nativen mobilen Anwendungen hält zusätzlich zur Elternklasse Attribute, welche das gewünschte Gerät sowie die Zugangsdaten zur *MobileDeviceCloud* definieren. Gleiches trifft auf die Fernsteuerung für mobile Browser zu. Die Fernsteuerung für mobile Browser enthält weitere Attribute, welche die Fenstergröße beschreiben. Diese Klasse ist für diese Arbeit besonders relevant.

Ein Teil der Methoden, welche von der Elternklasse definiert sind, werden von der Kindklasse überladen. Dazu gehören Methoden zu WebDriver Initialisierung oder spezialisierte Interaktion mit einem Gerät. Das betrifft besonders die zuvor beschriebenen *Desired Capabilities*. Des Weiteren werden neue Attribute und Methoden definiert, welche nur für den mobilen Kontext benötigt werden. Ein gutes Beispiel hierfür ist die Fenstergröße und ein dazugehörender Skalierungsfaktor.

Für die Interaktion mit den iOS-Geräten soll ähnlich zu den Android-Geräten ein auf *Selenium* basierender Treiber genutzt werden oder der bestehende so angepasst werden, dass die Interaktion auch für iOS-Geräte funktioniert.

2.2.7 TeamCity

Die Benutzeroberfläche „TeamCity“ stellt im AI4Test-Projekt die Schnittstelle zum Endnutzer dar. Auf dieser kann der Nutzer Einstellungen tätigen und somit spezifizieren, welche Anwendung getestet werden soll, welche Plattform genutzt werden soll und wie ausführlich getestet wird. Die Plattform interagiert außerdem mit den anderen Komponenten und steuert den Ablauf dieser, dient also auch zur Statusüberwachung.

2.3 Prinzipien und Design Patterns

Prinzipien sind allgemeingültige Grundsätze, nach denen gehandelt wird. Sie spielen in der Softwareentwicklung eine wichtige Rolle und das Verfolgen gewisser Prinzipien erleichtert das Entwerfen und Entwickeln von Software. [Bal09, S. 25f]

Design Pattern sind wiederverwendbare Vorlagen, welche die Qualität von Software und Code verbessern sollen. Sie sind also eine Hilfestellung für Entwickler, um gewisse Probleme nach einem gut durchdachten systematischen Muster zu lösen. [Aye18, S. 7f] Folgend werden solche Design Patterns vorgestellt, welche bei der Umsetzung dieser Arbeit helfen könnten. Dabei werden Vor- und Nachteile jedes Pattern evaluiert.

2.3.1 Strategy Design Pattern

Das Strategy Design Pattern erlaubt es, verschiedene Algorithmen für ein Problem zu verwenden, ohne dass der anwendende Teil mitbekommt, welcher genaue Algorithmus verwendet wird. [Aye18, S. 150]

Das kann hilfreich sein, wenn nicht genau bekannt ist, welcher der Algorithmen der effektivste ist. Durch dieses Design Pattern können die Algorithmen dynamisch ausgetauscht werden, je nachdem welcher der aktuell sinnvollste ist. [Aye18, S. 152]

Der Vorteil des Strategy Design Pattern ist, wie erwähnt, die Flexibilität, zur Laufzeit einer Anwendung aus verschiedenen Algorithmen den sinnvollsten zu wählen. Das macht die Software einfach erweiterbar oder wartbar. Denn die Algorithmen haben alle ein gleiches Ziel, sind jedoch unabhängig und können ohne weiteres ausgetauscht oder abgeändert werden.

Der Nachteil von diesem Pattern ist Codedopplung. Dadurch, dass viele verschiedene Algorithmen angeboten werden können, kann auch unnötiger Code auftreten, der eventuell nicht verwendet wird oder sich mit anderen Algorithmen doppelt.

2.3.2 Abstraktion

Das Prinzip der Abstraktion ist das Hervorheben des Wesentlichen, wobei das Konkrete weggelassen wird. Anders gesagt, das Erkennen gleicher Merkmale verschiedener Strukturen [Bal09, S. 26f]. Es wird als eines der wichtigsten Prinzipien der Softwareentwicklung gesehen, um komplexe Probleme zu verstehen [Kra07, S. 41].

In der objektorientierten Programmierung wird dieses Prinzip beispielsweise genutzt. Hier wird von Vererbung und Implementierung von Schnittstellen gesprochen. Wobei bei der Vererbung weitere Prinzipien, wie die Hierarchisierung, verwendet werden. Vererbung kann leicht in eine verbale Form übersetzt werden, durch das Schlüsselwort „IST“. Klasse A IST Klasse B oder noch einfacher: ein Hund IST ein Tier. Ein Tier ist also die abstrahierte Form von Hund und gibt Merkmale vor, die jedes Tier besitzt. Ein Hund wiederum kann andere und weitere Merkmale als andere Tiere besitzen.

Ein abstraktes Model ist Voraussetzung für weitere Prinzipien, wie die Strukturierung oder Hierarchisierung [Bal09, S. 49].

3 Anforderung und Entwürfe

Im vorherigen Kapitel wurde der IST Stand des AI4Test-Projektes vorgestellt. Von diesem Stand ausgehend werden im nächsten Abschnitt Anforderungen abgeleitet, welche notwendig sind, um das Ziel der Arbeit zu erreichen. Das Ziel dieser Anforderungen ist das der Arbeit selbst: die Erweiterung und Generalisierung, des bestehenden Systems, sodass iOS-Geräte unterstützt werden. Die Anforderungen werden mit einer Nummer versehen, was die Übersichtlichkeit erhöhen soll und die Anforderungen einfacher referenzieren lässt. Neben diesen qualitativen Anforderungen werden außerdem technische Anforderungen für des AI4Test-Projektes im generellen beschrieben, um den Fokus der Änderungen zu verdeutlichen.

3.1 Anforderungen

Die Anforderungen bekommen ein Kürzel, damit sie einfacher zu referenzieren sind. Die Vielfachen von Hundert stellen dabei Gruppen dar. Für jede Unteranforderung wird der Wert der Gruppe um Zehn erhöht.

Gruppe Null besteht aus grundlegenden Anforderungen. Die Gruppe 100 widmet sich Anforderungen, welche sich auf die Interaktion mit Webseiten beziehen. Gruppe 200 fasst Anforderungen zusammen, welche das Zusammenspiel von Komponenten betrifft. Die weiteren Gruppen stehen für sich allein.

/F010/ iOS-Gerät muss über die *MobileDeviceCloud* ansteuerbar sein:

Für die grundlegende Interoperabilität des Programms muss die neue Geräteklasse ansteuerbar sein. Das heißt im genauen, dass Anweisungen auf iOS-Geräten ausführbar sein müssen, wie in Abbildung 2.2 veranschaulicht, als Schnittstelle zwischen *Environment* und *MobileDeviceCloud*. Dazu muss ein Treiber initialisiert werden

und die *Desired Capabilities* müssen angepasst werden, wie in Abschnitt 2.2.6 beschrieben.

/F020/ Webseite muss aufrufbar sein

Eine grundlegende Funktion für das AI4Test-Projekt ist das Aufrufen von Webseiten, wie aus Abschnitt 2.2.4 hervorgeht. Das impliziert, dass ein Browser geöffnet werden muss, wenn dieser noch nicht geöffnet wurde. Anschließen muss eine Webseite aufrufbar sein.

/F100/ Webseite muss scrollbar sein, um größere Webseiten anzusehen:

Um von einer gesamten Webseite Bilder machen zu können, ist es notwendig durch die Webseite navigieren, respektive scrollen zu können. Diese Aktion wird in Abschnitt 2.2.4 erwähnt. Das Scrollen muss über eine fest definierte Weite geschehen, damit die erstellten Bilder um eine gewisse Größe überlappen und jedes Element der Webseite vollständig abgebildet ist. Das ist notwendig, damit der in Abschnitt 2.2.2 beschriebene *ElementFinder* die Möglichkeit hat, alle Elemente zu erkennen.

/F110/ Klicken auf Element

Um durch eine Webapplikation zu navigieren, muss auf dessen Elemente geklickt werden können. Diese Aktion wird in Abschnitt 2.2.4 mit erwähnt.

/F120/ Screenshots von einer Webseite müssen erstellt werden können:

Damit Komponenten wie der *ElementFinder*, welcher in Abschnitt 2.2.2 beschrieben wird, funktionieren, müssen Bildschirmfotos (Screenshots) erstellt werden können. Diese dürfen keine Systemelemente zeigen, da diese nicht zur Webapplikation gehören.

/F200/ Komponenten müssen funktionieren

Zur Überprüfung der Komponenten können bestehende Ergebnisse aus dem Android Projekt Ergebnissen als Vergleichswert genutzt werden.

/F210/ Default PopUps müssen behandelt werden können:

Bei der Ausführung des Programms, insbesondere des *Crawlers*, können PopUps auftreten. Das Programm muss in der Lage sein, diese zu handhaben, ohne abzubrechen. Auftretende PopUps können zum einen von der zu untersuchenden Webseite stammen (zum Beispiel Werbebanner oder Cookie Banner) und zum anderen vom Gerät selbst. Da echte Geräte zur Ausführung genutzt werden, kann es vorkommen, dass System-PopUps auftreten. Hierzu zählen beispielsweise Updatebenachrichtigungen. Das Auftreten und Umgehen von PopUps wird in Abschnitt 2.2.4 mit beschrieben.

