Testverfahren für eventgetriebene Software am Beispiel der ARTIS-Plattform

Robert Schwarzer [1], *Florian Adolf* [1], *Hans-Gert Gräbe* [2]

[1] Deutsches Zentrum für Luft- und Raumfahrt
Institut für Flugsystemtechnik
Abt. Systemautomation
Lilienthalplatz 7, 38108 Braunschweig
{robert.schwarzer, florian.adolf}@dlr.de

[2] Universität Leipzig
Institut für Informatik
Abt. Betriebliche Informationssysteme
Postfach 100920, 04009 Leipzig
graebe@informatik.uni-leipzig.de

Zusammenfassung: Die Systemsimulation ist für das Testen von eventbasierten Komponenten eines eingebetteten Systems nicht immer hinreichend genau. Es werden zusätzliche Strategien benötigt, um automatisiert und umfassend testen zu können. Mit dieser Arbeit wird ein modellbasierter Ansatz in Verbindung mit Unittest-Frameworks aufgezeigt.

1 Einleitung

Am DLR-Institut für Flugsystemtechnik in Braunschweig wird der Flugversuchsträger ARTIS (Autonomous Rotorcraft Testbed for Intelligent Systems) entwickelt. Es stehen dafür drei Plattformen (ARTIS, miniARTIS und maxiARTIS) auf der Basis von Modellhubschraubern zur Verfügung (siehe Abbildung 1). Neben Bordrechner und Datenlinks[1] ist ARTIS mit diversen Sensoren z.B. Satellitennavigation (GPS), Inertialplattform (IMU) und Magnetometer ausgestattet. Von zentraler Bedeutung ist zusätzlich der

1 WLAN und Funkmodem als redundante Kanäle

Einsatz von abbildenden Sensoren (z.B. Stereokamera). Im Mittelpunkt des Projektes stehen der Entwurf und die Erprobung von Algorithmen und Methodiken für autonome intelligente Funktionen. Dazu gehören u.a. die Betrachtung von den Mensch-Maschine-Schnittstellen, Flugregler- und Autonomiekonzepten sowie der Bildverarbeitung.

Abbildung 1: Die ARTIS-Familie

Vor jedem Flugversuch werden die integrierten Komponenten ausführlich in einer Systemsimulation unter Echtzeitbedingungen getestet. Es wird dabei ein Matlab/Simulink-Modell zur Simulation von Flugverhalten, Sensorik und den Aktuatoren eingesetzt. Die entwickelten Algorithmen werden als C/C++ Code in dieses Modell eingebunden. Bei der Systemsimulation wird in zwei Varianten unterschieden: Software-in-the-Loop Simulation (SITL) und Hardware-in-the-Loop Simulation (HITL). Während die SITL Simulation auf einem Computer als reine Software-Simulation stattfindet, wird die HITL Simulation auf mehrere Computer verteilt (siehe Abbildung 2). Es wird dabei die spätere Flugversuchshardware des ARTIS, ein echtzeitfähiger Simulations-Computer der Firma dSPACE sowie der Bodenstations-Computer verwendet.

Die ARTIS Systemsimulation weist in der bisherigen Form zwei Nachteile auf. Sie ist sehr zeitaufwendig, wenn alle Komponenten vollständig geprüft werden sollen und es mangelt an einer automatischen Protokollierung der Versuchsdurchführung. Die einzelnen Komponenten müssen somit schon vor der Integration ausführlich getestet werden. Im Rahmen dieser Diplomarbeit soll daher untersucht werden, wie sich die Komponenten des Entscheidungssystems vorab auf ihre Korrektheit prüfen lassen. Die Umsetzung des Testkonzepts soll, wie die eigentliche Systemerstellung selbst, modellbasiert erfolgen.

