„Negativ“-Tests interaktiver Systeme und ihre Automatisierung

Fevzi Belli[1], Michael Linschulte[1], Ralf Zirnsak[2], Günter Hofmann[2]

[1]Fakultät für Elektrotechnik, Informatik
und Mathematik
Universität Paderborn
Warburger Str. 100
33098 Paderborn
{belli, linschu}@upb.de

[2]fecher NordWest
Technologiepark 12
33100 Paderborn
{Guenter.Hofmann, Ralf.Zirnsak}
@fecher.de

Abstract: Die konventionelle, benutzungsorientierte Prüfung der Qualität von Software konzentriert sich aufs Testen des Systemverhaltens unter regulären Bedingungen. Der vorliegende Beitrag überprüft das Systemverhalten über diese „Schönwetter-Tests“ hinaus bei fehlerhaften, meist unerwarteten Benutzungseingaben („Negativ“-Tests). Dabei ist das Augenmerk auf Testautomatisierung gelegt, insbes. zur Präzisierung des Wertebereichs der Testeingabedaten, Verfeinerung der Testeingaben und Verdeutlichung kausaler Abhängigkeiten.

1 Einleitung und verwandte Arbeiten

Interaktive Systeme, die in sicherheitskritischen bzw. kapitalintensiven Umgebungen eingesetzt werden, verlangen nach einem hohen Maß an Zuverlässigkeit. Deswegen nimmt gerade die Prüfung dieser Systeme einen hohen Stellenwert ein, um Fehler aufzudecken, welche die Zuverlässigkeit und Benutzung der Systeme gefährden. Zu diesem Zweck werden Modelle und Techniken eingesetzt, die das System sowie die Interaktion des Benutzers mit dem System graphisch abbilden.

Die Modellierung und das Testen interaktiver Systeme mittels zustandsbasierter Modelle hat eine lange Tradition [1], [2]. Diese Ansätze analysieren das Modell des Prüflings und leiten aus Nutzeranforderungen Sequenzen von Nutzerinteraktionen ab, die als Testfälle genutzt werden. [3] führt dafür ein vereinfachtes zustandsbasiertes, graphisches Modell ein. Dieses Modell wurde in [4] um die Ereignisorientierung ergänzt und vor allem dahin gehend erweitert, um nicht nur *erwünschte*, sondern auch *unerwünschte* Situationen abzubilden. Etwas salopp ausgedrückt wird die erste Gruppe der Tests hier „Schönwetter-Tests“ genannt, die komplementäre Sichtweise dagegen „Negativ-Tests“. Damit entsteht eine *holistische* Sichtweise, die sich aus Überlagerung der beiden Sichten ergibt. [5] befasst sich mit dem Test von Web-basierten Systemen und erweitert den ereignisorientierten Ansatz aus [4] um die Berücksichtigung kausaler Abhängigkeiten der Eingabedaten. In [5] wurde der Test von unzulässigen Eingabedaten allerdings nur rudimentär berücksichtigt, d.h. die komplementäre Sichtweise von [4] vernachlässigt.

Die vorliegende Arbeit setzt den holistischen Ansatz ein; auf der Grundlage des in [5] vorgestellten Ansatzes für Web-basierte Systeme wird hier speziell auf Negativ-Tests und Strukturierung der Eingabedaten eingegangen. Zur Automatisierung des Testprozesses wird das Testwerkzeug TOSCA (*TOol for Structured testCase Administration*) benutzt, das speziell für den funktionalen Test von interaktiven Systemen entwickelt wurde. Der Ansatz ist zwar ursprünglich für den Test von Web-basierten Systemen entwickelt worden, kann aber auch allgemein zum Test von interaktiven Systemen benutzt werden, denn die meisten Web-basierten Systeme sind auch interaktive Systeme. Die hier fokussierte Sicht des Tests auf Dateneingabe ist ein zentraler Bestandteil der Benutzung-/System-Interaktion und damit nicht nur eine Komponente von Web-basierten Systemen, sondern generell auch von interaktiven Systemen, wie die Benutzung von grundlegenden Elementen zur Dateneingabe wie *Eingabefelder, Select-Boxen, Radio-Buttons* und *Check-Boxen* etc. zeigt.