/F220/ Cookies auf dem Gerät müssen manipuliert werden können:

Das Setzen und Löschen von Cookies ist für die konstante Ausführung notwendig, damit ein gleiches Verhalten der Webapplikation auftritt. Diese Notwendigkeit wird in Abschnitt 2.2.4 erklärt.

/F230/ *PageClusterer* für iOS

Der *PageClusterer* ist wie in Abschnitt 2.2.1 beschrieben eine essenzielle Komponente des AI4Test-Projektes und soll auch für die Plattform iOS funktionieren und eine ähnliche Anzahl an Clustern erkennen.

/F240/ *Elementfinder* für iOS

Die *Elementfinder*-Komponente wird in Abschnitt 2.2.2 vorgestellt und ist essenziell für die Projektausführung. Diese Komponente soll auch für die Plattform iOS funktionieren und äquivalente Anzahl an Elementen auf einer Webseite müssen erkannt werden.

/F300/ Funktionalität anderer Plattformen darf nicht eingeschränkt werden:

Die bestehenden Komponenten des AI4Test-Projektes haben aktuell einen funktionierenden Stand. Nach der Implementierung der neuen Architektur soll dieser Stand nicht eingeschränkt sein und auf gleichem Niveau funktionieren.

/F400/ Wartungsfreundlichkeit und Erweiterbarkeit der Komponenten verbessern:

Die Wartungsfreundlichkeit der Komponenten, die durch diese Arbeit verändert werden, soll verbessert werden. Damit soll auf Änderungen von Geräten oder neues Verhalten in Webapplikationen reagiert werden können. Außerdem soll durch eine verbesserte Erweiterbarkeit das Eingliedern von weiteren Plattformen erleichtert werden.

3.2 Qualitätsmerkmale

Nachfolgend werden die Qualitätsmerkmale nach ISO/IEC 9126–1 beschrieben. Die Abbildung 3.1 fasst diese zusammen, wie von Belzert vorgeschlagen. Diese Merkmale beschreiben den Fokus, welcher bei der Entwicklung gesetzt werden sollte. [Bal09, S. 468ff]

Systemqualität	sehr gut	gut	normal	nicht relevant
Funktionalität			x	
Zuverlässigkeit		x		
Benutzbarkeit				x
Effizienz			x	
Wartbarkeit		x		
Portabilität	x			

Abbildung 3.1: Qualitätsanforderungen nach ISO/IEC 9126–1

Die **Funktionalität** wird als „normal“ bewertet. Die Software soll die Anforderungen angemessen erfüllen, befindet sich jedoch in der BETA Phase, Abweichungen sind also akzeptabel, die grundlegende Funktionalität steht im Fokus. Die Genauigkeit spielt eine geringe Rolle, ähnlich wie die Interoperabilität. Die Komponenten mit denen interagiert werden muss sind klar definiert, ein Zusammenspiel mit anderen Komponenten ist nicht geplant. Die Genauigkeit kann nicht hoch bewertet werden, da Komponenten wie die KI, sowie Treiber nicht sehr genau sind. Die Sicherheit der Software soll aktuelle Standards nicht verletzen, sie steht allerdings nicht im Fokus.

Die **Zuverlässigkeit** wird als „gut“ bewertet. Die Software wird auf reellen Geräten ausgeführt, auf welchen nicht vorhersehbare Fehler auftreten können. Diese sollen bedacht werden, damit eine gewisse Fehlertoleranz und Reife gegeben ist. Eine Wiederherstellung eines bestehenden Standes ist nicht notwendig.

Die **Benutzbarkeit** ist irrelevant, da die Software von keinem externen Nutzern verwendet wird, jedoch ein einheitliches Nutzerinterface besitzt, welches die Software steuert (Siehe Abschnitt 2.2.7).

Effizienz spielt eine „normale“ Rolle. Das Zeitverhalten sollte nicht unnötig schlecht ausfallen, allerdings ist die generelle Interaktion mit den mobilen Geräten langsam. Auf ein Verbrauchsverhalten ist nicht zu achten, da die Ausführungsumgebung *MobileDeviceCloud* uneingeschränkt genutzt werden kann.

Wartbarkeit bedarf für die Software mehr Aufmerksamkeit und wird mit „Gut“ bewertet. Wie bereits bei der Funktionalität erwähnt, befindet sich die Software in der BETA Phase, sodass Änderungen zu jeder Zeit vorgenommen werden können. Aus gleichem Grund muss die Software analysierbar sein und einen stabilen Status besitzen, damit die Weiterentwicklung einfach möglich ist.

Die **Portabilität** steht bei der Software im Fokus und wird mit „Sehr gut“ bewertet. Nicht nur muss die Software sehr anpassbar sein, damit sie auf verschiedensten Architekturen funktioniert, auch verändern sich genau diese Architekturen selbst oder können ausgetauscht werden. Die *MobileDeviceCloud* wird auch von anderen unabhängigen Nutzern verwendet. Das heißt, die Software muss Koexistierten tolerieren.

3.3 Entwürfe

Nach dem zum einen die Anforderungen ausgearbeitet wurden und er Fokus bei der Programmierung durch die verschiedenen Qualitätsmerkmale vorgegeben ist, kann die eigentliche Umsetzung geplant werden. Dazu werden die Stellen im bestehenden System angesprochen, an denen gearbeitet wird und was an diesen Stellen geändert wird.

Ein Großteil der benötigten Systemänderungen werden sich in dem *Environment* und der *Crawler* Komponente abspielen, da diese zum einen die zentralen Komponenten sind und zum andern die Schnittstelle zum eigentlichen Gerät darstellen. Änderungen in anderen Komponenten sind nicht ausgeschlossen, nach einer ersten Analyse aber nicht ersichtlich.

Zur Umsetzung der Anforderung **/F010/** muss der Treiber über die *Desired Capabilities* anders konfiguriert werden, damit statt Android- oder Desktop-Geräten iOS-Geräte angesteuert werden. Hierbei ist auf Anforderung **/F300/** zu achten. Dafür bietet es sich an, zum einen das aktuelle *Environment* zu abstrahieren und eine erweiterte Klassenstruktur einzuführen. Zum anderen sind die *Desired Capabilities* aber Konfigurationsdaten, die zwar in einer Klassenstruktur systematisch aufgebaut werden können, jedoch eher eine zusammenhängende Datenstruktur sind. Es verschlechtert zum einen die Lesbarkeit des Codes, solche Datenstrukturen über die verschiedenen Hierarchieebenen zu verteilen und zum anderen erschwert es das individuelle Anpassen der Struktur zu einem späteren Zeitpunkt.

Daher wäre es vom Vorteil speziell für die Generierung der *Desired Capabilities* auf ein Design Pattern zurückzugreifen. Das Strategy Design Pattern hätte hier den Vorteil, die eben beschriebenen Probleme zu lösen. Die Struktur wird an einem Ort generiert. Sie wird in einer Art und Weise generiert, welche ohne Umbauen der Programmstruktur anpassbar ist und sogar zur Laufzeit abgeändert werden kann. Dem Entwickler wird außerdem die Möglichkeit gegeben, eigene Profile zum Testen zu generieren, ohne dass andere manipuliert werden müssen. Es würde zwar der Nebeneffekt der Codedopplung auftreten, was jedoch für Konfigurationsstrukturen akzeptabel ist. Bei Funktionen oder ähnlichem ist das anders, da diese vermehrt im Programmablauf verwendet werden und sich somit die Wartbarkeit des Programmes deutlich verschlechtern würde. Das Verwenden dieses Design Patterns würde demzufolge auch die Anforderung **/F400/** unterstützen.

Zuvor wurde die Erweiterung der bestehenden Klassenstruktur erwähnt, diese Entscheidung ist allerdings nicht trivial und soll folgend evaluiert werden. Wie im Abschnitt 2.2.6 beschrieben, wird für die Interaktion mit den anzusteuernden Geräten das *Selenium* Framework genutzt. Außerdem definiert die Anforderung **/F300/**, dass weiterhin selbiges Framework genutzt werden soll, respektive der *Appium* oder SeeTest Treiber.

Zur Klassenstruktur selbst gibt es wenig Spielraum, wie eine weitere Plattform integriert werden kann. Die aktuelle Struktur besteht, wie bereits in Kapitel 2.2.6 beschrieben und in Abbildung 2.5 veranschaulicht, aus zwei Hierarchieebenen: der Elternklasse und 3 Kindklassen, welche Spezialisierungen für Desktop, Android Web und Android native Plattformen darstellen. Auffallend bei dieser Struktur ist, dass es zwei Kindklassen gibt, welche thematisch mobile Plattformen und Desktop umfassen. In Anbetracht, dass mindestens eine weitere Klasse im mobilen Bereich hinzugefügt wird, die Klasse für iOS-Geräte, erscheint es sinnvoll, eine weitere Hierarchieebene hinzuzufügen, um das mobile Themengebiet besser von dem Desktopgebiet abzugrenzen. Diese Struktur ist in Abbildung 3.2 veranschaulicht. Es könnte außerdem sinnvoll sein, den nativen Bereich von dem Webbereich im mobilen Bereich zu unterscheiden. Dies würde eine weitere Hierarchieebene bedeuten. Dieses Model ist in Abbildung 3.3 noch einmal veranschaulicht.