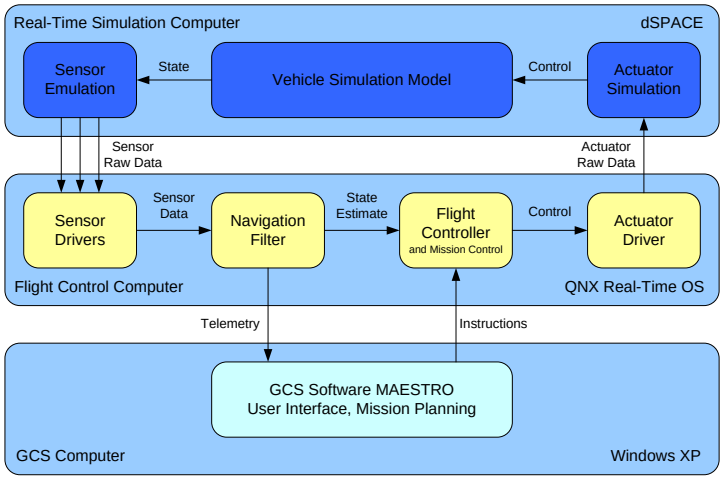

Abbildung 2: HITL Simulation

2 Grundlagen eventbasierter Systeme

In der Informatik wird der Begriff Event je nach dessen Kontext differenziert betrachtet. Zum einen werden Events bei der Programmierung von graphischen Benutzeroberflächen (GUI's) eingesetzt, zum anderen findet eine Verwendung auch bei der Prozesssynchronisation (Semaphoren, Mutexe) statt. Eine dritte Möglichkeit ist der Einsatz von so genannten Subject-Observer-Pattern im objektorientieren Design. Der Begriff Event wird auch in verschiedenen Programmiersprachen verwendet. Beispielsweise wird in Java ein Event als Objekt genutzt, wenn die Event-Quelle (Source) den Event-Empfänger (Listener) benachrichtigt will.

2.1 Events und Notifications

Trotz der Vielzahl von Anwendungsmöglichkeiten wurde in [6] versucht die beiden Begriffe Event und Notification zu definieren:

"*An event is a detectable condition that can trigger a notification. A notification is an event-triggered signal to send to a run-time-defined recipient.*"

Im Software-Kontext ist das Event nach dieser Definition die Ursache und die Mitteilung die Folge (siehe Abbildung 3). Der Mitteilungstyp kann wiederum unterschiedlich definiert werden. Beispielweise ist es möglich

die Mitteilung als Objekt zu definieren, dem auch weiterführende Eigenschaften und Funktionen zugeordnet sind.

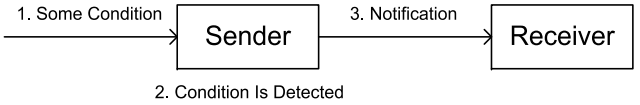

Abbildung 3: Zusammenhang zwischen Event und Notification

2.2 Event Subscription

Unter dem Begriff Event Subscription wird jener dynamische Prozess verstanden, der einen Event Publisher mit einem Event Empfänger (Subscriber) verbindet. Dabei veröffentlicht der Event Publisher zunächst eine Liste mit detektierbaren Events. Der Subscriber identifiziert in der Folge jene Events, an denen er interessiert ist und überträgt dies als Antwort. Der Event Publisher speichert diese Information und sendet die gewünschte Mitteilung, falls ein angefordertes Event detektiert wird.

Die Event Subscription kann auf vier möglichen Wegen erfolgen. Die elementarste besteht nur aus Event Publisher und Subscriber und wird als Direct-Delivery Model bezeichnet. Der Begriff Indirect-Delivery Model wird verwendet, wenn zwischen Event Publisher und Subscriber eine Middleware geschaltet ist. Das Senden der Event Subscriptions erfolgt direkt vom Subscriber oder aber über einen separaten Binder Agent. Dieser Agent ist in dem Direct-Delivery Model als auch dem Indirect-Delivery Model einsetzbar. Eine mögliche Klassifikation der Event Subscription wird in [4] und [6] vorgestellt. Die Unterteilung findet dabei in die Kategorien Channel, Type, Filter und Group statt.

Die Kategorie Channel definiert den physikalischen oder abstrakten Weg, wie die Mitteilung den Subscriber erreichen soll. In diesem Zusammenhang spielt der Begriff Event Mapping eine wichtige Rolle, da die Verteilung alle Mitteilungen entweder über einen Kanal (Single Channel) oder aber über mehrere Kanäle (Multiple Channels) erfolgen kann.

