Test graphischer Benutzeroberflächen mit der Klassifikationsbaum-Methode am Beispiel von Webanwendungen

Jirka Nasarek, Peter M. Kruse

Berner & Mattner Systemtechnik GmbH
Gutenbergstr. 15
D-10587 Berlin
jirka.nasarek@berner-mattner.com
peter.kruse@berner-mattner.com

1 Einleitung

Der Test von graphischen Benutzeroberflächen (GUI) ist nach wie vor mit hohem Aufwand, sowohl für Testerstellung als auch für die Wartung von Testsuiten, verbunden. Testen von GUIs ist dennoch essenziell, da es die Arbeitsweise der Software aus Sicht des Endanwenders prüft, also so, wie sie tatsächlich in der Praxis ausgeführt wird. Eine besondere Herausforderung ist dabei oftmals die nur informelle Beschreibung der auszuführenden Aktionen für den Test.

Beim Testen von Webanwendungen besteht das Ziel darin, die Anwendung mit einer Kombination von Eingabewerten und Zuständen auszuführen, um Fehlerwirkungen aufzudecken [DLF06]. Das manuelle Überprüfen von Webanwendungen ohne Werkzeugunterstützung ist eine müßige und fehleranfällige Tätigkeit. Dies hat auch nicht zu vernachlässigende Auswirkung auf die Testbereitschaft.

Als Alternative zum manuellen Testen existieren auch verschiedene (teil-)automatisierte Ansätze. Darunter fallen Capture-Replay-Werkzeuge [OAFG98], Software zur Generierung von zufälligen Eingabedaten [BWW11], Unit-Testing und Modell-basiertes Testen.

Bei Selenium RC[1] handelt es sich um eine Software, die ein komplexes Testen von Webanwendungen ermöglicht. Dazu wird eine Programmierschnittstelle bereit gestellt, die einen Browser direkt ansteuert. Webseiten lassen sich durch ein Programm wie mittels direkter Nutzer-Interaktion ansprechen. Insbesondere können Eingaben an dafür bestimmten Elementen der Seite getätigt und die vom Webservers generierten Antworten ausgewertet werden. Praktisch ist so ein komplexer Kontrollfluss von Geschäftsprozessen durchführ- und prüfbar.

Eine Verbindung der Klassifikationsbaummethode mit den Möglichkeiten von Selenium verspricht eine Vereinfachung der systematischen Tests der Anwendung und die Sicherstellung einer höheren Qualität beim Entwicklungsprozess.

Um eine Plattform für die Forschung bezüglich des Testens von GUI-Anwendungen insgesamt zur Verfügung zu stellen und zu vereinheitlichen, wurde ein Testframework entwickelt.

[1] http://docs.seleniumhq.org/

2 Ansatz

Eine Herausforderung beim GUI Test ist die große Heterogenität der zu testenden Systeme und der dafür zur Verfügung stehenden Werkzeuge. Um eine universelle Herangehensweise des Tests von graphischen Benutzeroberflächen zu realisieren, haben wir ein Framework basierend auf dem Struktur-Entwurfsmuster Adapter entwickelt. Die Adapter erlauben es, die einzelnen Phasen des Tests von austauschbaren Komponenten bearbeiten zu lassen und damit von der konkreten Technologie zu abstrahieren. Die Adapter ermöglichen gleichfalls eine Entkopplung der einzelnen Phasen für den Testprozess.