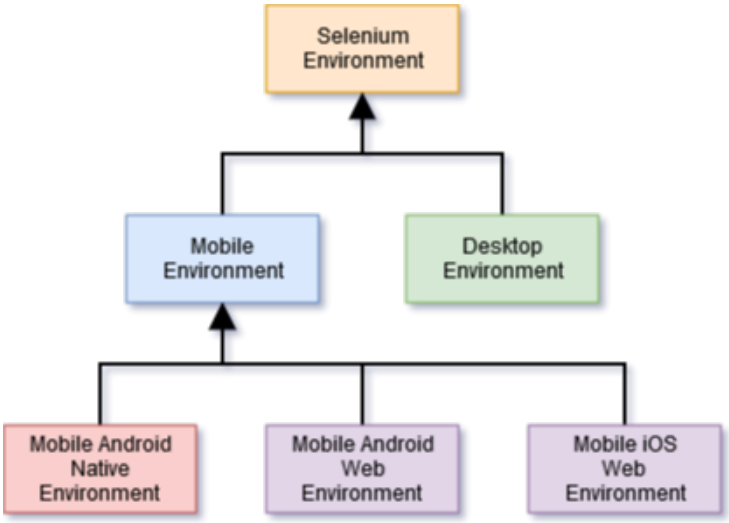

Abbildung 3.2: Strukturdiagramm Enviroment Entwurf 1

Anders formuliert heißt das, egal welches *Environment* genutzt wird, ob für Desktop, Android oder zukünftig iOS, im Kern enthält es einen *Selenium* Treiber und die

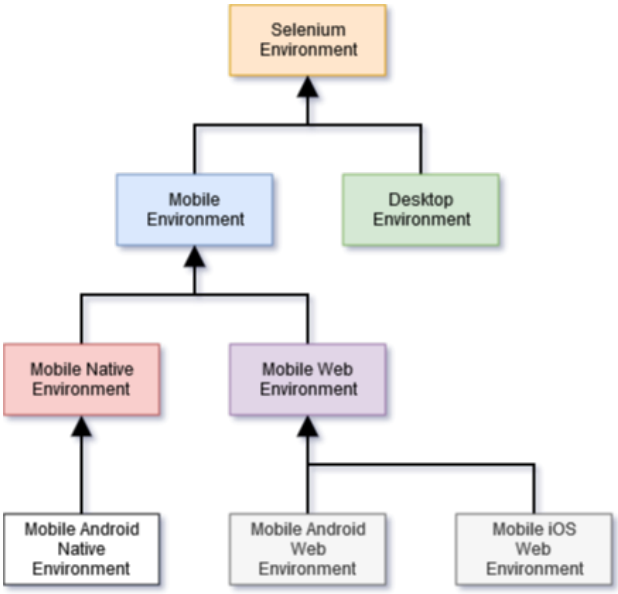

Abbildung 3.3: Strukturdiagramm Enviroment Entwurf 2

Daten um diesen Treiber zu initialisieren. Ein jedes solches *Environment* ist also ein „Selenium Environment“. Wie in Abschnitt 2.3.2 beschrieben stellt das eine Vererbung dar. Wie auch schon in der aktuellen Klassenstruktur zu sehen, spezialisieren die Kindklassen die Elternklasse „Selenium Environment“, welches die grundlegenden Funktionen zur Interaktion bereitstellt. Daraus kann geschlussfolgert werden, dass die Vererbungsstruktur erweitert werden muss, damit iOS-Geräte neben den anderen Plattformen integriert werden können. Der Vorteil davon ist unter anderem, dass die individuellen Unterschiede, welche bei der Implementierung der Plattformen beachtet werden müssen, voneinander unabhängig sind. Sie können in der Kindklasse implementiert werden, ohne dass sich diese auf andere Kindklassen auswirken. Dadurch wird die Funktionalität der anderen Kindklassen nicht eingeschränkt, was in Anforderung **/300/** definiert ist. Des Weiteren können trotzdem die gemeinsamen Funktionen in der Elternklasse implementiert sein, sodass keine Codedopplung auftritt und diese Funktionen weiterhin wartbar und übersichtlich geordnet bleiben. Andere Optionen, wie des Besitzes, also „Das Selenium Environment HAT ein spezifisches Environment“ sind nicht zielführend. Die Komponenten wären nicht unabhängig untereinander und es gäbe zwangsläufig Codedopplung. Auch das zuvor erwähnte Strategy Design Pattern ist auf Ebene des *Environments* nicht sinnvoll. Jedoch ist die Umsetzung dieses Patterns zur Integration der *Environments* in den *Crawler* denkbar.

Die Entscheidung zwischen den Modellen, welche in Abbildung 3.2 und 3.3 veranschaulicht werden, sollte anhand der Kriterien gefällt werden, wie sehr sich die nativen *Environments* von den *Webenvironments* unterscheiden und ob weitere mobile *Environments* für das AI4Test-Projekt geplant sind. Fachlich korrekt wäre das native vom *Webenvironment* zu trennen, hier besteht aber die Gefahr zu viel Komplexität zu schaffen, sodass es durchaus sinnvoll sein kann, keine extra Hierarchieebene einzuführen.

Viele der in Abschnitt 3.1 beschriebenen Anforderungen werden durch die zuvor beschriebene Klassenstruktur gelöst, da sie auf grundlegenden Funktionen des *Selenium* Frameworks basieren oder die Komponenten, mit denen interagiert wird, keine plattformspezifischen Daten erwarten. Es kann trotzdem zu Fehlern kommen, auf diese muss reagiert werden. Solch ein Fall zeichnet sich bereits nach ersten Analysen der iOS spezifischen Treiber ab und wird nachfolgend beschrieben.

Damit durch eine Webseite gescrollt werden kann (**/F100/**), muss zunächst die aktuelle Methode bewertet werden, welche auf Android-Geräten verwendet wird. Diese Methode muss mit den verfügbaren Methoden für iOS-Geräte abgeglichen werden. Falls die verwendete Methodik nicht für iOS-Geräte verfügbar ist, muss eine Alternative gefunden werden. Eine Möglichkeit hierfür könnte statt einer Scrollmethode, die Nachempfindung der Scrollgeste sein, also das Tippen und Halten am unteren Bildschirmrand und nach oben Ziehen des Fingers. Hierbei müsste geprüft werden, ob ein Kontextwechsel notwendig ist, aus dem Webkontext zu dem nativen Kontext. Das Problem des Scrollens kann als stellvertretendes Beispiel für andere native Interaktion mit dem iOS-Gerät gesehen werden. Es ist möglich, dass weitere native Anweisungen ausgeführt werden müssen, welche auf den anderen Architekturen anders implementiert sind und daher anders gehandhabt werden müssen. Das Vorgehen für solche Vorgänge bleibt gleich: Äquivalente Anweisungen für die nicht implementierte zu finden.

3.4 Vorgehen

Das Entwickeln in bereits bestehenden und auch komplexen Systemen stellt den Entwickler vor besondere Herausforderungen. Nicht nur ist das gesamte System schwer zu überschauen, auch kann es zu unvorhersehbaren Fehlern an unerwarteter Stelle kommen, da Komponenten sehr stark miteinander verknüpft sind. Um diesen

Problemen aus dem Weg zu gehen, kann der Entwickler verschiedene Strategien verfolgen.

Das Vorgehen für die Entwicklung ist wie folgt geplant: Zunächst sollte geprüft werden, ob iOS-Geräte ansprechbar sind. Dies kann experimentell geschehen, da noch kein finaler Code entsteht.

Ist dieser Schritt erledigt, muss das *Environment* umgebaut werden, wie es in Abbildung 3.2 und 3.3 vorgeschlagen wird. Dies schafft die Grundlage, aufkommende Probleme für die spezifische Implementierung für iOS-Geräte zu erkennen, ohne andere Implementierungen zu beeinträchtigen. Dabei ist darauf zu achten, dass der Umbau selbst zu keinen Fehlern führt.

Sobald die unabhängige Implementierung möglich ist, kann sequenziell jeder Crawlerschritt implementiert werden. Es bietet sich an, dies in Reihenfolge zu tun, da die Schritte jeweils voneinander abhängig sind und ein anderes Vorgehen die Fehlersuche stark erschwert. Außerdem kann der Fortschritt einfach festgehalten und überblickt werden, da er durch den aktuell zu implementierten Crawlerschritt repräsentiert wird.

Das Isolieren von Problemen hilft dabei, nicht von dem komplexen System überwältigt zu werden. Aus Programmierersicht heißt das, das Problem mit minimalem Code nachzustellen, um es aus dem komplexen System zu isolieren. Dadurch muss nur noch ein kleines überschaubares System analysiert werden, um das Problem zu lösen. Die Erkenntnisse, welche in dem isolierten System gewonnen werden, können häufig direkt in das originale, aber komplexe System übertragen werden und somit das Problem auch dort lösen oder zumindest mehr Auskunft darüber geben.

