

Modelgetriebenes Testen einer Webanwendung mit Hilfe von Zustandsautomaten und Testframeworks

Nils Petersohn
Fachhochschule Brandenburg
Brandenburg a. d. Havel, Deutschland
Email: post@berlin.de

Robert Stümer
Fachhochschule Brandenburg
Brandenburg a. d. Havel, Deutschland
Email: robert.stuemer@googlemail.com

Abstract—In dieser Arbeit wird das modellbasierte Testen einer Webseite durch das Zusammenspiel von einem aus dieser Applikation per Reverse Engineering entwickelten gerichteten Graphen und Testframeworks beschrieben. Um die Auswahl der gewünschten Testframeworks so variabel wie möglich zu gestalten, wurde ein plattformunabhängiges Testframework-Modell erstellt, welches eine Schnittmenge der zu implementierenden Testframeworks bildet. Durch eine “Model zu Text” (M2T) Transformation wird das Testframework-Modell mit Hilfe einer Templatesprache in spezifischen Testcode gewandelt.

Beim Durchlaufen des mit Hilfe von Reverse Engineering erstellten Graphen werden bei jedem Zustandsübergang Befehle für das Testframework erstellt, um Pfade im Graphen automatisch durch M2T in Testcode umzuwandeln.

I. EINLEITUNG

In der Vergangenheit hat das Testen von Webanwendungen immer mehr Bedeutung erlangt. Da die Anwendungen immer größer und unüberschaubarer werden, ist der Einsatz von Test- und testunterstützenden Werkzeugen eine große Hilfe.

In einer agilen Entwicklungsumgebung spielen Tests eine primäre Rolle. Test-Driven-Development (TTD) ist eine Methode, um bei agiler Entwicklung iterativ den Code abzusichern. Dieses Paper grenzt sich von diesem Testbegriff ab und handelt von Test-Vorgängen für eine bereits entwickelte Applikation. Die zu testende Applikation wird in dieser Arbeit als gerichteter Graph bzw. Zustandsautomat betrachtet. Durch diese Betrachtungsweise entstehen verschiedene Möglichkeiten zum Testen der Applikation. Zum Beispiel kann damit die Benutzbarkeit einer Webseite getestet werden, indem der A* Algorithmus dazu verwendet wird den Einkaufsprozess einer E-Commerce Seite zu untersuchen. Abstrakt existiert genau dann schon ein Graph, wenn ein Zustandsübergang jeglicher Form stattfinden kann. Zum Beispiel ruft der Klick auf einen Hyperlink einen Zustandsübergang hervor von Webseite A hin zu Webseite B. Im weitesten Sinne ist schon die “kleinste” Veränderung auf einer Webseite ein Zustandsübergang, z.B. wenn ein Pixel auf einer Webseite auch nur einen Bit in der Farbe wechselt.

Dieses Modell der Zustandsübergänge also des Graphen an sich, ist auf alle Testframeworks übertragbar. Durch das Ausführen eines Befehls z.B. das Klicken auf einen Button mit einer “id=btnX” (Pseudobefehl: Testframework.btn(‘btnX’).click()) führt zu einem Zustandsübergang,



Fig. 1.

wenn z.B. eine Javascript Funktion für den abgefeuerten Klick-event existiert der eine Veränderung der Webseite hervorruft.

Ziel des zu entwickelnden Systems soll es sein, eine Webseite in einen konfigurierbar feingranularen Graphen zu wandeln. An die Kanten des Graphen sollen der für den Zustandsübergang benötigte Befehl eines beliebigen Testframeworks gekoppelt werden, um somit beliebige Pfadschritte durch den Graphen in Testframeworkbefehle zu übersetzen. Die Entwicklung beschränkt sich weitestgehend auf den Entwurf der Umgebung, um Programme, die auf Graphwanderung spezialisiert sind eine Möglichkeit zu bieten Testcode parallel auszuführen.

