Automatische Generierung von Testsequenzen für GUI-basierte Anwendungen mit Ant Colony Optimization

Joachim Wegener, Sebastian Bauersfeld und Stefan Wappler

Berner & Mattner Systemtechnik GmbH
Gutenbergstr. 15
10587 Berlin

Joachim.Wegener/Sebastian.Bauersfeld/Stefan.Wappler@berner-mattner.com

1 Einleitung

Die Automatisierung des Tests von Anwendungen mit grafischer Benutzerschnittstelle (graphical user interface; GUI) ist trotz einer großen Anzahl von verfügbaren Testwerkzeugen in der Praxis immer noch unbefriedigend gelöst. GUI-Tests werden meist mit Capture-Replay-Werkzeugen aufgezeichnet und wieder abgespielt oder mit Hilfe von Testskripten implementiert. Bei Änderungen am grafischen Design der Benutzerschnittstelle lassen sich die aufgezeichneten oder implementierten Testskripte nicht direkt weiterverwenden, sondern müssen je nach Umfang der Änderungen am GUI mehr oder weniger stark angepasst werden. Häufig leidet darunter die Akzeptanz der Werkzeuge, so dass auch manuelle Tests weit verbreitet sind [1]. Ziel unserer Arbeiten ist es, den Test von GUI-basierten Anwendungen von der Testfallermittlung bis zur Testausführung zu automatisieren. Dabei sollen Tests generiert werden, welche die Funktionen des Prüflings möglichst umfassend überdecken. Die automatische Generierung von Interaktionsfolgen soll einerseits eine deutliche Erhöhung der Testtiefe und damit eine Steigerung der Testqualität ermöglichen und andererseits für eine Verbesserung der Testeffizienz und damit für eine Kostenreduzierung der Tests sorgen.

2 Ansatz auf Basis Ant Colony Optimization

Unser Konzept für den vollautomatischen Test von GUI-basierten Anwendungen beruht auf der Verwendung eines Suchverfahrens. Als Suchverfahren kommt Ant Colony Optimization [2] zum Einsatz, da dieses Verfahren für die Sequenzgenerierung besser geeignet ist als andere Metaheuristiken [3]. Ziel ist es, möglichst interessante Sequenzen von Benutzerinteraktionen für die Prüfung des Testgegenstands zu generieren, auszuführen und zu bewerten. Als interessante Testsequenzen gelten Interaktionsfolgen, die einen möglichst umfassenden Teil des Testobjekts ausführen. Denkbar sind aber auch andere Testziele, beispielsweise die Aufdeckung von Laufzeitfehlern, die Verletzung von Assertions oder eine möglichst hohe Code-Überdeckung.

In der Analogie von Ant Colony Optimization stellt ein Ameisenpfad eine Interaktionsfolge dar. Der Pfad wird während der Suche auf Basis sogenannter Pheromone und anderer Parameter konstruiert. Die Pheromone akkumulieren die Häufigkeit, mit der ein bestimmtes Pfadelement – in unserem Kontext eine ganz bestimmte Interaktion – in einer bisherigen Sequenz verwendet wurde. Die Idee dahinter ist, dass ein Pfadelement, das von vielen Ameisen benutzt wurde, besonders erfolgversprechend ist. Beim Passieren legen die Ameisen zur Markierung ihres Weges die Pheromone ab.

Die Anwendung eines Suchverfahrens macht es erforderlich, die Güte einer generierten Sequenz durch eine sogenannte Fitnessfunktion zu beurteilen. Die Fitnessbewertung für die generierten Testsequenzen wird auf Basis einer Bytecode-Instrumentierung des zu prüfenden Systems vorgenommen. Die Instrumentierung dient dazu, während der Testdurchführung den Call-Tree [4] für die generierten Interaktionsfolgen zu bestimmen. Der Call-Tree ist ein Modell der Aufrufpfade, die bei der Ausführung der Interaktionsfolge durchlaufen werden. Anhand des Call-Trees wird der Fitnesswert für die jeweils erzeugte Interaktionsfolge berechnet.

Die Generierung der Testsequenzen erfolgt dynamisch-inkrementell, d.h. nach jeder einzelnen, ausgeführten Interaktion wird das GUI neu nach den sichtbaren und anwählbaren GUI-Elementen durchsucht, beispielsweise nach Schaltflächen, Eingabefeldern und Auswahlboxen. Für jedes anwählbare GUI-Element werden die möglichen Benutzeraktionen wie Tastatureingabe, Mausklick oder Drag & Drop bestimmt. Aus den möglichen Benutzeraktionen wird dann diejenige mit einer bestimmten Wahrscheinlichkeit ausgewählt, die am erfolgversprechendsten ist. Durch diese dynamisch-inkrementelle Vorgehensweise ist sichergestellt, dass nur gültige Testsequenzen von Benutzerinteraktionen generiert werden. Das für andere Verfahren der Testsequenzgenerierung typische Problem der Generierung ungültiger bzw. nicht ausführbarer Testsequenzen kann mit unserem Verfahren nicht auftreten.