Dieses Vorgehen kann besonders hilfreich sein, wenn dem Entwickler nicht alle Komponenten bekannt sind oder kein tiefgreifendes Verständnis der anderen Komponenten vorhanden ist. Denn ist das Problem isoliert und auf minimalem Code nachstellbar, hilft es auch anderen Entwicklern weiter, welche versuchen, das Problem zu lösen, da das Problem bereits auf das Wesentliche reduziert wurde.

4 Implementierung

Dieses Kapitel wird sich mit der Implementierung der zuvor beschriebenen Lösungsvorschläge beschäftigen. Dabei soll der Entwurf aus dem Abschnitt 3.3 umgesetzt werden und die Anforderungen aus Abschnitt 3.1 erfüllt werden. Die Anforderungen sollen im nächsten Kapitel durch Tests überprüft werden.

Die Implementierung wird nach dem Vorgehen stattfinden, welches zuvor in Kapitel 3.3 beschrieben wurde.

4.1 Desired Capabilities

Der erste Schritt bei der Implementierung ist, initial die Verbindung zu einem iOS-Gerät auf der *MobileDeviceCloud* herzustellen. Hierfür muss die Konfiguration zur Interaktion angepasst werden. In diesem Projekt sind die *Desired Capabilities* die Konfiguration.

Die Konfiguration besteht aus Authentifizierungsdaten für die *MobileDeviceCloud*, Ablauf, Regelungen für Standardverhalten und am wichtigsten, die Gerätekonfiguration und Auswahl, anhand derer ein Gerät aus dem zur Verfügung stehenden Pool der *MobileDeviceCloud* ausgewählt wird.

Die DeviceQuery wählt ein Gerät und ist mit einem Filter vergleichbar. Sie ist selbsterklärend und kann einfach durch iOS spezifische Beschreibungen ersetzt werden. Für das Projekt hat sich folgender Wert ergeben:

```
@os='ios' and contains(@modelName, 'iPhone 8') and @version='14.1'
```

Es wird das Betriebssystem iOS spezifiziert, sowie die Geräteauswahl auf das Model iPhone 8 mit Betriebssystem 14.1 begrenzt. Die letzten beiden Werte sind nicht

notwendig, schaffen jedoch eine konstante Ausführungsumgebung, welche zum Entwickeln hilfreich ist.

Um das Feld BrowserName mit dem korrekten Wert zu versehen, hilft ein Blick in die offizielle *Appium* Dokumentation, gegebenenfalls *Appium* xcuitest Treibers. [Fou21] [src21] In dieser wird der Wert „Safari“ vorgegeben. Zusätzliche Felder wie „chromeOptions“ wurden aus den *Desired Capabilities* entfernt.

Nach diesen Schritten sind iOS-Geräte auf der *MobileDeviceCloud* ansteuerbar.

Das Implementieren des Design Patterns ist der nächste Schritt. Dafür wird der Konstruktor der *Environment* Klassen mit einem weiteren Parameter versehen, mit welchem ein Profil definiert wird. Dieser Parameter ist vom Typ des Generator-Interfaces, welches zur Umsetzung des Strategy Design Patterns benötigt wird. Für jedes *Environment* wird nun ein dazugehöriges Profil angelegt, welches das zuvor erwähnte Interface implementiert. In diesem Profil werden die spezifischen *Desired Capabilities* definiert. Abbildung 4.1 veranschaulicht das Strategy Design Pattern noch einmal. Wobei die abgebildeten Profile nur Beispiele sind. Es können beliebig viele Profile angelegt und verwendet werden.

Durch das Umsetzen des Patterns konnten einige der Klassenvariablen eliminiert werden, da sie nur für die Generierung der *Desired Capabilities* notwendig sind.

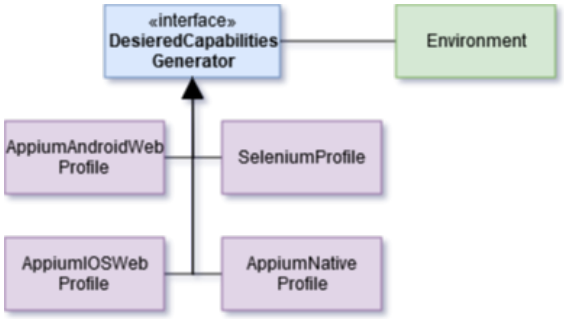

Abbildung 4.1: Strukturdiagramm Strategy Design Pattern

4.2 Klassenstruktur

In Kapitel 3.3 wurden bereits zwei Architektur-Möglichkeiten vorgestellt, welche implementiert werden können. In Vorbereitung dafür muss die bestehende Struktur dahingehend analysiert werden, dass die Zuordnung zur jeweiligen Hierarchieebene klar ist. Die Hierarchieebenen wurden bereits in den Abbildungen 3.2, 3.3 veranschaulicht.

Für die Analyse kann ein Klassendiagramm des *Environments* zu Hilfe genommen werden. Ein solches Klassendiagramm ist in Abbildung 4.2 zu sehen. In diesem Diagramm sind bereits einige Eigenschaften hervorgehoben. Dazu zählen gemeinsame Attribute und Funktionen (Blau), sowie explizite Unterschiede (Grün oder Fett) zwischen der Android- und iOS-Architektur.

Es wurden zwei verschiedene Entwürfe in Kapitel 3.3 vorgestellt. Während der erste Entwurf versucht Komplexität zu minimieren, setzt der zweite Entwurf auf Erweiterbarkeit und Generalisierung. In Anbetracht dessen, dass das AI4Test-Projekt plant, weitere neue Architekturen neben iOS zu integrieren (smart Devices), erscheint der zweite Entwurf attraktiver. Auch mit dem Blick auf die bestehende Struktur erscheint es sinnvoll, den zweiten Entwurf umzusetzen. Funktionen wie „check_frames_for_cookies“, „handle_widget_popups“ und weitere, sind Gemeinsamkeiten, für die ein generalisiertes *WebEnvironment*, also eine weitere Hierarchieebene, spricht.

Bei der Implementierung wurden einige Klassen abstrakt definiert. Darunter die Klassen *Selenium-Environment*, Mobile-*Environment* und Mobile-Web-*Environment*. Eine Instanziierung der Klassen ist nicht sinnvoll, vielmehr vereinen diese Klassen gemeinsame Methoden und Attribute.

Mit der umgesetzten neuen Klassenstruktur muss auf die Initialisierung des *Environments* geachtet werden. Bei der Initialisierung von Kindklassen muss der Konstruktor der Elternklasse aufgerufen werden, in der bestehenden Struktur, geschah der Aufruf inmitten des Kind-Konstruktors. Dies hat zur Folge, dass Veränderungen sehr schnell zu fehlerhaftem Verhalten an unerwarteter Stelle führen kann, da die Konstruktor-Prozedur sehr unübersichtlich ist.

Das Problem der Unübersichtlichkeit kann einfach umgangen werden, indem der Konstruktor der Elternklasse gleich als Erstes im Konstruktor der Kindklasse auf-

gerufen wird. Somit wird die Initialisierung von der Elternklasse startend bis zur Kindklasse endend ausgeführt, was deutlich übersichtlicher ist.

Um die beschriebene Strategie umzusetzen, sind lediglich die bereits beschriebenen Änderungen zu beachten und, falls nötig, die Initialisierung von Variablen anzupassen. Die finale Struktur des *Environments* ist durch die Abbildung 4.3 als Klassendiagramm dargestellt.