mds
15. Juli 2011

A. Grundlagen / verwandte Arbeiten

Wie in [1] und [2] beschrieben, wurde ebenfalls ein endlicher Zustandsautomat verwendet, um mögliche Zustände und Zustandsübergänge der Webanwendung in einem Modell darzustellen. Dabei bilden Webseiten der Anwendung die Zustände (wobei eine Webseite auch mehrere Zustände einnehmen kann) und Eingaben der Nutzer führen zu einem Zustandsübergang. Der Startzustand entspricht der Startseite der Anwendung. Für einen ersten Durchstich wurde ein simpler Parser entwickelt der aus Java Enterprise Edition-Anwendungen per Reverse Engineering einen Zustandsautomat erstellt. So ein Zustandsautomat besteht aus Knoten und Kanten. Die Knoten repräsentieren die Objekte und die Kanten die Aktionen zwischen ihnen.

II. HAUPTTEIL

A. Vorgehen

Eine Webanwendung wird geparkt und daraus ein Graph erstellt. Die Eingaben für den Automaten werden durch das

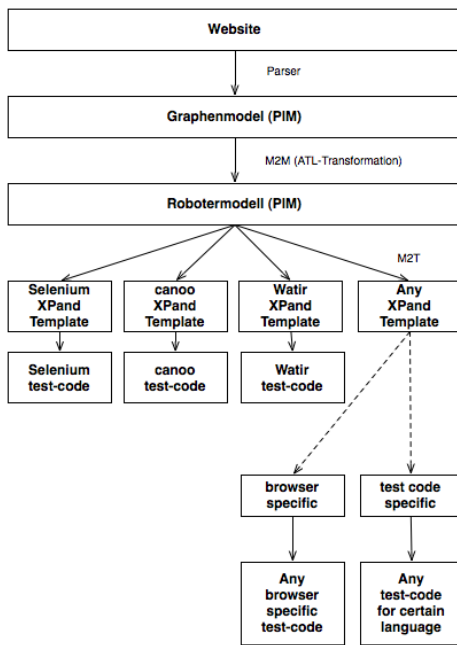


Fig. 2. Globaler Aufbau (Top-Down)

Programm Graphwalker¹ erstellt. Diese liest das Modell des Graphen ein und generiert Eingaben für selbigen. Typische Eingaben sind Wege durch den Graphen, die ein Maximum der vorhandenen Zustände abdecken, um den Testvorgang zu optimieren.

Zur Erstellung dieser Wege können verschiedene Algorithmen verwendet werden. Der bekannteste ist der A-Stern Algorithmus. Die Komplexität der Vorgehensweise wird dabei der Größe des generierten Automaten angepasst. Die aufwendigste wäre, jede mögliche Folge von Ereignissen zu durchlaufen. Da dies bei großen Automaten kaum zu gewährleisten ist, gibt es verringerte Varianten, die nur Eingaben erzeugen, die einen Zustandswechsel zur Folge haben, da hier die meisten Fehler passieren. Eine andere Variante wäre, jede eingehende Kante eines Zustandes mit jeder ausgehenden Kante des Zustandes auszuführen.

Für die Umsetzung wurden die Eclipse Frameworks XPand², EMF³ und ATL⁴ genutzt.

B. Reverse Engineering und Automatenmodell

Zuerst wurde ein Programm entwickelt, das die Struktur eines Webprojektes in einem Graph abbildet. Dazu wurde ein einfacher Parser in Groovy entwickelt, um mit Hilfe von Rekursion und Tiefensuche die gerichtete Graph-Datenstruktur von Scott Stark⁵ zu instanzieren. Parallel dazu wird eine

¹<http://graphwalker.org>

²XPand ist darauf ausgelegt aus EMF Modellen Quellcode zu generieren

³Das Eclipse Modeling Framework ist ein Sammlung von Werkzeugen zur Modellgetriebenen Entwicklung in Eclipse

