Dipl.-Inform. M.Sc.
Marc-Florian Wendland

(E-Mail: marc-florian.wendland@fokus.fraunhofer.de)
studierte Informatik an der BHT (ehemals TFH) Berlin. Seit 2008 arbeitet er beim Fraunhofer Institut FOKUS, Kompetenzzentrum MOTION, zunächst als Student, seit 2009 als wissenschaftlicher Mitarbeiter. Seine Arbeiten umfassen die Bereiche modellgetriebene Software-Entwicklung und modellbasiertes Testen auf Basis der Standardspezifikationen UML, UTP und TTCN-3. Herr Wendland ist iSQI-zertifizierter TTCN-3-Experte und Vorsitzender der aktuellen UTP Revision bei der OMG.

Prof. Dr.-Ing.
Ina Schieferdecker

(E-Mail: ina.schieferdecker@fokus.fraunhofer.de)
beschäftigt sich mit Fragen der modellbasierten Software-Entwicklung, der Analyse, des Testens und Bewertung softwareintensiver Systeme und der Automatisierung und Optimierung von Prozessen zur Software-(Weiter-)Entwicklung und Qualitätssicherung. An der Technischen Universität Berlin leitet sie das Fachgebiet Entwurf und Testen von kommunikationsbasierten Systemen und ist am Fraunhofer Institut FOKUS, Berlin, Leiterin des Kompetenzzentrums MOTION.

Dipl.-Inform.
Andreas Hoffmann

(E-Mail: andreas.hoffmann@fokus.fraunhofer.de)
studierte an der Humboldt-Universität zu Berlin. Seit 1997 arbeitet er als wissenschaftlicher Mitarbeiter am Fraunhofer Institut FOKUS im Bereich der Modellierung und des Testens komplexer, verteilter Systeme. Er war Mitglied zahlreicher Standardisierungsarbeiten bei der OMG, ETSI und ITU. Seit 2010 ist er stellvertretender Leiter des Kompetenzzentrums MOTION.

Modellbasiertes Testen mit Fokus!MBT

Nachdem modellbasierte Entwicklungsmethoden in den letzten Jahren zunehmend Einzug in die Entwicklungsabteilungen gehalten haben, ist das modellbasierte Testen (MBT) verstärkt in den Fokus von Forschung und Industrie gerückt, um die Vorteile der modellbasierten Softwareentwicklung auf den Testprozess zu übertragen. Nach [Utting] steht das klassische Testen immer noch vor denselben Problemen wie vor zehn Jahren: die Systematik bei der Erhebung von Testfällen ist oftmals nicht nachvollziehbar, wodurch sich der Gesamtprozess nicht reproduzieren lässt; die Dokumentation des Testprozesses ist oft mangelhaft oder gar nicht vorhanden, was beim Ausscheiden eines Testers zu signifikantem Verlust von Wissen führt; oft erfährt der Beruf des „Testers“ generell eine zu geringe Wertschätzung. Nach Uttings Einschätzung (sowie zahlreicher weiterer Experten) ist MBT ein viel versprechender Ansatz, um einige dieser Probleme zu lösen, mindestens jedoch deutlich abzuschwächen.

Beim modellbasierten Testen werden die genutzten beziehungsweise erstellten testspezifischen Artefakte durch Modellartefakte beschrieben. Diese testspezifischen Modellartefakte werden in Testmodellen gebündelt. Auch der Testprozess selber kann ganz oder teilweise modelliert werden, was in diesem Artikel nicht weiter betrachtet wird. Durch gezieltes Auslassen irrelevanter Details wird die Beschreibung der Tests auf ein erhöhtes Abstraktionsniveau gehoben. Diese abstrakte Sicht ist notwendig, um die Komplexität des zu testenden Systems (engl. *system under test* – SUT) und der damit verbundenen Tests besser zu überschauen.