Die Kategorie Type grenzt die Events durch ihre unterschiedlichen Eigenschaften voneinander ab. Differenzierte Event Typen werden dabei durch verschiedene Eigenschaften symbolisiert. Zum Beispiel können die Event Typen für einen Button in einer GUI-Applikation Clicked, Moved oder Resized sein.

Ein Filter ermöglicht eine Auswahl von Events nach verschiedenen Kriterien (Filtering Expressions). Da sich das Filtern z.B. beim dem Content Filtering als aufwendig erweist, benutzen komplexere Systeme den Filter oft als separate Komponente.

Die Kategorie Gruppe repräsentiert Subscriber mit gleichen Event Subscriptions, welche somit die gleichen Events anfordern. Eine Gruppe kann daher als ein virtueller Subscriber betrachtet werden, wobei eine Überlagerung der Gruppen möglich ist.

3 Modellierung eventbasierter Systeme

Hauptsächlich werden eventbasierte Systeme mittels UML modelliert. Dazu werden State Charts, Aktivitäts-, Sequenz-, Kommunikations- und Interaktionsdiagramme genutzt. Es besteht auch die Möglichkeit, Petrinetze sowie Lollipop-, Espresso- und Catalysis-Diagramme einzusetzen [6]. Weiterhin existieren Ansätze zur Modellierung mit der Specification and Description Language (SDL). Der Fokus dieser Arbeit liegt auf der Modellierung mit den UML State Charts, da hierfür ein umfangreicher Tool Support gegeben ist. Eine weitere Arbeit[2] in dem Institut befasst sich parallel zu dieser mit der Modellierung über Petrinetze.

3.1 Überblick UML State Charts

State Charts sind Zustandsautomatendiagramme mit einer endlichen Menge an Zuständen, wobei zu jedem Zeitpunkt genau ein Zustand aktiv ist. Das Anwendungsgebiet umfasst vor allem Verhaltens- und Testspezifikation. Eine Klassifikation erfolgt in abstrakte und konstruktive State Charts, wobei die Übergänge zwischen beiden Varianten fließend sind. Abstrakte State Charts werden zu Dokumentationszwecken eingesetzt, konstruktive State Charts zur Code-Generierung. Die Generierung erfolgt im modellbasierten Kontext auf zwei unterschiedliche Varianten. Zum einen sind flache Automaten möglich, die sich auf Grund der optimierten Form[3] für eingebettete Systeme eigenen.

2 In Zusammenarbeit mit dem Institut für Verkehrssicherheit und Automatisierungstechnik der TU Braunschweig.

3 Reduzierung der modellierten Struktur (u.a. Zusammenfassung aufeinander folgender Abläufe)

Zum anderen können wiederverwendbare Automaten generiert werden, die durch ihr ausgeprägtes objektorientiertes Design kennzeichnend sind. Diese generierten Automaten sind für die Entwickler nachvollziehbarer, da die Modellstruktur beibehalten wird. Änderungen können daher via Roundtrip-Verfahren übersichtlicher eingepflegt werden.

Wichtige Elemente des State Charts sind die Zustände selbst, mit ihren internen Aktivitäten (Entry, Exit und Do), Pseudozustände wie der Initial-, der End- und der History-Zustand sowie Zustandsübergänge mit Event-Trigger, Wächterbedingung und Aktivität. Durch Kombination lassen sich zusammengesetzte bzw. orthogonale Zustände bilden. Mehr Informationen zu diesem Thema können u.a. in [1] nachgeschlagen werden.

Schon während der Modellierung können verschiedene Fehler auftreten. Dazu gehören u.a. isolierte oder unerreichbare Zustände sowie fehlende oder fehlerhafte Trigger und Guards. Ein generelles Grundproblem stellt das anschließende Testen dar, da theoretisch eine unendliche Menge an potentiellen Abläufen überprüft werden muss.