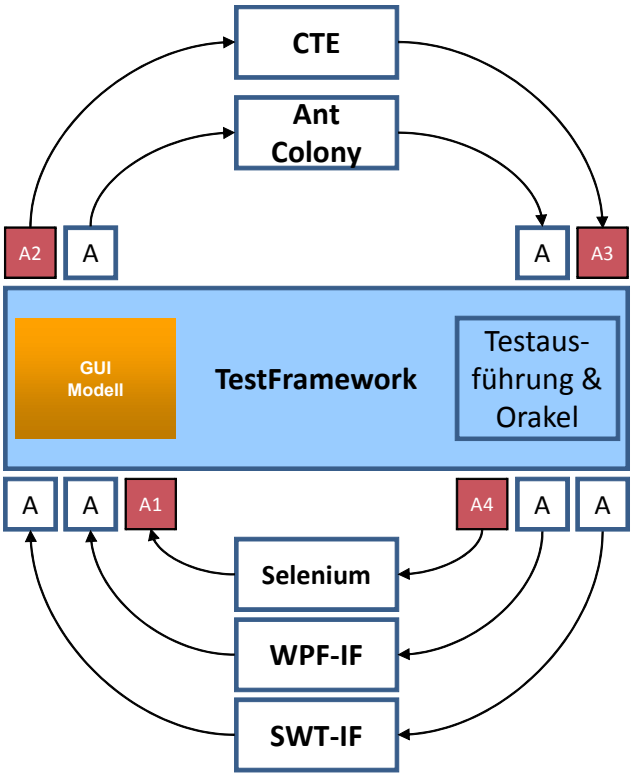

Abbildung 1: Generischs GUI Modell

Adapter **A1** liest das GUI aus und überführt es in ein Plattform-unabhängiges Format. **A2** erlaubt den Import des generischen GUI Modells in das für die Testfallerstellung verwendete Werkzeug. **A3** exportiert die spezifi-

zierten Testfälle in das generische Testfallmodell. Durch Beschreibung der Testfälle wird die Testausführung und Überwachung durch das Orakel gesteuert, welches mit dem Adapter **A4** den Prüfling instrumentiert.

Die GUI-Elemente einer Webseite lassen sich durch eine Schnittstelle weitestgehend automatisch erfassen und im Klassifikationsbaum [GG93] abbilden. Dafür wird zunächst die zu testende Seite in ein allgemeines GUI-Modell überführt und die Seitenelemente samt Interaktionsmöglichkeiten im Modell repräsentiert.

Die Klassifizierung der Elemente für die Übertragung in das GUI Modell wird durch ein heuristisches Vorgehen umgesetzt. Der Test der Seiten ist auf die Möglichkeiten von Klassifikationsbäumen (Klassifikationen, Kompositionen, Klassen) abzubilden. Der Klassifikationsbaum-Editor (CTE XL Professional[2]) bietet darüber hinaus an, die Knoten im Klassifikationsbaum mit Attributen zu annotieren, um Referenzen auf Seitenelemente zu verwalten [KL10]. Dafür wurde eine geeignete Semantik entworfen.

Der gewünschte Ablauf der einzelnen Kontrollflüsse der Webapplikation wird mit diesen Mitteln dargestellt. Von dort aus werden die Tests und das erwünschte Verhalten festgelegt und im nächsten Schritt ausgeführt.

Komponenten des generischen GUI-Modells

Das GUI Modell liefert eine hierarchische Repräsentation von Zuständen der graphischen Benutzeroberfläche. Ein GUI Modell ist ein Tupel $M = (MD, GM)$, wobei MD Metadaten und GM, die eigentliche GUI Beschreibung enthält. Die Beschreibung des GUI (GM) setzt sich aus der Menge der Standard-Aktionen (AD – beispielsweise Maus-Klick, Maus-Hover etc.), den spezifischen Aktionen (A) und den GUI Zuständen (S) zusammen: $GM = (AD, AS, S)$. Ein Zustand setzt sich aus einem Zeitstempel t und der Menge der GUI Elemente $EC = E_1, \ldots, E_n$ zusammen, wobei die einzelnen GUI Elemente andere Kinderelemente (geschachtelt) enthalten können. Jedes GUI Element ist ein Tupel $E = (isaction, VS, AS, P, EC)$ und besitzt einen eindeutigen Bezeichner (ID). Die Eigenschaft $isaction \in \{true, false\}$ kennzeichnet, ob für das Element Aktionen aktiv sind, VS beschreibt die durch das Element annehmbaren Wertemengen und ob es sich dabei um Eingabe- oder Ausgabewerte handelt, auch jede Wertemenge hat einen eindeutigen Bezeichner. AS enthält die entsprechenden Referenzen auf die Aktionen, P ist eine Sammlung frei definierbarer Attribut-Werte-Paare (*Properties*) und EC die oben erwähnten Kindelemente.