Abbildung 4.2: Klassendiagramm des alten Environments

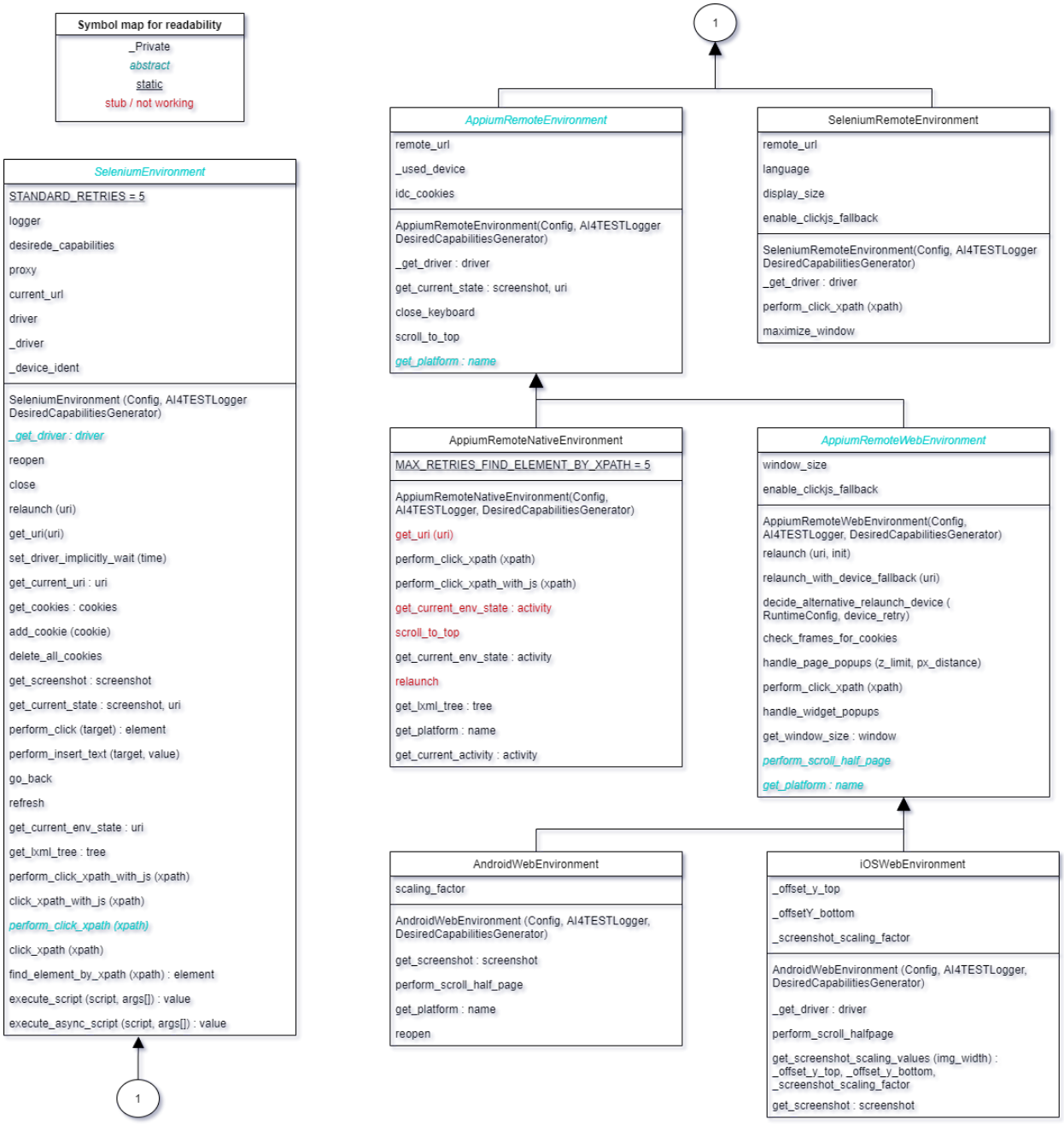

Abbildung 4.3: Klassendiagramm des neuen Environments

4.3 Scroll

Um durch eine Webseite zu navigieren, muss unter anderem gescrollt werden. Im AI4Test-Projekt wird versucht, immer um die Hälfte der aktuell angezeigten Seite zu scrollen. Das soll dem *ElementFinder* ermöglichen, alle Elemente auf einer Seite mindestens einmal vollständig angezeigt zu bekommen.

In dem Android spezifischen *Environment*, findet eine sogenannte „TouchAction“ Anwendung. Diese ist eine von *Appium* bereitgestellte Klasse, welche dem Nutzer die Verwendung von treibernahen Anweisungen erleichtern soll. Die Klasse bietet verschiedene Funktionen an, darunter die Emulierung von Scroll-Gesten. Dabei kann die genaue Scrollweite definiert werden. Wird eine solche TouchAction, insbesondere die Funktion zum Scrollen, unter iOS-Geräten verwendet, tritt ein Fehler auf. Der Fehler indiziert, dass der *XCTest* Treiber die genutzten Funktionen nicht implementiert. Um das Scrollen trotzdem zu implementieren, muss die Scroll-Geste anders emuliert werden. Im Detail besteht die Geste aus dem Ansetzen des Fingers an einer definierten Position, dem Bewegen des Fingers und dem anschließenden Loslassen des Displays an einer bestimmten Position.

Die *Appium XCTest* Dokumentation [src21] bietet verschiedene Ansätze, diese Geste nachzuempfinden. Dazu gehören: „mobile: swipe“, „mobile: scroll“ und „mobile: dragFromToForDuration“. Auch wenn die ersten zwei Funktionen auf den ersten Blick nach der Lösung des Problems aussehen, sind sie ungeeignet. Beide bieten keine Möglichkeit, präzise zu scrollen. Der Swipe ist von der Weite nicht kontrollierbar und der Scroll selbst, scrollt über genau eine Seite. Die letzte Funktion „ziehe von, nach, in Zeit“ erscheint zwar auf den ersten Blick ungeeignet, erlaubt jedoch mit den richtigen Einstellungen genaues Scrollen. Es bleibt eine gewisse Ungenauigkeit bestehen, dadurch dass die Geste schnell durchgeführt wird und dadurch das Scrollen anschließend noch „abbremst“ oder „weiter fließt“.

4.4 JavaScript Ausführung

Viele der Komponenten, welche im AI4Test-Projekt verwendet werden, greifen auf das Ausführen von JavaScript auf den Geräten zurück. Dazu gehören Komponenten, welche Standardverhalten bei PopUps oder Cookie-Banner behandeln, aber auch

Komponenten, welche essenziell für den Ablauf des Projektes sind. Dazu gehören unter anderem der *xPathGenerator*. Diese Komponente verarbeitet die Ergebnisse des *ElementFinders* zu xPath Angaben und ist unabdingbar für die erfolgreiche Ausführung des Projektes. Bei der Verwendung der eben genannten Komponenten tritt jeweils ein Fehler auf, welcher die korrekte Ausführung verhindert. Das heißt PopUps können nicht korrekt behandelt werden, Cookie-Banner nicht geschlossen werden und der *xPathGenerator* nicht verwendet werden. Die Fehlermeldung, welche bei dem Debuggen auftritt, ist sehr generisch und gibt keinen Aufschluss über den Ursprung des Problems, dass JavaScript nicht richtig ausgeführt werden kann. Entweder wird indiziert, dass in einem leeren JavaScript-Code ein Syntaxfehler vorliegt oder, dass ein unbekannter serverseitiger Fehler aufgetreten sei.

Um auszuschließen, dass iOS-Geräte generell solche JavaScripts nicht ausführen können, wurde eine eigene Webseite verwendet, welche durch Betätigen eines Knopfes JavaScript Code ausführt. Der hinterlegte Code besteht aus den nötigsten Teilen des *xPathGenerator*. Der Test führte zu keinem Fehler. Der Gedanke, dass die Webseite den JavaScript-Code bereits beinhaltet, wurde weiterverfolgt. Dafür wurde der Code auf einem Webserver für die Geräte der *MobileDeviceCloud* zugänglich gemacht und zur Laufzeit in den Header der zu untersuchenden Webseite eingebunden. Das Projekt injizierte nun nicht mehr den gesamten JavaScript Code, sondern rief nur noch die nötigen Methoden auf. Diese Methode führte allerdings zu den gleichen initialen Problemen, dass das Ausführen von JavaScript zu Fehlern führt.

Nun wurde versucht auszuschließen, dass das Problem durch die *MobileDeviceCloud* verursacht wird. Dazu wurde versucht, ein Gerät direkt über *Selenium* anzusprechen, ohne die *MobileDeviceCloud* zu verwenden. Dazu ist ein Apple-Gerät mit der XCode Software notwendig, sowie ein Entwicklerkonto bei dem Appledienst „iTunes“. Bei der Versionsprüfung der notwendigen Software wurde ebenfalls die *Selenium* Version des *MobileDeviceCloud* überprüft und auf den neusten Stand gebracht. Die installierte Version „1.18“ wurde auf die Version „1.20.2“ aktualisiert. Dieses Update löste die beschriebenen Probleme, dass JavaScript Code nicht korrekt ausgeführt werden konnte und dadurch Komponenten wie der *xPathGenerator* nicht funktionieren. Der Versuch iOS-Geräte direkt anzusprechen wurde fallen gelassen, da das Anbinden nicht trivial ist und nicht weiter zielführend wäre.

Mit dem Update der Appiumversion auf der *MobileDeviceCloud* lösten sich außerdem weitere Probleme, wie das Auslesen von Cookies.

4.5 Screenshot

Die Generierung von Bildschirmfotos (Screenshots) ist essenziell für das AI4Test-Projekt. Sie werden vom *ElementFinder* verwendet und dienen zur Dokumentation sowie zum Debuggen.

Die von *Appium* zur Verfügung gestellte Methode zum Erstellen von Screenshots unterscheidet sich zwischen iOS- und Android-Geräten. Während auf Android-Geräten lediglich die Webseite selbst dargestellt wird (solange sich der Treiber in der WebView befindet), ist auf iOS-Geräten der gesamte Bildschirminhalt abgebildet. Neben der Webseite sind also auch Systemelemente, wie die Suchleiste oder ein vor und zurück Knopf zu sehen. Während diese Systemelemente für Dokumentationszwecke oder zum Debuggen des Projektes nicht stören, führen sie beim *ElementFinder* zu Problemen. Bei der Ausführung führt das Vorhandensein von nicht Webseitenelementen zu Problemen, da diese vom *Elementfinder* gefunden werden und als Webseitenelemente mit aufgenommen werden. Wird versucht auf ein Systemelement zu Klicken, führt das zu Fehlern, da sich das Element in einem anderen Kontext als dem WebView befindet, also nicht Teil der Webseite selbst ist.