Der nächste Abschnitt stellt zunächst das zugrunde liegende Modell aus [4] und [5] kurz vor, bevor im Abschnitt 3 das erstellte Modell anhand einer Fallstudie analysiert und der Ansatz erläutert wird. In Abschnitt 4 wird das Testwerkzeug beschrieben und erklärt, wie dieses Werkzeug den Testprozess signifikant unterstützt. Abschnitt 5 fasst die Arbeit zusammen.

2 Modell

Das Modell aus [5] nutzt den Ansatz aus [4] für die Modellierung von Systemverhalten, insbesondere für Interaktionen mit dem System aus Nutzersicht für reguläre Tests. Diesem Ansatz liegen Ereignissequenzgraphen zugrunde.

Ein *Ereignissequenzgraph* (ESG) ist ein gerichteter Graph mit einer endlichen Menge von Knoten und damit auch einer endlichen Menge von Kanten, welche die Knoten miteinander verbinden. Der ESG hat zusätzlich einen definierten Startknoten (‚[') und einen definierten Endknoten (‚]') mit der Eigenschaft, dass jeder Knoten des Graphen von dem Startknoten aus erreichbar ist und dass von jedem Knoten aus der Endknoten erreicht werden kann. Start- und Endknoten sind Pseudoknoten, welche legalen Anfang und erwartetes Ende einer Benutzung-/System-Interaktion angeben. Zusätzlich kann ein Knoten im ESG durch einen weiteren ESG strukturiert (sESG) werden (beispielhaft für den Knoten α in Abbildung 1).

Ein *Ereignis* ist ein von außen wahrnehmbares Phänomen, das im Verlauf der Operation des beobachteten Systems einen Stimulus durch den Benutzer oder eine Systemreaktion darstellt. Die Knoten des ESG repräsentieren diese Ereignisse (s. Abbildung 1). Sequenzen beliebiger Länge von ESG-Knoten stellen *Ereignissequenzen* (ES) dar; zwei aufeinander folgende Ereignisse werden auch als *Ereignispaar* (EP) bezeichnet. Eine ES ist *vollständig*, wenn sie mit dem Startknoten anfängt und mit dem Endknoten terminiert. Dadurch entstehen *vollständige ES* (vES).

Üblicherweise wird der Prüfling durch eine Menge von ESGs modelliert; jeder ESG stellt dabei eine (Teil-)Funktion des Prüflings dar. vES sind „erfolgreiche" Läufe durch

den ESG, d.h. sie stellen End-Ereignisse dar, welche von der Benutzung erwünscht werden. Durch diese Semantik wird das schwer zu behandelnde Orakel-Problem umgangen, das sich mit der Bestimmung der erwarteten Ergebnisse beim Testprozess befasst [9]. In Bezug auf ESG wird die erfolgreiche Ausführung der vES bis zum End-Ereignis erwartet. Wo immer das nicht möglich ist, wird der Test als nicht bestanden markiert.

ESGs sollten vor Beginn des Testprozesses verfügbar sein. Sie können aber auch inkrementell während des Testprozesses erstellt werden. Sollten andere Spezifikationsmittel, z.B. Statecharts, benutzt werden, kann der hier dargestellte Ansatz auch dann benutzt werden [10].

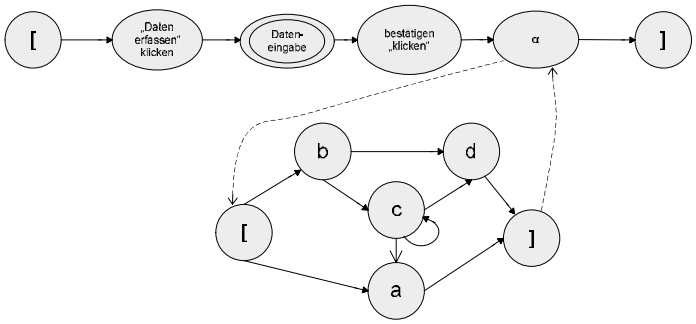


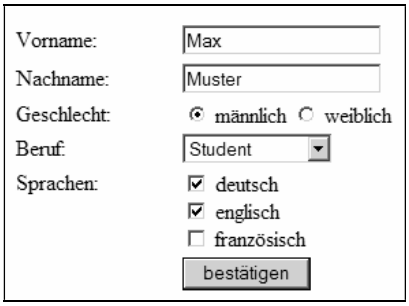
Abbildung 1: Beispiel eines ESG und Verfeinerung eines Knotens