3.2 Umsetzung am Beispiel der Ablaufsteuerung

Als Anwendungsbeispiel ist für die Modellierung und das spätere Testen die Ablaufsteuerung als Komponente des Entscheidungssystems ausgewählt. Die Ablaufsteuerung erhält als Eingabe Status-Informationen des Modellhubschraubers, Direktkommandos und Verhaltenssequenzen von dem Operator der Bodenstation. Diese Eingaben wird aufbereitet und an den nachgeschalteten Flugregler[4] als Ausgabe weitergereicht.

Zu Beginn der Arbeit wurden mittels strukturierter Anforderungsanalyse die Spezifikationen an die zu modellierende Komponente zusammengetragen und als Use Cases formuliert. Die Methodik besteht dabei aus der Interviewtechnik (Projektleiter, Entwickler) und der Dokumentenanalyse von der ursprünglichen Spezifikation sowie dem aktuellen Quellcode. Eine Übersicht zur Methodik und entsprechender Werkzeugunterstützung ist mit [2] gegeben.

Die Ergebnisse der Vorbetrachtung sind mit dem Konzept eventbasierter Systeme verknüpft und mit dem Tool Rhapsody der Firma I-Logix hierarchisch modelliert. Die dazugehörige Spezifikation wird mit dem Konzept der Unittests verknüpft und ebenfalls modellbasiert erstellt.

4 PID- bzw. PID2-Regler

Rhapsody wurde aus zwei Gründen gewählt. Zum einen ist dieses Tool ein Industrie-Standard, der u.a. auch bei der Firma Airbus genutzt wird, zum anderen ist es möglich, auf Basis des Modells konstruktiv Code generieren zu können.

Die Abbildung 4 zeigt schematisch das erstellte Modell, das nachfolgend kurz erläutert werden soll. Durch Detektierung von periodischen und spontanen Eingabewerten erzeugt der Event Sender so genannte Low Level Events und benachrichtigt damit den Event Publisher.

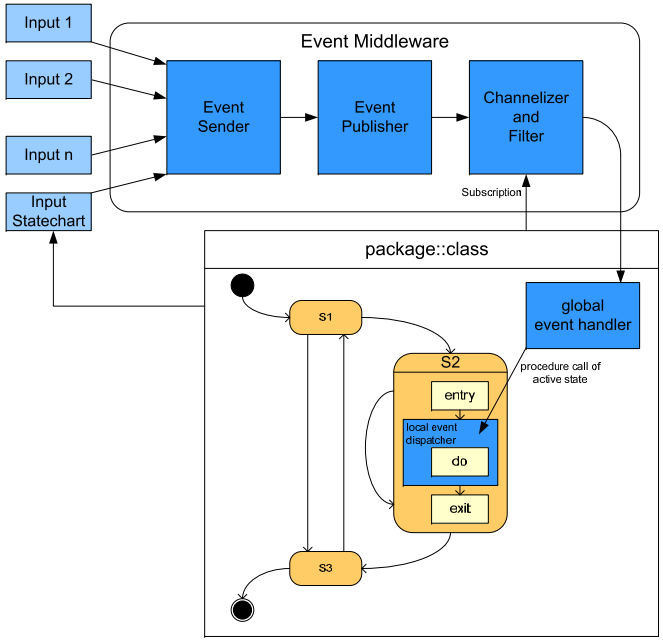

Abbildung 4: Schematische Darstellung des Modells

Dieser kombiniert die Informationen zu High Level Events, mit welchen die Transitionen des State Charts schaltbar sind. Zunächst werden die High Level Events jedoch durch den Event Filter unter Berücksichtigung des aktuellen State Chart Zustands und der vordefinierten Priorität gefiltert. Anschließend werden die Events an den Event Handler des State Charts übergeben. Dieser ruft die dazugehörige Prozedur für das Event Dispatching, also der Event-Verarbeitung, auf.

Bei der Modellierung ergaben sich in Bezug zur Echtzeitfähigkeit der Komponente die nachfolgenden Restriktionen: Die Event Middleware wird nur von dem IMU-Thread[5] im MCP[6] getriggert, d.h. der State Chart selber darf keine Anfrage eigenständig senden (z.B. während der Do-Aktivität). Des Weiteren darf pro Takt lediglich ein High Level Event generiert werden. Wird dies nicht eingehalten, können Sequenzketten als Folge einer Kettenreaktion auftreten, welche wichtige Berechnungen verzögern, so dass die für den Flugregler notwendigen Eingaben nicht rechtzeitig zusammengesetzt werden können.