Die Auswahl der vielversprechendsten Benutzeraktion richtet sich nach dem gelernten Wissen

über das Verhalten des Testobjekts. Während der Suche nach interessanten Interaktionsfolgen erwirbt das Suchverfahren Wissen über das Testobjekt in Form von internen Parametern, die kontinuierlich angepasst werden. Zu jeder ausgeführten Testsequenz wird der Call-Tree bestimmt. Es sollen Testsequenzen erzeugt werden, die zu einem möglichst großen Call-Tree führen, da dessen Größe einen Anhaltspunkt für die Komplexität und den Umfang des durch die Testsequenz ausgeführten Programmanteils liefert: Sequenzen von Benutzerinteraktionen, die zu einem großen Call-Tree führen, durchlaufen mehr Systemfunktionalität als Benutzerinteraktionen mit einem kleinen Call-Tree und werden in unserem Verfahren folglich positiver bewertet. Dabei stellt die Anzahl der Blattknoten des Call-Trees einer Sequenz die Fitness dieser Sequenz dar. Über die Menge der ausgeführten Testsequenzen wird gelernt, welche Benutzerinteraktionen besonders häufig in Sequenzen mit umfangreichem Call-Tree enthalten sind. Dieses Wissen wird über die Speicherung von Pheromonwerten zu den einzelnen Benutzeraktionen kumuliert. Während der Generierung einer neuen Testsequenz erhält eine Benutzeraktion mit hohem Pheromonwert eine höhere Wahrscheinlichkeit, in die neue Testsequenz mit aufgenommen zu werden, als eine Benutzeraktion mit niedrigem Pheromonwert.

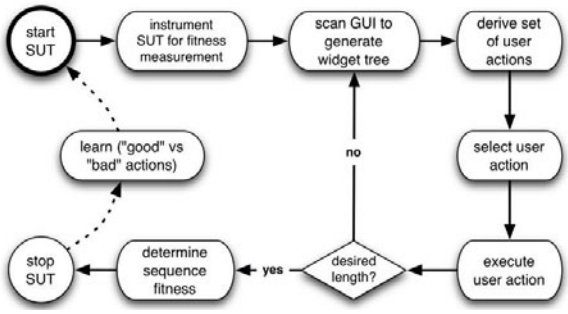

Abbildung 1: Ablauf des Verfahrens

Abbildung 1 zeigt zusammenfassend den Ablauf des Verfahrens. Zuerst wird das System under Test (SUT) gestartet und anschließend instrumentiert. Die Instrumentierung dient dazu, die Call-Trees zu ermitteln. Dabei wird das SUT nach GUI-Elementen durchsucht; diese werden in einem sogenannten Widget-Tree modelliert. Aus diesem Widget-Tree wird anschließend die Menge der möglichen Benutzerinteraktionen extrahiert. Auf Basis der gelernten Wahrscheinlichkeitswerte wird daraus eine durchzuführende Aktion ausgewählt und ausgeführt. Ist eine bestimmte Sequenzlänge durch Wiederholung des Schritts der Widget-Tree-Generierung (bzw. –Erweiterung), Interaktionsmengenextraktion, Interaktionsauswahl und –ausführung erreicht, wird die Fitness der so erzeugten Interaktionsfolge auf Basis des Call-Trees berechnet, den die Interaktionsfolge erzeugt hat. Schließlich wird das SUT gestoppt. Nachdem die internen Parameter (Pheromone) angepasst wurden, beginnt der nächste Durchlauf des Suchverfahrens, bis ein bestimmtes Ende-Kriterium erreicht ist.

3 Prototyp und Fallstudie

Das entwickelte Konzept wurde in einer prototypischen Implementierung umgesetzt, welche die suchbasierte Generierung von Testsequenzen für Java-Anwendungen mit SWT-Benutzungsoberflächen erlaubt. Zur Prüfung des Ansatzes wurde der CTE XL Professional mit unserem Verfahren getestet. Die Ergebnisse wurden mit einem Zufallstest verglichen. In umfangreichen Testreihen mit Untersuchungen zur statistischen Signifikanz der Ergebnisse konnte nachgewiesen werden, dass der suchbasierte Test der GUI-basierten Anwendung auf Basis unseres Ansatzes einem entsprechenden Zufallstest deutlich überlegen war. Es ist davon auszugehen, dass der suchbasierte Test auch beim Test anderer GUI-basierter Anwendungen ähnlich effektiv ist.

4 Zusammenfassung und Ausblick

Es lassen sich reproduzierbar Testsequenzen mit wesentlich umfangreicheren Call-Trees erzeugen als dies bei der zufälligen Generierung von Testsequenzen der Fall ist. Gegenüber anderen Lösungsansätzen, wie dem Einsatz von Capture-Replay-Werkzeugen oder Testskripten, besitzt unser Verfahren zudem den Vorteil, dass es sich stets neu an die gegebene Benutzungsoberfläche adaptiert. Die aufwändige Wartung vorhandener Testsuiten im Rahmen von Regressionstests kann entfallen.

In der nächsten Ausbaustufe werden wir die entwickelte Testumgebung so erweitern, dass ganze Testsuiten erzeugt werden. Dabei wird sichergestellt, dass die einzelnen Testsequenzen möglichst wenige Überlappungen zueinander aufweisen und zusammen eine möglichst vollständige Ausführung des Testobjekts sicherstellen. Hierzu ist es vorstellbar, die Fitnessfunktion so zu erweitern, dass sie die bereits durch vorhandene Sequenzen erzeugten Call-Trees berücksichtigt und besonders Sequenzen belohnt, die zu neuen, bislang unerreichten Programmteilen führen.

Referenzen

[1] Memon, A. M.: A comprehensive framework for testing graphical user interfaces. Dissertation, Universität Pittsburgh, USA (2001)

[2] Dorigo, M., Blum, C.: Ant colony optimization theory: a survey. Theoretical Computer Science 344, S. 243-278 (2005)

[3] Bauersfeld, S.: A metaheuristic approach to automatic test case generation for GUI-based applications. Diplomarbeit, Humboldt-Universität zu Berlin (2010)

[4] McMaster, S., Memon, A.: Call-stack coverage for GUI test suite reduction. IEEE Transactions on Software Engineering 34, S. 99-115 (2008)