2.1 Erweiterung des Modells durch Entscheidungstabellen

Die Dateneingaben werden in dieser Arbeit mittels *Entscheidungstabellen* (ET) modelliert und durch einen Doppelkreis im ESG kenntlich gemacht (s. Knoten Dateneingabe in Abbildung 1). ET sind tradierte Beschreibungsmittel in der Informationsverarbeitung (DIN 66241, Ausgabe: 1979-01) und werden auch als Hilfsmittel fürs Testen, wie z.B. in Ursachen-Wirkungsanalyse, benutzt (s. [8] und [9]). Eine ET verknüpft Bedingungen („Wenn") mit Ereignissen („Dann") und stellt die Auslösung der Ereignisse in konjunktiver Abhängigkeit von den Bedingungen dar. Für Verarbeitung komplexer Zusammenhänge im Testprozess können sie mit Hilfe der Booleschen Algebra operationalisiert werden [7]. Abbildung 1 zeigt schematisch die Erstellung und Verarbeitung eines Beispiel-Formulars zur Datenerfassung. Das Formular selbst und die dazugehörige ET sind in Abbildung 2 dargestellt. Der Knoten Dateneingabe aus Abbildung 1 wird in der Abbildung 2 durch eine ET verfeinert. Beruf wird in der ET mehrfach erfasst, weil einerseits alle möglichen Ausprägungen erfasst werden müssen und weiterhin das Nachfolgeereignis von der Ausprägung abhängt, die die Datenauswahl beeinflusst. Sind alle ESG und die dazugehörigen ET aufgestellt, können anschließend Sequenzen von Ein- und Ausgaben, d.h. ES, generiert und am System ausgeführt werden.

Zusammenfassend besteht das Modell, das dieser Ansatz benutzt, somit aus ESG, dessen Knoten verfeinert und durch ET präzisiert werden können. Dabei visualisiert der jeweilige ESG beliebige Sequenzen von Ereignissen, was die Erkennung von Unstimmigkei-

ten in der sequentiellen Ausführung von Benutzung-/System-Interaktion erlaubt. Die dazugehörige ET attributiert den ESG zur Unterstützung der Prüfung kausaler Zusammenhänge dieser Ereignisse. Damit entsteht ein bezüglich seiner Darstellungskraft mächtiges Hilfsmittel, das ein systematisches Testen komplexer Systeme erlaubt.



Dateneingabe	R1
Vorname $\in \Sigma^*$	-
Nachname $\in \Sigma^*$	-
Geschlecht $\in$ {männlich, weiblich}	-
Beruf $\in$ {Schüler, Student, Angestellter, Selbständiger}	-
Sprachen $\subseteq$ {deutsch, englisch, französisch}	-
Beruf=Schüler	F
Beruf=Student	T
…	
„bestätigen“ klicken	X

Abbildung 2: Eingabedatenformular (links) und zugehörige ET (rechts)

2.2 Komplementierung des Modells für Negativ-Tests

Wie bereits in der Einleitung angemerkt, fehlt bei der bisherigen Umsetzung des Modells in [5] die komplementäre Sichtweise aus [4] für die Dateneingabe. Dazu werden die Kanten im ESG durch gestrichelte Kanten (auch *Fehler-Kanten* genannt, die Negativ-Tests triggern) so ergänzt, dass (s. [4] und [5]):

- jeder Knoten eine auf sich selbst gerichtete Kante (*Schleife*) besitzt.
- jeder Knoten mit allen anderen Knoten durch eine Hin- und Rückkante verbunden ist.