Für die anschließenden Tests wurde das Modell so vereinfacht, dass die Transitionen ausschließlich durch Events geschaltet werden. Guards sind dabei in die Event Middleware verlagert und Actions in die nachfolgenden Entry-Aktivitäten des entsprechenden Zustands.

4 Teststrategien

Da der modellierte UML State Chart ein konstruktives Modell darstellt, eignet sich der Einsatz von White-Box-Tests. Diese Tests sollen hier auf einem Unittest-Framework (CppUnit) aufbauen und sowohl flache als auch wiederverwendbare Automaten gleichermaßen testen. Die vorgestellte Teststrategie wurde nach Grundelementen von [1] und [3] aufgebaut und erweitert. Es wird bei der Umsetzung der verschiedenen Test Suiten in abstrakte und detaillierte Test Cases unterschieden. Die abstrakten Test Cases überprüfen die Struktur des State Charts auf Kontrollflussbasis mit dazugehöriger Zustands-, Transitions- und Pfadüberdeckung.

Eine Test Suite für die Zustandsüberdeckung besitzt einen Test Case für jeden erreichbaren Zustand. Dieser wird als Sequenz von Eingaben ab dem Startzustand modelliert. Die Test Cases für die Test Suite der Transitionsüberdeckung zeichnen sich durch eine Sequenz von Eingaben in einem beliebigen Ausgangszustand bis zur Auslösung der Transition aus. Die Test Suite sollte somit alle schaltbaren Transitionen testen (siehe Abbildung 5).

5 Der Thread der Inertialplattform ist am zeitkritischsten.
6 Master Control Program der Onboard-Funktionalität des ARTIS

Da bei der Pfadüberdeckung die Menge der Pfade unter Berücksichtigung von Schleifen unendlich ist, wird diese auf die minimale Schleifenüberdeckung reduziert. Die Schleifen werden dabei mindestens einmal durchlaufen. Dieses Kriterium hat sich in der Vergangenheit als stark und dennoch praktisch erwiesen [1].

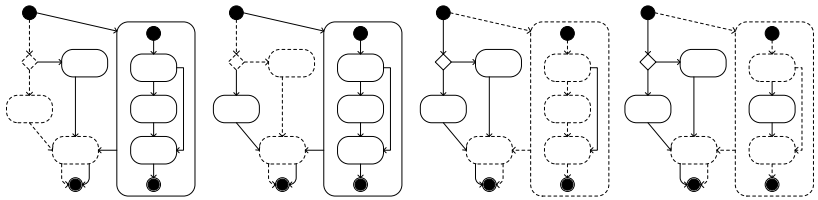

Abbildung 5: Transitionsüberdeckung als Beispiel

Bei der Auswahl der Testumgebung muss darauf geachtet werden, dass alle Transition schaltbar und damit alle Zustände erreichbar sind. In einer Simulationsumgebung wie SITL oder HITL ist dies nicht immer möglich. Dieser Umstand muss in den zu konzipierenden Tests berücksichtigt werden.

Des Weiteren ist es sinnvoll die Testdurchläufe zu protokollieren, damit ein Gesamtbild über die getestete Komponente gebildet werden kann. Wenn beispielsweise einige Zustände nicht geprüft sind, dürfen diese in einem späteren Flugtest nicht ausführbar sein.

Ein weiterer Testansatz, der in dieser Arbeit allerdings nicht berücksichtigt wird, ist das Model Checking. Dabei wird das Modell als Kripke-Strukur erstellt. Die Spezifikation wird mittels temporaler Logik (z.B. Computation Tree Logic, CTL) formuliert. Anschließend wird das Modell gegenüber der Spezifikation verifiziert. Mehr Informationen zu Thema Model Checking für eventbasierte Systeme sind u.a. in [5] zu finden. An dieser Stelle sollte außerdem das Open Source Tool UPPAAL[7] erwähnt werden, mit welchem Model Checking elegant durchführbar ist.