Komponenten des Testfall-Modells

Die Testdaten des Testmodells werden als Tupel $TD = (guiRef, TS)$ dargestellt. $guiRef$ stellt einen Verweis auf das verwendete GUI-Modell dar, TS enthält beliebig viele Test-Suiten, die wiederum beliebig viele Testfälle enthalten können. Ein Testfall enthält eine Anweisungsmenge, die sich aus Eingabe-, Ausführungs- und Orakel-Anweisungen zusammensetzt und besitzt einen eindeutigen Bezeichner (ID).

In der *Eingabeanweisung* wird auf das Element-, sowie auf dessen Ziel-Wertetyp mittels deren IDs im GUI-Modell referenziert. Der Wert, den das Element annehmen soll, wird in einem Feld *inputValue* direkt angegeben. Hierfür lassen sich beliebige XML-Schema-Typen verwenden.

Die *Ausführungsanweisung* setzt sich aus einem Verweis auf das zu aktivierende Element, einem Verweis auf eine Aktion (sei sie eine Standard-Aktion, wie einem Maus-Klick oder eine spezifische Aktion bei detektiertem Event-Handler).

Eine *Orakel-Anweisungen* setzt sich zusammen aus: Der ID des Ausgabeelements, einem Vergleichs-Operator $op \in \{=, \neq, <, \leq, >, \geq\}$ einer Referenz auf einen Wertetyp des Elements, sowie einer optionalen Menge von zu überprüfenden Eigenschaften.

Alle drei Anweisungsarten haben noch eine Kommando-Nummer, welche die Reihenfolge während der Testausführung bestimmt.

3 Implementierung

A1: Auslesen der GUI

Der A1 Adapter erzeugt aus einer Webseite mit Hilfe von Selenium das GUI Modell. Dazu wird das HTML Dokument als Baum per Tiefensuche durchwandert. Zu jedem Knoten findet eine Prüfung statt, ob es sich dabei um ein Tag mit bekannten Eigenschaften, zum Beispiel ein Form-Element, handelt, um festzustellen, ob sie für die Ein- oder Ausgabe verwendet werden können und ob Aktionen (bspw. Javascript-Events) für sie registriert wurden. Eine besondere Rolle spielen HTML-Form-Elemente. Deren Vorteil liegt darin, dass der HTML-Standard[3] die Bestandteile für konkrete Einsatzzwecke spezifiziert. So lassen sich aus Form-Elementen auch mögliche Wertebereiche herleiten.

Im HTML 5 Standard wurden eine Reihe von neuen Typen von Eingabeelementen[4] eingeführt. Diese erleichtern die Erzeugung passender Testdaten in späteren Schritten des Testprozesses. Die Ausgabe des A1 ist eine generische Beschreibung des GUI.

A2: Import in CTE

Der Adapter A2 überführt das GUI-Modell der Seite in ein für den Klassifikationsbaum-Editor CTE XL Professional [KL10] lesbares Format. Im CTE wird das GUI Modell graphisch aufbereitet und der Benutzer kann (je nach Testziel) relevante Teile auswählen. Zu gewählten Teilen werden dann automatisch Elemente des Klassifikationsbaums erstellt und Aktionselemente farblich gekennzeichnet. Ebenso werden zusätzliche Informationen aus dem GUI Modell als Attribute angefügt. Die Klassen im Klassifikationsbaum repräsentieren die Werte – als Test-

[2] http://www.cte-xl-professional.com/

[3] http://www.w3.org/TR/html401/

[4] http://www.w3.org/TR/html-markup/input.html

daten (Eingabe) oder Soll-Werte zum Vergleich durch das Orakel (Ausgabe). Sind annehmbare Werte automatisch auslesbar (wie bei Markierungsfeldern und Auswahllisten) übernimmt der Adapter das Einfügen der Klassen automatisch. Klassen für die übrigen Werte müssen vom Tester selbst eingegeben werden.