Der Versuch das Problem zu lösen, indem der Treiber explizit dazu aufgefordert wird, von dem BrowserFrame ein Screenshot zu erstellen, schlug fehl, da diese Funktion nicht von *Appium* für iOS implementiert ist, beziehungsweise *XCTest* eine solche Funktionalität nicht ermöglicht.

Eine zielführende Lösung ist, die erstellten Screenshots zurechtzuschneiden und zu skalieren. Die Skalierung ist notwendig, da die Screenshots in einer anderen Auflösung erstellt werden als das Gerätedisplay anzeigt. Die Verzerrung der beiden Auflösungen führt dazu, dass Elemente an falscher Position vermutet werden und versucht wird an Positionen außerhalb des WebView Kontextes zu Klicken.

Für das Zurechtschneiden der Screenshots muss die Größe der oberen und unteren Systemelemente ermittelt werden. Damit dies möglich ist, muss ein Kontextwechsel stattfinden, da der WebView die Systemelemente nicht kennt. Im richtigen Kontext können die Systemelemente über ihre Namen „TopBrowserBar“ und „BottomBrowserToolbar“ abgerufen werden. Die Höhe muss, wie der Screenshot selbst, noch einmal skaliert werden, da sich der native Kontext ebenfalls in der Auflösung vom WebView unterscheidet. Zur Skalierung kann das Verhältnis von Screenshotbreite

und Browserbreite verwendet werden. Die Skalierung des Screenshots kann über das Verhältnis der Webseitenbreite und der Screenshotbreite geschehen.

Abbildung 4.4: Visualisierte Skalierung

Die Abbildung 4.4 visualisiert das Problem der Skalierung. Die roten Punkte stellen die Originalwerte des *Elementfinders* dar. Es sind nur drei Punkte zusehen, da sich die restlichen Punkte außerhalb des aktuellen Kontextes und Sichtfeldes befinden. Die blauen Punkte stellen die korrigierten Werte dar. Die Systemelemente wurden bereits weggeschnitten und werden daher nicht erkannt. Abbildung 2.3 in Kapitel 2.2.2 hatte diesen Umstand bereits gezeigt.

5 Evaluation

Dieses Kapitel bewertet die Änderungen am AI4Test-Projekt. Dazu werden die in Abschnitt 3.1 beschriebenen Anforderungen evaluiert. Dazu wird gesondert die Architekturanpassung, sowie der Vergleich von der Plattform iOS und Android evaluiert.

5.1 Tests

Anforderung	F010 – iOS-Gerät muss über *MobileDeviceCloud* ansteuerbar sein.
Test	Initialisieren des Treibers, anschließende Kontrolle auf der *MobileDeviceCloud*.
Erwartung	Session für iOS-Gerät wurde eröffnet.
Ergebnis	iOS-Session wurde eröffnet, Gerät ist ansteuerbar.
Anforderung	F020 – Webseite muss aufrufbar sein.
Test	Initialisieren des Treibers, anschließendes aufrufen einer Webseite.
Erwartung	Browser wird geöffnet, Webseite wird aufgerufen und angezeigt.
Ergebnis	Beliebige Webseiten sind aufrufbar, Safari startet, wenn nicht bereits gestartet.
Anforderung	F100 – Webseite muss scrollbar sein, um größere Webseiten anzusehen.
Test	Scrollbare Webseite wird aufgerufen, Scrollanweisung wird ausgeführt.
Erwartung	Ansicht der Webseite verschiebt sich um die Hälfte der Bildschirmgröße.
Ergebnis	Es wird ungefähr um die Hälfte der Bildschirmhöhe gescrollt, allerdings nicht genau.

Anforderung	F110 – Klicken auf Element.
Test	Webseite mit klickbarem Element wird aufgerufen, Klickanweisung für Element wird ausgeführt.
Erwartung	Element wird angeklickt.
Ergebnis	Menüs, Links oder Buttons konnten erfolgreich angewählt werden.
Anforderung	F120 – Screenshots von einer Webseite müssen erstellt werden können.
Test	Webseite wird aufgerufen, Screenshot wird erstellt.
Erwartung	Bild der Webseite wurde erstellt, keine Systemelemente sind zu sehen.
Ergebnis	Erstellung von Screenshots ist erfolgreich, keine Systemelemente werden angezeigt. Das gilt für erweiterte Suchleiste und minimierte Suchleiste.
Anforderung	F200 – Komponenten müssen funktionieren.
Test	Durchlauf des Projekts wird gestartet.
Erwartung	Ablauf beendet erfolgreich.
Ergebnis	Ablauf beendet erfolgreich.
Anforderung	F210 – Default PopUps müssen behandelt werden können.
Test	Webseite mit PopUp-Verhalten wird aufgerufen, bei Auftreten des PopUps wird die PopUp Behandlung durchgeführt.
Erwartung	Das PopUp wird geschlossen oder versteckt.
Ergebnis	PopUps konnten erfolgreich behandelt werden.
Anforderung	F220 – Cookies auf dem Gerät müssen manipuliert werden können.
Test	Webseite mit Cookie-Verhalten wird aufgerufen, Cookie-Stand wird definiert und nach Neuladen wiederhergestellt.
Erwartung	Cookies werden gelesen und gesetzt.
Ergebnis	Ein bestimmter Stand an Cookies kann wiederhergestellt werden.
Anforderung	F230 – *PageClusterer*
Test	Erkannte Gruppen auf der gleichen Webseite werden zwischen iOS und Android verglichen.
Erwartung	Eine gleiche Anzahl an Gruppen wird erkannt.
Ergebnis	Auf der Plattform iOS werden signifikant weniger *PageCluster* erkannt.

Anforderung	F240 – *Elementfinder*
Test	Ein Screenshot einer Webseite wird an den *Elementfinder* gesendet.
Erwartung	Der *Elementfinder* akzeptiert die Daten und gibt als Ergebnis die wesentlichen Elemente auf den Screenshot zurück.
Ergebnis	BoundingBoxes werden erfolgreich zurückgegeben, es treten keine Fehler auf.

5.2 Architekturanpassung

Durch die Architekturanpassung sollten mehrere Aspekte an dem AI4Test-Projekt verbessert und erweitert werden. Dazu zählt in erster Linie die Einbindung von der Plattform iOS. Aspekte wie Erweiterbarkeit, Analysierbarkeit und Wartbarkeit sollten ebenfalls verbessert werden.

Durch die Anpassung der Architektur konnte die Erweiterbarkeit und Wartbarkeit deutlich verbessert werden. Es wurde eine Klassenstruktur gewählt, die zusätzliche Hierarchieebenen beinhaltet. Dies ermöglicht es, zusätzliche Komponenten einfacher in das bestehende Projekt einzupflegen. Als Beispiel hierfür können native Applikationen genommen werden. Aktuell ist nur die generelle Klasse „Mobile Native“ vorhanden und die Klasse für native Android-Applikationen indiziert, wie in Abbildung 3.3 in Kapitel 3.3 mit dargestellt. An dieser Stelle ist die Einbindung von nativen iOS-Applikation einfach umsetzbar. Komplett neue Plattformen, wie Internet of Things (IoT)-Geräte oder Wearables können unabhängig eingebunden werden, ohne dass andere Klassen beeinflusst werden. Dafür müsste auf der zweiten Hierarchieebene eine weitere Klasse hinzugefügt werden.

Die Wartbarkeit konnte ebenfalls verbessert werden. Durch die Klassenstruktur kann der Fehler zum Beispiel einfacher und genauer auf eine bestimmte Klasse eingegrenzt werden. Konfigurationsmöglichkeiten, wie die *Desired Capabilities*, können durch das eingesetzte Design Pattern so angepasst werden, dass einfach auf sie zugegriffen werden kann und zum Beispiel Debugging Profile genutzt werden können. Während der Durchführung dieser Arbeit wurden außerdem andere Komponenten angepasst, darunter der *Crawler* selbst, welche eine ähnliche Klassenstruktur übernommen hat wie das *Environment*. Das führt dazu, dass die Navigation im Code einfacher fällt und der Code besser analysiert werden kann.

5.3 Plattform iOS

Die neue Einbindung der Plattform iOS, kann verschieden bewertet werden. Es bietet sich an, gewisse Kenngrößen von Projektausführungen auf iOS und Android Plattformen zu vergleichen. Mögliche Kenngrößen sind: Die Anzahl an Kanten des entwickelten Graphen, welcher während der Projektausführung generiert wird. Diese Kenngröße beschreibt, wie viele ausführbare Aktionen auf der Applikation gefunden wurden. Ausführbare Aktionen können unter anderem Klicks, das Ausfüllen von Textfeldern oder das Prüfen, ob Elemente angezeigt werden sein. Die Anzahl an *PageClustern* ist eine weitere Größe. Sie beschreibt, wie viele Seitengruppen auf der gesamten Applikation gefunden wurden. Außerdem kann die Anzahl an generell gefundenen Elemente auf der Applikation ausgewertet werden, im Grunde also, wie viele Elemente die KI des Projektes identifizieren konnte. Neben den genannten Kenngrößen kann außerdem die Ausführungszeit zur Evaluation verwendet werden. Als letzte zwei Kenngrößen können die generierten Testfälle und Skripte verwendet werden. Hier kann zum einen die Anzahl der Test-Skripte verwendet werden und zum andern die Anzahl der Elemente, die schlussendlich überprüft wird. Mit diesen beiden Werten kann gut das eigentliche Ergebnis einer Projektausführung eingeschätzt werden.