7 www.uppaal.com

4.1 Test Suite für die Event Middleware

Die Funktionalität der Event Middleware soll mit dieser Test Suite überprüft werden. Dazu wird zunächst das State Chart Modell durch ein abgeleitetes Modell ersetzt. Es werden in diesem abgeleiteten Modell die Methoden des Event Handlers und der Event Subscription überladen und zusätzlich eine Methode für das Setzen und Auslesen von Zustand und Event eingefügt.

Nach einer Grenzanalyse der verschiedenen Auslösebedingungen für die Low Level Events werden die notwendigen Eingabewerte für die Event Middleware gesetzt. An dieser Stelle kann erstmals die Detektierung der Auslösebedingungen überprüft werden. Da die Event Supscription durch das abgeleitete Modell manipulierbar ist, wird nun der gewünschte Zustand gewählt und anschließend die Event Middleware ausgeführt.

Diese verarbeitet die gegebenen Eingabewerte sowie die manipulierte Event Subscription und sendet anschließend ein High Level Event in Richtung des abgeleiteten Modells. Diese Mittelung wird durch den überladenen Event Handler abgefangen und kann abschließend ausgewertet werden. Um eine möglichst hohe Testabdeckung der Funktionalität zu erreichen, wird in dieser Test Suite jede denkbare Kombination von Eingabewerten und State Chart Zuständen geprüft.

4.2 Test Suite für die State Chart Struktur

Damit die logische Struktur des State Charts getestet werden kann, wird in dieser Test Suite zunächst die Event Middleware durch einen Event Generator ersetzt. Dieser ermöglicht die Generierung verschiedener Eventketten beliebiger Länge. Des Weiteren wird die Funktionalität der internen Aktivitäten der Zustände deaktiviert, damit der Strukturtest nicht durch Seiteneffekte verfälscht wird. Es bieten sich dafür zwei Ansätze an: Zum einen kann dies durch die Konfiguration vor der Code-Generierung umgesetzt werden, zum anderen ist auch der Einsatz von Präprozessor-Direktiven möglich.

Ein Test Case besteht in dieser Test Suite somit aus der sequentiellen Abarbeitung der generierten Eventkette. Es wird dabei von dem Startzustand ausgehend jeder Folgezustand ausgewertet. Damit sich der Testaufwand reduzieren lässt, wird das Prinzip der minimalen Schleifenüberdeckung berücksichtigt.

Die Vollständigkeit der State Chart Modellierung (complete, ignore oder chaos) bestimmt die möglichen Transition und muss somit berücksichtigt werden. Es lassen sich diesbezüglich allerdings gewisse Einschränkungen treffen, eine vollständige Modellierung ist sehr aufwendig und nicht praktikabel, d.h. sie wird fast nie zu testen sein. Im Gegensatz dazu ist ein chaotisches Verhalten bei eingebetteten Systemen eher unerwünscht und wird daher praktisch nicht verwendet. Der Event Handler der Ablaufsteuerung ist so modelliert, dass dieser für einen Zustand unbekannte High Level Events ignoriert. Daher ist es folgerichtig, nicht erforderliche Eventketten mindestens einmal überprüfen zu lassen, um diese Art der Modellierung zu testen.

4.3 Test Suite für die State Chart Funktionalität

Nachdem bereits die Funktionalität der Event Middleware und die Logik der State Chart Schaltung durch die ersten beiden Test Suiten überprüfbar ist, muss abschließend die Funktionalität der internen Aktivitäten der Zustände selbst getestet werden.

Hierfür wird der Test Case wie folgt modelliert: Zunächst wird der State Chart in den zu testenden Zustand gesetzt. Die Eingabewerte werden manipuliert, so dass eine Veränderung durch die interne Aktivität überprüfbar ist. Anschließend wird ein generiertes Event zur Auslösung der Do-Aktivität dem Event Handler übermittelt. Die interne Aktivität des Zustands wird somit durchlaufen und kann ausgewertet werden.