⁴ATLAS Transformation Language ist eine Programmiersprache zum Transformieren von Modellen

⁵Scott.Stark@jboss.org

```
<graphml>
<graph edgedefault="undirected">
<node id="n1"/>
<node id="n2"/>
<node id="n3"/>
<edge id="e1" source="n1" target="n2" directed="true"/>
<edge id="e2" source="n2" target="n3" directed="false"/>
<edge id="e3" source="n3" target="n1"/>
</graph>
</graphml>
```

Fig. 3. Graphml Beispiel

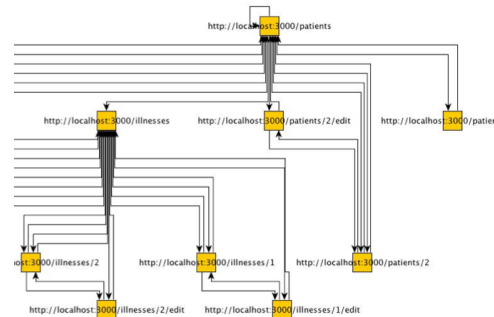


Fig. 4. yEd Graph

Datei im *.graphml (Fig. 3) Format erstellt, um die Webseite bzw. deren Graphabbildung mit yEd zu visualisieren (Fig. 4). Hauptsächlich aber deswegen, weil das Graphwalker Programm dieses Format importieren kann. Dieses Format ist die Schnittstelle zwischen Graph und vorhandener "Pfadwanderungssoftware".

Genauer wurden hier die einzelnen Webseiten in Zustände und die Navigation über Links, Buttons und Actions als Zustandsübergänge interpretiert. Der Graph wird zusätzlich im XMI⁶ Format gespeichert. Dieses Format beschreibt eine Instanz eines Metamodells auf Basis von Eclipse Tools für Modellgetriebene-Softwareentwicklung (MDSO). Das Metamodel, welches den Graphen beschreibt, wurde als Ecoremodelliert und umfasst den Graphen, welcher aus Knoten und Kanten besteht. Eine Kante (engl. Edge) hat einen Ziel und einen Quell-Knoten (engl. Vertex) und repräsentiert ein Webseitenelement. Um die Komplexität zu verringern, repräsentiert ein Pfeil zwischen den Knoten (die Kante) einen Webseite-Hyperlink, welcher einen Text umschließt, den LinkText. Durch Klicken des Links wird eine neue Seite aufgerufen und somit auch ein Zustandsübergang ausgelöst (Fig. 5).

C. Transformation vom Graphen zu Abstrakten Testschritten

Das Graphen-Metamodell wird anschließend mit Hilfe von ATL in ein Testframework-Metamodell transformiert, um den Ablauf von Testframeworkbefehlen abzubilden. Dieses zweite Metamodel (Fig. 6) beschreibt einen virtuellen Testroboter (AnyBot) welcher sich leicht mit einem Roboter, der Mausklicks und Tastatureingaben auf Webseiten tätigt, gedanklich vorstellen lässt.

Der Anybot führt dementsprechend verschiedene Schritte (steps) nacheinander durch welche sich in einer bestimmten Ordnung (order) befinden und actions ausführen. Genauer

⁶XMI ist eine XML Beschreibungssprache die für Eclipse MDA Tools als Eingabe verwendet werden kann

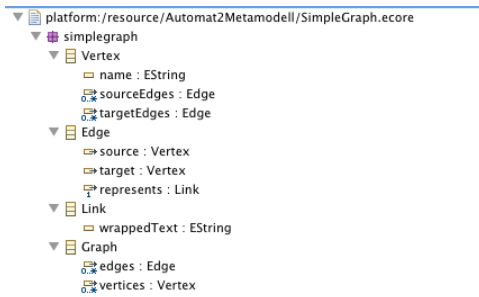


Fig. 5. Simple Graph Ecore

```

«IMPORT SimpleBot»

«DEFINE main FOR Step»
«FILE "x.java"»
«EXPAND clickAction FOREACH clickActions»
«ENDFILE»
«ENDEFFINE»

«DEFINE clickAction FOR ClickAction»
  bot.linkWithText("«this.linkText»").click();
«ENDEFFINE»