Die Abbildung 5.1 zeigt die eben genannten Kenngrößen im Vergleich von iOS- und Android-Testläufen auf drei verschiedenen Webseiten. Das Projekt wurde je Plattform jeweils zweimal, mit gleichem Umfang ausgeführt. Durch die doppelte Ausführung kann ein grober Überblick gegeben werden, wie konsistent die Ausführungen sind.

Der Fakt, dass diese Daten generiert werden können, beweist allein, dass die Erweiterung der Architektur um die Plattform iOS funktioniert hat. Die Generierung wäre nicht möglich gewesen, wenn das Projekt nicht erfolgreich gestartet und abgeschlossen werden kann.

Mit erstem Blick auf die Daten fällt auf, dass mit Ausnahme der Daten für die Webseite „nrw.de“, ein deutliches Defizit zwischen iOS und Android besteht. Android findet deutlich mehr Elemente und *PageCluster* und damit auch mehr Aktionen. Die generierten TestSkripte sind im gleichen Verhältnis größer für Android und prüfen mehr Elemente. Die Projekt Ausführung terminiert deutlich schneller auf der

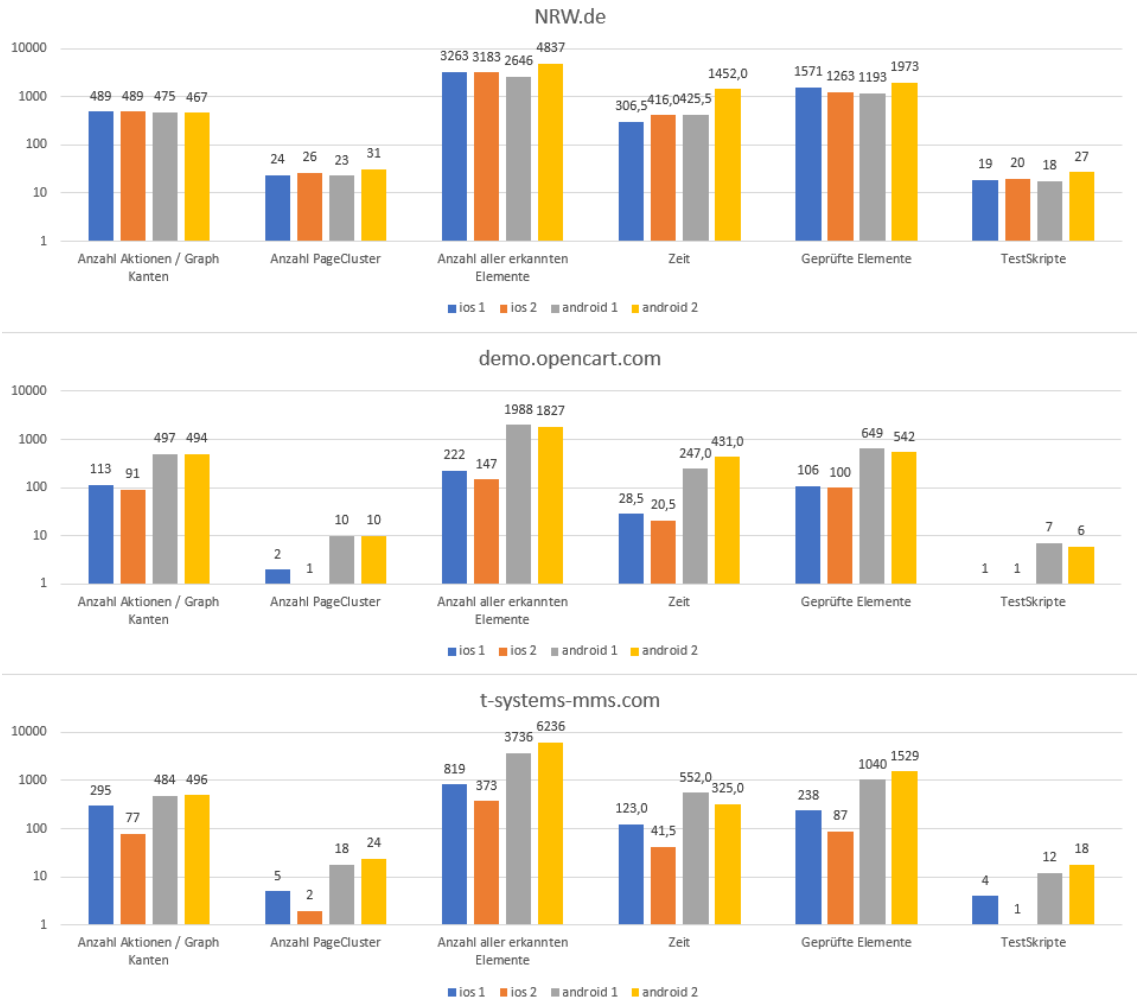

Abbildung 5.1: Vergleich iOS und Android Ergebnisse

Plattform iOS und führt generell auch weniger Schritte aus, bricht also ab, obwohl der maximale Ausführungsumfang noch nicht erreicht wurde.

Dieses Defizit kann mehrere Gründe haben. Der offensichtlichste ist, dass der *ElementFinder* weniger Elemente findet. Werden die gefundenen Elemente stichprobenartig an einer gleichen Stelle, bei einer iOS und Android Ausführung verglichen, kann kein solches Defizit festgestellt werden. Die Abbildung 5.2 zeigt beispielhaft einen solchen Vergleich. Die Ergebnisse sind links aus einer Android und rechts aus einer iOS Ausführung entnommen. Die roten Umrandungen sind die direkten Ergebnisse des *ElementFinders*. Blaue Umrandungen sind die aufgearbeiteten Ergebnisse des *xPathGenerators*.

Abbildung 5.2: Vergleich iOS und Android Ergebnisse des ElementFinders

Ein weiterer Grund für das Defizit könnte die *PageClusterer* Komponente sein. Es wurden nicht nur weniger Elemente, sondern auch weniger *PageCluster* gefunden. Das lässt darauf schließen, dass Seiten gar nicht erst gefunden oder aufgerufen werden. Entscheidet der *PageClusterer* zu grob, werden Seiten mit neuem Inhalt nicht analysiert, da sie fälschlicherweise als bereits bekannt eingestuft werden. Gerade Menüs sind dafür sehr anfällig, da sie wenig an der Seitenstruktur der Webseite verändern. Aber gerade Menüs sind sehr wichtig, um auf neue Seiten zu stoßen. Werden diese nicht analysiert, da sie als bereits bekannt eingeordnet wurden, werden neue Seiten nicht entdeckt und des kann zu dem beschriebenen Defizit kommen.

Für ein Problem mit dem *PageClusterer* spricht der generierte Graph bei der Projektausführung. Dieser speichert, die gefundenen Seiten und wie diese gefunden wurden. Diesem kann entnommen werden, dass Klicks auf Links auf der Webseite, welche definitiv zu einem neuen *PageCluster* führen sollten, im Graphen auf das gleiche *PageCluster* zeigen. Wenn der *PageClusterer* korrekt funktioniert, sollte im Graphen ein Verweis auf ein neues *PageCluster* eingetragen sein.

5.4 Gesamtsystem

Die Architektur des AI4Test-Projektes konnte um die Plattform iOS erweitert werden. Die grundlegende Funktionalität der neuen Plattform ist trotz noch bestehender Defizite sichergestellt, sodass sinnvolle Ergebnisse generiert werden können. Die Ausführung funktioniert auf den verschiedensten iOS-Geräten ab dem iPhone Model „iPhone 7“.

Bei den drei getesteten Webseiten wurden jeweils in zwei unabhängigen Projektausführungen, Testfälle mit ähnlichem Umfang generiert. Werden die Ergebnisse jedoch mit äquivalenten Ausführungen auf der Plattform iOS und Android verglichen, ist das Ergebnis durchwachsen, wie in Abbildung 5.1 zu sehen und im vorherigen Abschnitt beschrieben. Die Webseite „nrw.de“ sticht allerdings positiv heraus. Bei der Projektausführung konnten äquivalent Ergebnisse erzielt werden und die Durchführungszeit war sogar schneller.

Für die Lösung des Problems, dass der *PageClusterer* nicht richtig entscheidet, wird eine weitere und genauere Analyse benötigt. Diese sollte zum einen daraus bestehen, ob eine einfache Schwellenwertanpassung für den *PageClusterer* zielführend ist. Ist dies der Fall, muss ein passenderer Schwellenwert, spezifisch für die Plattform iOS gefunden werden. Zum anderen sollte herausgefunden werden, warum der *PageClusterer* auf der Plattform iOS anders entscheidet als auf der Plattform Android. Beide Plattformen zeigen oberflächlich die gleiche Webseite an und sollten sich nicht unterscheiden. In dieser Hinsicht könnte der DOM-Baum untersucht werden, anhand dessen der *PageClusterer* seine Entscheidung trifft, ob eine Seite zu einem *PageCluster* gehört. Falls bei dem DOM-Baum Unterschiede festzustellen sind, müsste eine Methode gefunden werden, welche diese Unterschiede behandelt, sodass vergleichbare Ergebnisse erzielt werden können.