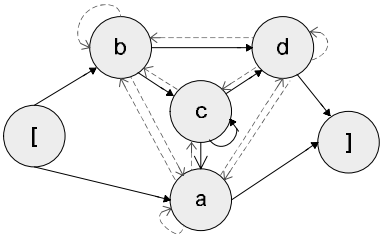


Abbildung 3:Vollständiger ESG

Ein Beispiel zeigt Abbildung 3 in Anlehnung an das Beispiel aus Abbildung 1. Zwei Ereignisse, die durch eine Fehler-Kante miteinander verbunden sind, werden *fehlerhaftes Ereignispaar* (FEP) genannt. Um diese FEP zu testen, werden Sequenzen gebildet, die mit dem Start-Knoten beginnen und mit dem FEP enden. Kann diese Sequenz nicht erfolgreich zum Ende geführt werden, gilt ein Test als bestanden, andernfalls als nicht bestanden („negative“ Logik!). Da jedes FEP durch eine eigene Sequenz abgetestet werden muss, erhöht sich allerdings auch der Aufwand für den Test, da wesentlich mehr Testfälle in Form von Testsequenzen erzeugt werden. Gültige EP hingegen können im Idealfall durch eine vES abgedeckt werden. Komplementiert man nun einen ESG mit Dateneingabeknoten nach [4], so ist zu klären, wie Fehler-Kanten in Verbindung mit Dateneingabeknoten zu interpretieren sind. Allgemein müssen drei Arten von Fehler-Kanten unterschieden werden:

1. Auf sich selbst gerichtete Fehler-Kanten (s. Abbildung 4)
2. Ausgehende Fehler-Kanten (s. Abbildung 5)
3. Eingehende Fehler-Kanten (s. Abbildung 6)

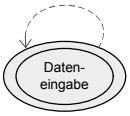


Abbildung 4:Auf sich selbst gerichtete Fehler-Kante

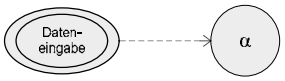


Abbildung 5:Ausgehende Fehler-Kante

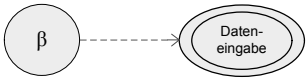


Abbildung 6:Eingehende Fehler-Kante

Sofern sich nicht dynamisch anhand einer Dateneingabe das Eingabeformular verändert, kann es keine auf sich selbst gerichtete Fehler-Kante geben. Dateneingaben können in diesem Fall beliebig oft revidiert werden und Dateneingabeknoten müssen bereits bei der Modellierung der zulässigen Eingabedaten mit einer entsprechenden Schleife versehen werden. Ändert sich jedoch das Eingabeformular dynamisch durch die Dateneingabe und eine Abänderung der gleichen Datenfelder ist nicht mehr möglich, so kann durch die selbstgerichtete Fehlerkante dieser Umstand nun getestet werden. Im 2. Fall, also einer Fehler-Kante von einem Dateneingabeknoten auf ein anderes Ereignis (s. Abbildung 5), darf das Nachfolgeereignis unabhängig von den eingegebenen Daten nicht zur Ausführung gebracht werden können. Für die spätere Ersetzung des Eingabedatenknotens in der Ereignissequenz durch Daten muss deswegen möglichst für jede Regel der ET ein Datensatz generiert werden. Das Nachfolgeereignis kann unter Umständen in Abhängigkeit von einer Regel der ET ausführbar sein. Eine eingehende Fehler-Kante (s. Abbildung 6) ist so zu interpretieren, dass bereits die Dateneingabe nicht möglich sein kann. Deswegen reicht die spätere Ersetzung des Dateneingabeknotens durch Daten einer beliebigen Regel der ET aus.

Abbildung 7: ISELTA – Ergebnisliste mit Suchverfeinerung

3 Eine Beispiel-Sitzung anhand einer nicht-trivialen Applikation

Anhand des kommerziellen Buchungs- und Verwaltungssystems ISELTA (*Isik's System for Enterprise-Level Web-Centric Tourist Applications*) wird der in Kapitel 2 und 3 beschriebene Ansatz beispielhaft erläutert. ISELTA ist ein virtueller Marktplatz für Leistungen der Touristik-Branche. Ziel dieses Systems ist es, Hoteliers und sonstigen Anbietern von Hotelzimmern die Möglichkeit zu geben, ihr Angebot einer breiten Masse von Verbrauchern über das Internet zur Verfügung zu stellen. Um dies zu erreichen, kann nach einer Erfassung des Hotels im System eine eigene Suchseite eingerichtet und mittels eines individuellen Links in eine bereits bestehende Internet-Seite integriert werden. Über diese Suchseite ist es dem Verbraucher dann möglich, sich Zimmerangebote herauszusuchen (Abbildung 7) und diese zu buchen.