```

Fig. 8. Beispiel Xpand

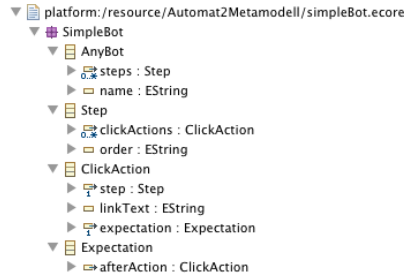


Fig. 6. AnyBot Ecore

```

-- @path SimpleGraph=/Automat2Metamodell/SimpleGraph.ecore
-- @path SimpleBot=/Automat2Metamodell/simpleBot.ecore

module Graph2AbstractModel;
create OUT : SimpleBot from IN : SimpleGraph;

rule Edge2Step {
  from
    s: SimpleGraph!Edge
  to
    t: SimpleBot!Step (
      clickActions <- clickOnWhat
    ),
    clickOnWhat: SimpleBot!ClickAction (
      linkText <- s.represents.wrappedText
    )
}

```

Fig. 7. ATL Regeln für M2M Transformation

wurde in diesem Metamodel nur die "Klick-Action" spezifiziert und dazugehörig der Linktext, der geklickt werden soll bei einer klick Action. Da Link Texte mehrdeutig sein können, empfiehlt sich hier eine eindeutige Repräsentation durch css selektor-syntax oder xPath ausdrücken, da so keine Erweiterung des Websitequelltextes nötig ist. Weiterhin beschreibt eine Action auch eine Erwartung, nämlich dass, was erwartet wird, wenn die Action ausgeführt ist. Die Erwartung soll ein Zustand im Graphen sein, welcher von der Kante aus erreichbar ist. Die Transformationsbeschreibung (Fig. 7) konvertiert das Graphenmodell in das Robotermodell, welches Kanten in Schritte bzw. Actions umwandelt, spezifisch in Klickactions für Hyperlinks. Die zwischenregel "clickOn-What" nimmt den vom Link umschlossenen Text und reicht ihn an den auszuführenden AnyBot-Step weiter damit dieser als Identifikation für den Link funktionieren kann.

D. Codegenerierung

Um die Tests ausführen zu können, muss per ModelTo-Text Transformation Quellcode erstellt werden. Hierzu wurde Xpand verwendet, da es aus Modellen textuelle Artefakte erzeugen kann und in die verwendete Eclipse-Umgebung sehr gut integriert ist. Ausgehend von dem Metamodell für die Testschritte wird mit der Templatesprache Xpand der Testcode generiert. Hier findet die Umwandlung vom Plattformunabhängigen (PIM) in das Plattformspezifische Model (PSM) statt. Das jeweilige Template für ein bestimmtes Testframework kann nun verwendet und ausgetauscht werden, wenn ein anderes Testframework zum einsatz kommen soll. Ein Beispiel für so ein Template ist in Fig. 8 dargestellt. Darin iteriert Xpand über alle click Actions der Modelinstanz und erzeugt einen String, welcher direkt die Ausführungsbefehle des Testframeworks ausdrückt.

E. Durchführung von Tests

Bei der Durchführung des Tests kann zuerst der generierte Graph analysiert werden. Auf diesem Graphen lassen sich schon Tests durchführen. Es kann geprüft werden ob es Zustände (Seiten der Webseite) gibt, die nicht erreichbar sind, oder Verlinkungen, die ins Leere führen. Es wird ebenfalls getestet, ob Navigationen vereinfacht oder weitere Verlinkungen hinzugefügt werden können, um eine schnellere Navigation auf der Seite zu ermöglichen.