Das Senden eines generierten Events wird wiederholt, wobei diesmal die Exit-Aktivität sowie die Entry-Aktivität des Folgezustands durchlaufen werden soll. Die erwartete Ausgabe muss nun mit der tatsächlichen Ausgabe verglichen werden. Durch die Kombination der verschiedenen Test Cases ist es möglich, eine hohe Testüberdeckung zu erzielen.

5 Ausblick

Das händische Erstellen der vorgestellten Test Suiten ist nach vorausgehender Analyse eher trivial. Viele Test Cases ähneln sich bzw. werden aus anderen Test Cases zusammengesetzt. Auffällig ist auch, dass die Abläufe stets als Automat modellierbar sind. Aus diesem Grund soll für die verbleibende Zeit der Diplomarbeit exemplarisch gezeigt werden, wie das Testkonzept als modellbasierter Testautomat umgesetzt werden kann.

Ein solcher Testautomat kann als State Chart modelliert und hierarchisch aus den drei folgenden Komponenten aufgebaut sein (siehe Abbildung 6). Die erste Komponente (Test Key Generator) generiert einen (binären) Schlüssel, welcher für die beiden nachfolgenden Komponenten die Abfolge der Transitionen festlegt. Dieser Schlüssel bestimmt beispielsweise für die Test Suite der Event Middleware die Konfiguration der Eingabewerte zur Detektierung der Low Level Events.

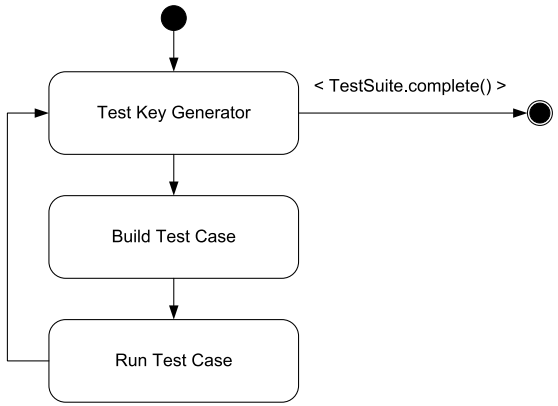

Abbildung 6: Grundkomponenten des Testautomaten

Die zweite Komponente (Build Test Case) wird anschließend nach Vorgabe des Schlüssels durchlaufen. Es werden nun die notwendigen Eingabewerte für den Test Case gesetzt. Beispielsweise wird für die zweiten Test Suite in dieser Komponente die Eventkette generiert.

Die dritte Komponente (Run Test Case) führt den nun vorbereiteten Test Case durch und wertet das Ergebnis aus. Im Fehlerfall ist dabei ein ausführlicher Report über den Test Case selbst, die eigentliche Spezifikation sowie das tatsächliche Verhalten der Ablaufsteuerung angedacht.

Durch diesen modelbasierten Ansatz und die zusätzliche Integration des Unittest-Frameworks in die Testautomaten sind diese den händisch implementierten Test Suiten in zwei Punkten überlegen: Es wird in einer abstrakteren Sprache übersichtlicher modelliert und es können durch Kombinatorik alle erforderlichen Testszenarien abgedeckt werden. Die Testautomaten ermöglichen somit das automatisierte Testen in einer sehr ausführlichen Form.

Literatur

[1] B. Rumpe: Agile Modellierung mit UML. Xpert.press, Springer Verlag, 2005.

[2] C. Hood, A. Kreß, R. Stevenson, G. Versteegen und R. Wiebel: Anforderungsmanagement: Methoden und Techniken, Einführungsszenarien und Werkzeuge im Vergleich. iX Studie, Heise Zeitschriften Verlag, 2005.

[3] E. Dustin, J. Rashka und J. Paul: Software automatisch testen. Xpert.press, Springer Verlag, 2001.

[4] G. Mühl, L. Fiege und P. Pietzuch: Distributed Event-Based Systems. 1st Edition, Springer-Verlag, 2006.

[5] K. Schneider: Verification of reactive systems : formal methods and algorithms. Berlin, 1st Edition, Springer-Verlag, Texts in Theoretical Computer Science (EATCS Series), 2004.

[6] T. Faison: Event-Based Programming: Taking Events to the Limit. 1st Edition, Apress, Berkely, CA, 2006.