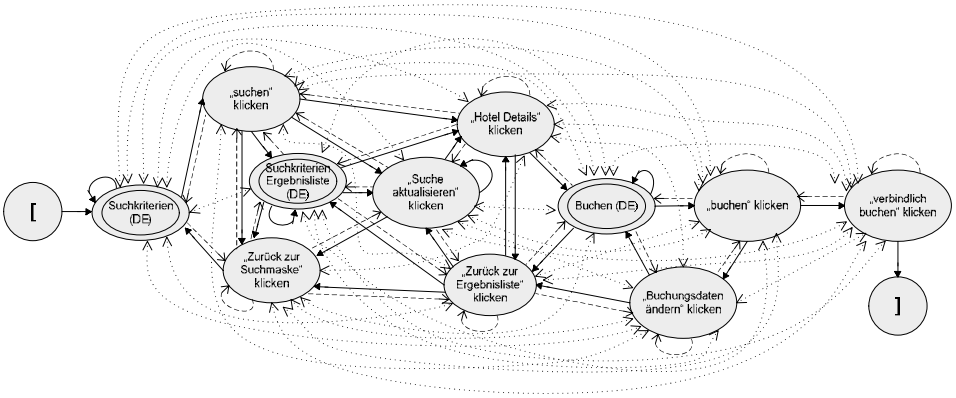


Abbildung 8: ESG zum Suchen und Buchen über ISELTA

In der Fallstudie wurde die Komponente des Suchens und Buchens, also die Schnittstelle des Systems mit dem Endkunden, getestet. Hierzu wurde das ESG-Modell (s. Abbildung 8) mit den ET aufgestellt, anschließend die Testsequenzen konstruiert und mittels des *TOSCA Commander*, der in Abschnitt 4 näher beschrieben wird, zur Ausführung gebracht.

Tabelle 1: ET zu „Suchkriterien Ergebnisliste"

ET – Suchkriterien Ergebnisliste		
Orte ∈ {alle Orte, Bielefeld, Paderborn}	-	-
Datum von ∈ {TT.MM.JJJJ}	-	-
Datum bis ∈ {TT.MM.JJJJ}	-	-
Verpflegung ∈ { ---, Frühstück, Halbpension, Vollpension, All Inclusive}	-	-
Datum von < Datum bis	T	F
„Suche aktualisieren" klicken	x	-

Tabelle 1 zeigt die ET zum Knoten Suchkriterien Ergebnisliste. Die zugrunde liegenden Eingabefelder verändern sich nicht dynamisch anhand der Dateneingabe. Der model-

lierte ESG besitzt 11 Knoten (ohne ‚[, und ‚]') mit 27 gültigen Kanten. Die Anzahl an Gesamtkanten in einem ESG entspricht der quadrierten Anzahl der Knoten. Der Beispiel-Graph besitzt somit 94 Fehler-Kanten. Dementsprechend existieren 27 gültige EP und 94 FEP. 3 Knoten sind Dateneingabeknoten, sodass 49 der FEP Dateneingabeknoten enthalten.

Die gepunkteten Kanten des ESG bilden die Fehler-Kanten für Knoten, die nicht verbunden waren. Die gestrichelten Kanten sind ergänzende Fehlerkanten zu gültigen Kanten oder fehlenden Schleifen. Die 27 gültigen EP können durch eine ES mit gültigen Eingabedaten komplett abgedeckt werden. Insgesamt wurden 123 Testsequenzen aufgestellt. Weil jedes FEP mindestens durch eine eigene Testsequenz abgedeckt werden muss (die Ausführung des FEP soll schließlich nicht möglich sein), dienten 115 der 123 Testsequenzen dem Test von FEP, 7 weitere zum Test ungültiger Eingabedaten bei sonst gültigem Nachfolgeereignis. Der Test von FEP oder ungültigen Daten gilt als bestanden, wenn das Nachfolgeereignis nicht ausführbar war.