Das Java Programm "Graphwalker" ist open source und kann somit verwendet werden um dessen abgelaufenen Pfade dazu zu verwenden um Plattform spezifischen Testcode zu erzeugen. Die Model basierte Testimplementierung⁷ von Tigris zeigt die Verwendung des Graphwalker, weil dieser aus diesem MBT-Projekt entstanden ist. Bei der Initialisierung von Graphwalker wird die graphml Datei angegeben, um Pfade dieser zu durchlaufen. Es kann z.B. der Zielknoten im Graphen angegeben werden und nach durchlauf wird z.B. die Zeit gemessen, bis der ausgewählte A* Algorithmus es geschafft hat den Zielknoten zu erreichen. Daraus ergeben sich Performancetests jeglicher Pfade. Die Standard-Ausgabe von dem Programm listet die zurueckgelegten Wege auf. Genau dieser Output befindet sich in der Methode org.graphwalker.ModelBasedTesting.writePath(PrintStream out) und kann dazu verwendet werden Vertex

⁷<http://mbt.tigris.org/servlets/ProjectHome>

und Edge Klassen (org.graphwalker.graph.Vertex, org.graphwalker.graph.Edge) zu überschreiben um eigene, aus dem Graphen-Metamodell zu verwenden die dennoch hinreichend für die Funktionalität von Graphwalker bleiben. Die writePath Methode kann dann überschrieben werden um separat eine XMI Datei zu erzeugen um diese dann später in das Testframework-Metamodell zu wandeln und dann zum spezifischen Framework-Code als Ausführungssequenz. Da Xpand mehrschichtige Textgenerierung unterstützt sind noch speziellere Generierungsverfahren möglich, die z.B. Browserspezifische Merkmale unterstützen. Weiterhin kann dies noch auf eine bestimmte Programmiersprachen spezialisiert sein, wenn die geeigneten Templateteile überschrieben werden.

III. FAZIT UND AUSBLICK

Der erstellte Graph liefert einen besseren Einstieg, da so der Umfang und die Struktur des Projektes auch ohne Programmierkenntnis leicht zu überschauen ist.

Bei der Analyse des Automaten können Fehler die mit herkömmlichen Methoden nur schwer zu finden sind mit wenig Aufwand entdeckt werden. Nicht erreichbare Seiten der Anwendung genauso wie fehlerhafte Verlinkungen zwischen Webseiten können erkannt werden.

Auch mögliche Fehler in der Logik der Anwendung können erkannt werden z.B. das der Nutzer eine Bestellung abschicken kann, ohne sich vorher registriert zu haben.

Außerdem können fehlende Verlinkungen aus Sicht der Benutzbarkeit erkannt werden, um so dem Nutzer eine bessere Navigation durch die Seite zu bieten. Durch A*-Suche können die besten Wege durch die Anwendung erkannt werden und so eventuell durch Entfernen oder Ändern von Verlinkungen die Benutzeraktionen beschleunigt werden. Durch die Eingabe von Testdaten ist es möglich die Validierung von Formularfelder zu prüfen.

Die Implementierung der hier beschriebenen Vorgehensweise ist teilweise realisiert worden, bedarf aber zukünftig noch erheblichen Aufwands um eine maximal mögliche Automatisierung bereitzustellen. Die Erwartungen also die Assertions sind nicht intensiv betrachtet wurden und stellen Forschungsstoff für zukünftige Arbeiten da, genau wie Erweiterungen des Graphen-Metamodells um mehrere Repräsentationen von Websiteelementen die z.B. Veränderungen von Webseiteformularen hervorrufen können. Dementsprechend muss das Testframework-Metamodell erweitert werden.

REFERENCES

- [1] El-Far, I.K. and Whittaker, J.A., *Model-Based Software Testing* Wiley Online Library, 2001.
- [2] Robinson, H., *Software Testing & Quality Engineering/ Intelligent test automation*. Sept./Oct. 2000.