Außerhalb dieser Arbeit wurden zahlreiche Anpassungen an andern AI4Test-Projekt Komponenten vorgenommen, mit dem Fokus, die Logging und Debugging Möglichkeiten zu verbessern, sowie einen besseren Code Überblick zu schaffen. Diese Änderungen gehen Hand in Hand mit den in dieser Arbeit beschriebenen Änderungen, ergänzen sich nahtlos und tragen zur allgemein verbesserten Wartbarkeit bei, wie es bereits in Abschnitt 5.2 beschrieben wurde.

6 Zusammenfassung und Ausblick

Ziel dieser Arbeit war herauszufinden, ob zum einen für das AI4Test-Projekt die bestehende Schnittstelle zu Geräten generalisiert werden kann und zum anderen, ob das AI4Test um die Plattform iOS erweitert werden kann. Dafür wurde die bestehende Architektur untersucht, mit der Erkenntnis, dass ein Framework zur Interaktion mit den Geräten verwendet wird, welches auch die Plattform iOS unterstützt. Das *Selenium* Framework ermöglicht es über eine generalisierte Schnittstelle mit vielen verschiedenen Geräten zu interagieren. Dies wurde bereits im AI4Test-Projekt genutzt um mit den Plattformen Desktop und Android zu interagieren.

Es wurde festgestellt, dass die bestehende Architektur umgestaltet und generalisiert werden muss, damit die Plattform iOS integriert werden kann und das AI4Test-Projekt für weitere Plattformen vorbereitet ist. Diese Anpassung wurde in der *Environment* Komponente geplant, da diese zur Interaktion mit den Geräten über das *Selenium* Framework zuständig ist und die anderen Komponenten von den Eigenheiten der verschiedenen Plattformen zu großen Teilen abschirmt. Die Architekturanpassung bestand daraus, eine Klassenstruktur mit mehreren Vererbungsebenen einzuführen. Auf diese Weise kann realisiert werden, dass jede Plattform die Möglichkeit hat, Eigenheiten durch Überladen einzelner Methoden zu behandeln, ohne andere Plattformen zu beeinflussen und zugleich eine einheitliche Struktur vorzugeben.

Eigenheiten der Plattform iOS konnten für die Scrollfunktion, das Erstellen von Screenshots und dem Ausführen von JavaScript festgestellt werden. Die Behandlung dieser Eigenheiten konnte dank der geplanten Architekturänderung umgesetzt werden, sodass die Projektausführung auf der Plattform iOS erfolgreich ablaufen kann und sinnvolle Ergebnisse liefert. Es ist jedoch weitere Arbeit notwendig, um äquivalente Ergebnisse zur Plattform Android zu erreichen. Der Grund für das Defizit könnte die *PageClusterer* Komponente sein, welche andere Ergebnisse für die Plattform iOS liefert, als für die Plattform Android.

Diese Arbeit hat eine Übersicht über das AI4Test-Projekt gegeben und eine Architekturanpassung zur Integration einer neuen Plattform beschrieben. Weitere kurzfristige Aufgaben für das Projekt werden die Analyse des *PageClusterer* sein, mit dem allgemeinen Ziel der Verbesserung der Ergebnisse für die Plattform iOS. Ansätze dafür wurden bereits im Kapitel zuvor beschrieben. Weitere längerfristige Aufgaben an dem AI4Test Projekt könnten unter anderem, die Einbindung weiterer Plattformen sein. Die Einbindung der nativen Umgebungen von iOS und Android wurde bereits in der Arbeit angesprochen, zusätzlich könnten neuartige Plattformen, wie zum Beispiel auf Wearables genutzt, ebenfalls infrage kommen.

Glossar

DOM Document Object Model. 10, 47

IoT Internet of Things. 43

KI künstliche Intelligenz. 1, 6, 8, 10, 24, 44

SDK Software Development Kit. 6

UI User Interface. 7

URI Uniform Resource Identifier. 13

Literaturverzeichnis

[Aye18] Sakis Ayeva, Kamon Kasampalis: *Mastering Python design patterns a guide to creating smart, efficient, and reusable software*, 2018.
URL http://slubdd.de/katalog?TN_libero_mab2

[Bal09] Helmut Balzert, Helmut Balzert: *Lehrbuch der Softwaretechnik 1 Basiskonzepte und Requirements-Engineering / Helmut Balzert. Unter Mitw. von Heide Balzert ...*, Spektrum Akad. Verl., Heidelberg, 3. aufl. Aufl., 2009, ISBN 9783827417053.
URL http://slubdd.de/katalog?TN_libero_mab2)1000206319

[Beu07] Hani Beust, CédricSuleiman: *Next generation Java testingTestNG and advanced concepts*, Addison-Wesley, Upper Saddle River, NJ, 2007, ISBN 0321503104.
URL http://slubdd.de/katalog?TN_libero_mab215591811

[Cle10] Torsten Cleff: *Basiswissen Testen von Software Vorbereitung zum Certified Tester (Foundation Level) nach ISTQB-Standard*, W3L-Verl., Herdecke, 2010, ISBN 9783868340129.
URL http://slubdd.de/katalog?TN_libero_mab2)1000074109

[CRKH05] Jonathan Corbet, Alessandro Rubini und Jonathan Kroah-Hartman, Greg Corbet: *Linux device drivers [where the Kernel meets the hardware]*, O'Reilly, Beijing, 3rd ed. Aufl., 2005, ISBN 9780596005900.
URL http://slubdd.de/katalog?TN_libero_mab213998062

[Fou21] JS Foundation: *The Safari Driver*, Sept. 2021, URL: https://appium.io/docs/en/drivers/safari/, besucht am 22.09.2021.

[Gun18] Satya Gundecha, Unmesh Avasarala: *Selenium WebDriver 3 practical guide end-to-end automation testing for web and mobile browsers with*

Selenium WebDriver, Packt Publishing, Birmingham, UK, second edition. Aufl., 2018, ISBN 9781788996013.
URL http://slubdd.de/katalog?TN_libero_mab2

[Her18] Axel Hertzschuch: *GENERIERUNG VON PAGE OBJECTS MITTELS DOM-BAUMANALYSE UND GERICHTETEM CRAWLING*, Dissertation, Technische Universität Dresden, 9 2018.

[Inc20] Digital.ai Software Inc.: , Jan. 2020, URL: https://docs.experitest.com/display/TC/Capabilities, besucht am 08.10.2021.

[Kra07] Jeff Kramer: *Is abstraction the key to computing?*, 2007.
URL http://slubdd.de/katalog?TN_libero_mab2

[Lam19] Erwin Lammenett: *Praxiswissen Online-Marketing Affiliate-, Influencer-, Content- und E-Mail-Marketing, Google Ads, SEO, Social Media, Online- inklusive Facebook-Werbung*, Springer Gabler, Wiesbaden „ 7., überarbeitete und erweiterte auflage Aufl., [2019], ISBN 9783658251352.
URL http://slubdd.de/katalog?TN_libero_mab216645564

[Red16] Joseph Redmon: *Darknet: Open Source Neural Networks in C*, http://pjreddie.com/darknet/ besucht am 31.08.2021, 2013–2016.

[RF18] Joseph Redmon und Ali Farhadi: *YOLOv3: An Incremental Improvement*, *CoRR*, Bd. abs/1804.02767, 2018.
URL http://arxiv.org/abs/1804.02767

[SBS12] Harry M. Sneed, Manfred Baumgartner und Richard Seidl: *Der Systemtest von den Anforderungen zum Qualitätsnachweis*, Hanser, München, 3., aktualisierte und erw. aufl. Aufl., 2012, ISBN 3446426922.
URL http://slubdd.de/katalog?TN_libero_mab215620363

[Sha17] Mukesh Sharma: *Software testing 2020 preparing for new roles*, Taylor & Francis Group, CRC Press„ Boca Raton, [2017], ISBN 1498788882.
URL http://slubdd.de/katalog?TN_libero_mab2)1000688467

[Sol] T-Systems Multimedia Solutions: *Mobile Device Cloud*, URL: https://mobiledevice.cloud/, besucht am 25.08.2021.

[src21] *appium-xcuitest-driver*, Sept. 2021, URL: https://github.com/appium/appium-xcuitest-driver, besucht am 22.09.2021.

[Sta20] StatCounter: , Febr. 2020, URL: https://gs.statcounter.com/, besucht am 29.09.2021.

[Wah19] Matthias Wahl: *Digitale Nutzung in Deutschland: Die Smartphone-Nutzung stieg 2019 werktags um 14 Prozent an, am Wochenende sogar um 19 Prozent / Vor allem Nutzer mittleren Alters treiben Wachstum*, *Bundesverband Digitale Wirtschaft (BVDW) e. V.*, 10 2019, an optional note.

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Mittweida, den 20. Oktober 2021

Ansgar Dabow