Tabelle 2: Auszug der gefundenen Fehler

Nr.	Fehler
1	Nachdem „Hotel Details" geklickt wurde, konnte „Suche aktualisieren" geklickt werden
2	Nachdem „Hotel Details" geklickt wurde, konnte Dateneingabe für Suchverfeinerung noch vorgenommen werden
3	Straße keine Pflichteingabe bei der Eingabe der Buchungsdaten
4	Hausnummer keine Pflichteingabe bei der Eingabe der Buchungsdaten
5	Buchstaben als Telefonnr. möglich

Tabelle 2 enthält einen Auszug der gefundenen Fehler durch die Testsequenzen. Insgesamt wurden 12 Fehler gefunden. 2 Fehler (Nr. 1 und 2 von Tabelle 2) wurden dabei nur durch den Test der FEP gefunden. Dies zeigt, dass durch den Negativ-Test zusätzliche Fehler gefunden werden, die durch einen Positiv-Test nicht aufgedeckt werden können.

4 Zusammenspiel Modell – Testumgebung

4.1 Die Testumgebung TOSCA

Zur Unterstützung des in Abschnitt 2 beschriebenen Modells kommt die Testumgebung *TOSCA Commander* der Firma Triton zum Einsatz. Der *TOSCA Commander* wurde zum Durchführen und Verwalten von Tests an interaktiven Systemen entwickelt. Hierzu werden unter anderem Funktionen für Capture, Play und Reporting zur Verfügung gestellt. Tosca betrachtet den Prüfling als Black-Box und ist damit fokussiert auf die Ein- und Ausgabe zwischen einem Benutzer und dem zu prüfenden System. Der *TOSCA Commander* setzt für die Simulation eines Benutzers, also zur Steuerung des Prüflings, auf spezielle Adaptoren, um den unterschiedlichen Technologien interaktiver Systeme gerecht zu werden (s. Abbildung 9). Mittels eines Capture-Tools, dem *Wizard*, werden die vorliegenden Masken des Prüflings und die sich darauf befindenden Controls inkl. ihrer

qualitativen Merkmale erfasst und stehen anschließend innerhalb des *TOSCA Commander* als Module zur Verfügung. Der TOSCA Commander nutzt zur Steuerung dieser Controls fachliche Kriterien, so dass im Fall einer Änderung der Oberfläche in Bezug auf die Anordnung der Controls keine Korrekturen der Testfälle erforderlich werden.

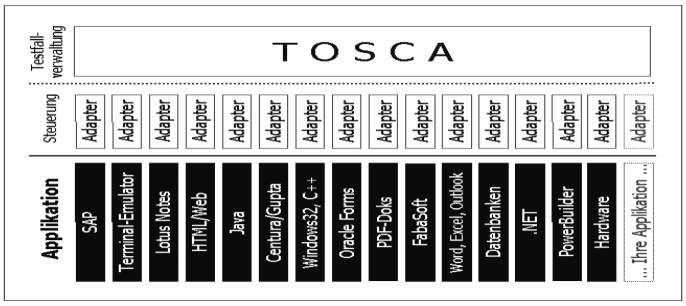


Abbildung 9: TOSCA-Adaptoren (Quelle: www.triton.at)

4.2 Automatisierung von regulären („Schönwetter“-)Tests

Anhand dieser Module können nun die gewünschten Interaktionen zu Testfällen zusammengestellt und mit Daten gefüllt werden. Dabei werden in der Praxis zunächst so genannte „Schönwettertests“ durchgeführt, die sicherstellen sollen, dass die wichtigsten Funktionen zur Verfügung stehen, also der Prüfling grundsätzlich wie gefordert bedienbar ist. Als Grundlage für diese ersten Testfälle werden Spezifikationen über die als wichtigste Geschäftsvorfälle identifizierten Interaktionen genutzt. Im Anschluss daran werden diese Testfälle sukzessive erweitert. Weiterhin kann ein erstellter Testfall als Vorlage für Testfälle dienen, die sich nur durch Eingabe-Daten unterscheiden. Dazu wird der Testfall in ein *Template* (s. Abbildung 10, schwarzer Rahmen) umgewandelt und mit Daten aus einer Excel-Anbindung verknüpft. In Abhängigkeit von den vorhandenen Datensätzen werden dann die Testfälle erstellt (s. Abbildung 10, grauer Rahmen). Die hierfür benötigten Eingabedaten werden in der Regel manuell ausgewählt. Der *TOSCA Commander* bringt von sich aus kein Schema zur systematischen Erstellung von Testfällen mit sich. Dieses ist als kritisch zu betrachten, weil es dem Tester überlassen bleibt, geeignete Testfälle auszuwählen und zu erstellen, sofern nicht entsprechende Testfälle gegeben sind. An dieser Stelle hilft das Modell des ESG, indem es den Tester bei der systematischen Testfallerstellung unterstützt.