Danach lassen sich Testfälle im CTE entweder manuell spezifizieren [WG93] oder automatisch generieren [KL10].

A3: Testfallgenerierung

Der Adapter A3 überführt die in CTE spezifizierten Testfälle in das allgemeine Testfall-Modell. Jede im Testfall gesetzte Markierung entspricht einer Klasse des Klassifikationsbaums und einem zu setzenden Wert (im Fall eines Eingabeelements) bzw. einer zu prüfenden Bedingung (im Fall eines Ausgabeelements) oder einer auszuführenden Aktion (im Fall eines Aktionselements).

A4: Testfallausführung

Die Abarbeitung der Testfälle erfolgt durch den Adapter A4 in Zusammenspiel mit Selenium. Dabei werden die Testfälle aus der allgemeinen Beschreibung als konkrete Befehlssequenzen interpretiert.

Funktion von A4 ist Starten des Prüflings, Herstellen des Testkontexts und Ausführung der Testfälle. Das Ausführen besteht aus Bestimmen der Ausführungs-Reihenfolge, Auffinden der Elemente, Durchführen der Operationen, Initialisieren des Test-Orakels sowie Auswertung des Tests.

4 Verwandte Arbeiten

Klassisches Capture Replay gibt es bereits eine ganze Weile als Bestandteil des Softwaretests. So diskutieren Ostrand et al. Wartung von Testsuiten [OAFG98]. Memon und Xie evaluieren Fehlererkennungsraten von Capture Replay [MX05]. Gerade bei der Wartbarkeit mangelt es den bestehenden Ansätze. Änderungen an der GUI führen so leicht zu aufwändigen Anpassungsarbeiten. Deshalb gibt es Versuche, die Testsuite-Wartung zu automatisieren [GXF09]. Auch die Arbeit von Leotta et al. zielt darauf ab, konventionelle, auf Selenium basierende Ansätze robuster gegen Änderungen an der Anwendung zu machen [LCRS13].

Ein anderer Ansatz besteht darin, die Anfälligkeit für GUI-Änderungen ganz zu vermeiden und zur Laufzeit ein jeweils aktuelles Modell zu erzeugen [BWW11]. Dies führt allerdings auch dazu, dass funktionale Fehler (Falsche Ausgaben, Layouts, ...) nicht gefunden werden können.

Mao umgeht das Problem der Oberflächenänderungen, indem er sich auf zu Grunde liegende Web Services beim kombinatorischen Test konzentriert [Mao08]. Er extrahiert dazu relevante Testaspekte aus den formalen Schnittstellenbeschreibungen.

Sun und Jones analysieren ebenfalls das Application Programmable Interface (API) um GUI Tests zu generieren [SJ04].

Ricca et al. gehen noch einen Schritt weiter. Sie versuchen durch Beobachtung ein internes Laufzeitmodel der Webanwendung zu erzeugen [RT01]. Basierend auf einem solchen Modell haben Nguyen et al. bereits kombinatorischen Testentwurf in Zusammenspiel mit Modellbasiertem Test angewendet [NMT12].

Klassifikationsbäumen lassen sich auch aus Z-Spezifikationen generieren [HHS03].

5 Zusammenfassung

Ziel der Arbeit war die prototypische Umsetzung eines Frameworks für den Test graphischer Benutzeroberflächen von Webanwendungen. Dazu wurde ein generisches Framework entwickelt und geeigenete Sprachmittel geschaffen. Das gewählte Adapter-Konzept gewährleistet eine klare Aufgabentrennung und ermöglicht so die Austauschbarkeit von Komponenten.