Aus den ESG werden zunächst mit Hilfe der Software „GATE“ [6] ES generiert. Die ES beinhalten jedoch noch keine Daten aus den Entscheidungstabellen. Diese müssen manuell anhand der ET erstellt und zu den ES hinzugefügt werden. Nun müssen die ES von Hand im *TOSCA Commander* eingegeben werden. Bei der Eingabe der ES wird der Anwender durch den o.g. Wizard und die Modulbibliothek unterstützt, indem die aufgezeichneten Masken als Module zur Verfügung stehen. Die Module können per Drag&Drop beliebig kombiniert und wiederholt benutzt werden. Unterscheiden sich ES

nur durch unterschiedliche Daten, so können die jeweiligen Daten in einer Excel-Tabelle erfasst werden. Dazu wird die sonst identische ES als Testfall-Template erstellt und mit den Daten aus der Excel-Tabelle verknüpft. Anhand der Verknüpfung können die verschiedenen Testfallvarianten vom *TOSCA Commander* automatisiert erzeugt werden. Abschließend wird eine Ausführungsliste erstellt und ausgeführt. Das Ausführungsergebnis wird gespeichert und kann als Report für die Testdokumentation ausgegeben werden.

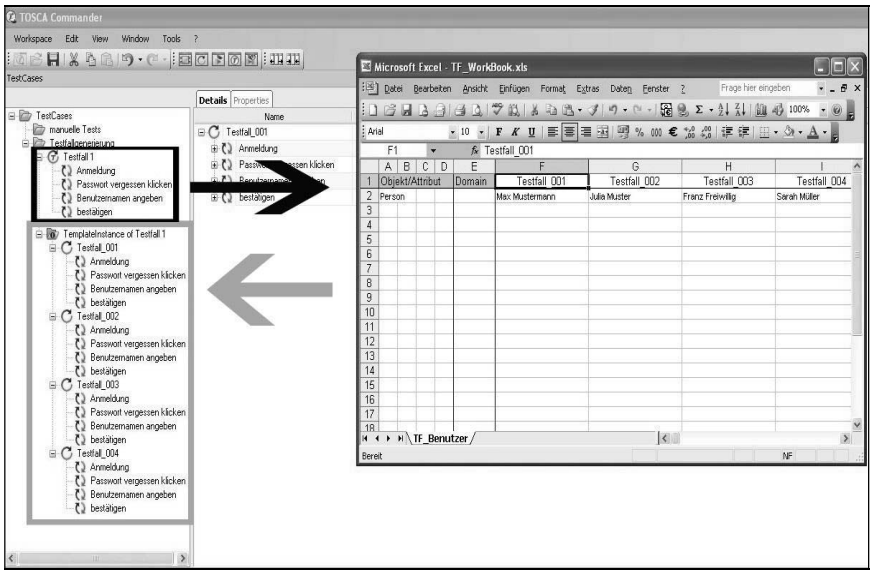


Abbildung 10: Excel-Anbindung im *TOSCA Commander*

4.3 Automatisierung von Negativ-Tests

Negativ-Tests werden mit dem *TOSCA Commander* in gleicher Weise formuliert wie die „Schönwetter-Tests“. Die für den Negativ-Test relevanten Daten liegen außerhalb der gültigen Wertebereiche des jeweiligen Objektes; die logische Abfolge der Eingaben wird bewusst nicht eingehalten. In diesen Fällen reagiert das zu testende System mit Warnhinweisen, die in allen Fällen bei der Testfallgestaltung bekannt sind und somit – wie auch bei den „Schönwetter-Tests“ - automatisiert werden können. Im Sinne des Testwerkzeuges werden Negativ-Tests ebenfalls positiv formuliert. Bei der Automatisierung von Negativ-Tests müssen die Reaktionen des Systems deshalb gleichermaßen antizipiert werden wie bei Positivtests. Dennoch benötigt das Testwerkzeug Verfahren, die nicht vorhersehbare Ereignisse behandeln. Der *TOSCA Commander* stellt hierfür ein Recovery zur Verfügung, das bei nicht geplanten Aktionen einsetzt und den Zustand des Gesamtsystems dokumentiert. In Abhängigkeit vom Fehlerfall kann das Recovery sehr weit reichende Aktionen selbständig durchführen, um einen stabilen Systemzustand wieder herzustellen, auf den die weiteren Tests aufsetzen können. Die unbeaufsichtigte Testdurchführung kann nur erreicht werden, wenn das Recovery bezogen auf die zu testende Software spezifisch reagiert.

5 Zusammenfassung und Ausblick

In diesem Beitrag wurde ein bestehender Ansatz zum Test interaktiver Systeme in Verbindung mit Eingabedaten betrachtet und um Negativ-Tests erweitert. In einer Fallstudie wurde der Ansatz evaluiert und gezeigt, dass weitere wichtige Fehler gefunden werden. Zusätzlich wurde ein Testwerkzeug zur Unterstützung des in [5] dargestellten Ansatzes vorgestellt. Es wurde beschrieben, wie das Werkzeug den Tester bei der Ausführung der aus dem Modell erstellten Testfälle unterstützt. Zwar wird einiges an Zeit und damit verbunden Personal-Kosten für die Eingabe der Testfälle in den *TOSCA Commander* benötigt, jedoch werden diese Kosten nur einmal investiert, sofern sich keine Änderungen im zugrunde liegenden Modell ergeben. Die Testfälle können somit unbegrenzt wiederholt werden. Dies ist gerade bei Weiterentwicklungen von Systemen sinnvoll. Was bisher noch nicht automatisiert durchgeführt werden kann, ist die Datengenerierung aus den ET. Zwar können über die ET die Daten zielgerichtet und systematisch ausgewählt werden, eine entsprechende Unterstützung durch Software fehlt allerdings bisher. Automatisierungspotential steckt zudem in der manuellen Eingabe der ES. Mit der Eingabe von Daten geht in der Regel auch der Wunsch nach der Überprüfung von Daten im System einher. Diese Funktion bietet der *TOSCA Commander*. Hier könnte der Ansatz aus [5] erweitert werden um *Checkpoints*, an denen Daten zwischenzeitlich überprüft werden.

Referenzen

[1] Parnas, D.L.: On the Use of Transition Diagrams in the Design of User Interface for an Interactive Computer System. In Proc. of ACM Nat'l. Conf., ACM Press 1969; S. 379-385.

[2] Shehady, R.K.; Siewiorek, D.P.: A Method to Automate User Interface Testing Using Finite State Machines. In Proc. Int. Symp. Fault-Tolerant Computing, 1997; S. 80-88.

[3] White, L.; Almezen, H.: Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences. In Proc. of Int. Symp. on Softw. Reliability and Eng., IEEE Comp. Press, 2000; S. 110-119.

[4] Belli, F.: Finite-State Testing and Analysis of Graphical User Interfaces. In Proc. of Int. Symp. on Softw. Reliability and Eng. IEEE Comp. Press, 2001; S. 34-43.

[5] Belli, F.; Budnik, C.J.; Linschulte, M.; Schieferdecker, I.: Testen Web-basierter Systeme mittels strukturierter, graphischer Modelle – Vergleich anhand einer Fallstudie. In Proc. Model-based Testing, 2006, LNI, Vol. 94, 2006; S. 266-273.

[6] Belli, F.; Budnik, C.J.; White, L.: Event-based Modeling, Analysis and Testing of User Interactions: Approach and Case Study. The Journal of Software Testing, Verification and Reliability. John Wiley & Sons, Vol. 16(3), 2006; S. 3-32.

[7] Goodenough, B.; Gerhart, S.L.: Toward a Theory of Test Data Selection. IEEE Trans. Softw.Eng., 1975; S.156-173.

[8] Myers, G.J.: The Art of Softw.Testing, John Wiley & Sons, 1979.

[9] Binder, R.V.: Testing Object-Oriented Systems, Addison-Wesley, 2000.

[10] Belli, F.; Budnik, C.J.; Hollmann, A.: Holistic Testing of Interactive Systems Using Statecharts. Journal of Mathematics, Computing & Teleinformatics (AMCT), Vol. 1(3), 2005; S. 54-64.