Die Untersuchung des Erstellungs- und Wartungs-Aufwands, auch im Vergleich mit bestehenden Techniken wie Capture Replay, steht noch aus.

Für neuere Webanwendungen, die mittels dynamischer Ausdrucksmittel den Aufbau der Webseite zur Laufzeit ändern, ergeben sich darüber hinaus ganz neue Anforderungen, die auch mit unserem Ansatz noch nicht adressiert werden.

Danksagung

Diese Arbeit wurde durch das FITTEST Projekt (EU ICT-257574) unterstützt.

Literatur

[BWW11] Sebastian Bauersfeld, Stefan Wappler und Joachim Wegener. An approach to automatic input sequence generation for GUI testing using ant colony optimization. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, GECCO '11, Seiten 251–252, New York, NY, USA, 2011. ACM.

[DLF06] Giuseppe A. Di Lucca und Anna Rita Fasolino. Testing Web-based applications: The state of the art and future trends. *Inf. Softw. Technol.*, 48:1172–1186, December 2006.

[GG93] Matthias Grochtmann und Klaus Grimm. Classification trees for partition testing. *Software Testing, Verification and Reliability*, 3(2):63–82, 1993.

[GXF09] Mark Grechanik, Qing Xie und Chen Fu. Maintaining and evolving GUI-directed test scripts. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, Seiten 408–418. IEEE, 2009.

[HHS03] Robert M. Hierons, Mark Harman und Harbhajan Singh. Automatically generating information from a Z specification to support the Classification Tree Method. In *3rd International Conference of B and Z Users, LNCS volume 2651*, Seiten 388–407, June 2003.

[KL10] Peter M. Kruse und Magdalena Luniak. Automated Test Case Generation Using Classification Trees. *Software Quality Professional*, 13(1):4–12, 2010.

[LCRS13] Maurizio Leotta, Diego Clerissi, Filippo Ricca und Cristiano Spadaro. Comparing the maintainability of selenium WebDriver test suites employing different locators: a case study. In *Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation*, JAMAICA 2013, Seiten 53–58, New York, NY, USA, 2013. ACM.

[Mao08] Chengying Mao. Performing Combinatorial Testing on Web Service-Based Software. In *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 02*, CSSE '08, Seiten 755–758, Washington, DC, USA, 2008. IEEE Computer Society.

[MX05] Atif M. Memon und Qing Xie. Studying the Fault-Detection Effectiveness of GUI Test Cases for Rapidly Evolving Software. *IEEE Trans. Softw. Eng.*, 31(10):884–896, 2005.

[NMT12] Cu D. Nguyen, Alessandro Marchetto und Paolo Tonella. Combining model-based and combinatorial testing for effective test case generation. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ISSTA 2012, Seiten 100–110, New York, NY, USA, 2012. ACM.

[OAFG98] Thomas Ostrand, Aaron Anodide, Herbert Foster und Tarak Goradia. A visual test development environment for GUI systems. *ACM SIGSOFT Software Engineering Notes*, 23(2):82–92, 1998.

[RT01] Filippo Ricca und Paolo Tonella. Analysis and testing of Web applications. In *Proceedings of the 23rd International Conference on Software Engineering*, ICSE '01, Seiten 25–34, Washington, DC, USA, 2001. IEEE Computer Society.

[SJ04] Yanhong Sun und Edward L Jones. Specification-driven automated testing of GUI-based Java programs. In *Proceedings of the 42nd annual Southeast regional conference*, Seiten 140–145. ACM, 2004.

[WG93] Joachim Wegener und Matthias Grochtmann. Werkzeugunterstützte Testfallermittlung für den funktionalen Test mit dem Klassifikationsbaum-Editor CTE. In *Proceedings of the GI-Symposium Softwaretechnik '93*, Seiten 95 – 102, Dortmund, November 1993. Daimler-Benz AG, Forschung Systemtechnik, Berlin.