Ein Metamodell für das Testen

Ein Test-Metamodell integriert Konzepte zur modellbasierten Beschreibung von Tests. Neben strukturellen Aspekten und der verhaltensspezifischen Benutzungsbeschreibung des SUT umfasst dies Konzepte für Testdaten sowie für Testergebnisse. Ein Testmodell ist die Instanziierung eines Test-Metamodells und wird für die Spezifikation und Dokumentation von Tests als auch die Durchführung von Testprozessen (oder einzelnen Testphasen) eingesetzt. Interessanterweise existiert keine einheitliche Definition, welche Konzepte als *testrelevant* zu betrachten sind. Zu den testspezifischen Aspekten zählen wir insbesondere folgende komplementäre Einzelaspekte: Testziele, Testarchitekturen, Testverhalten, Testdaten, Testdirektiven, Testausführung, Testergebnisse und Teststrategien.

Keines der verfügbaren Test-Metamodelle [UTP, TPTP, TTCN-3] deckt alle diese Konzepte a priori ab. Für den Entwurf der Fokus!MBT-Werkzeugkette stand daher zur Diskussion, mehrere bereits existierende Test-Metamodelle parallel zu verwenden. Diese Option muss jedoch die modellübergreifende Konsistenzprüfung beziehungsweise -wahrung berücksichtigen. Die Verteilung einzelner Belange auf mehrere Test-Metamodelle erfordert eine stete Synchronisation zwischen den beteiligten Testmodell-Instanzen. Dies kann mitunter zu unlösbaren Konflikten oder Inkonsistenzen führen. Durch die Verwendung eines dedizierten Test-Metamodells wird diesen Problemen vorgebeugt. Der Vorteil eines integrierten und konsistenten Metamodells ist seine Kohäsion. Sämtliche als relevant erachteten Testkonzepte sind in einem einzigen Modell abgebildet. Dabei ist sicherzustellen, dass die verschiedenen Konzepte nicht nur syntaktisch, sondern auch semantisch zu einander passen, was eventuell zu Lasten der Kompatibilität mit dem Metamodell führt, aus welchem das oder die Konzepte entliehen wurden.

Eines der Ziele der Fokus!MBT-Werkzeugkette war der Entwurf eines Test-Metamodells, welches in der Lage ist, Tests und die in verschiedenen Testprozessen (oder einzelnen Phasen) genutzten Artefakte kohärent zu beschreiben. Das in diesem Artikel vorgestellte Test-Metamodell

Architecture	Behavior	Data	Time	Purpose	Generation	Execution	Deployment	Environment
SUT	Test Case	Data Pool	Timer	Test Purpose	Generation Directive	Execution Trace	Deployment Configuration	Setup Directive
Test Component	Message	Data Partition	Timer Events	Requirement	Coverage Goal	Verdict	Machine	Teardown Directive
Test Context	Events	Partition Rules	Timezone	Requirement Coverage	Stop Criteria	Verdict Reason	Artifact	Type Mapping
Test Control	Validation Action	Partition Instances		Requirement Verdict		Verdict Traces	Manifestation	Interface Mapping
Arbiter	Default	Wildcards				Execution History	Location	
	Logging	Coding Schemas						

Abbildung 1: Konzepte für das TestingMM

trägt den Namen *TestingMM* und bildet eine Grundlage für dieses Ziel. Das TestingMM wurde initial im Rahmen des EU-Projekt Modelplex [Modelplex] spezifiziert und implementiert. In der Spezifikationsphase wurden zunächst die testrelevanten Teilbereiche identifiziert, die vom TestingMM zu erfüllen sind. Ausgangspunkt für Analyse und Implementierung war das MOF-basierte Metamodell des *UML Testing Profils* [UTP], welches eine solide Basis testrelevanter Konzepte definiert. Diese wurden anschließend durch weitere Testkonzepte angereichert. Abbildung 1 stellt das Ergebnis der Analyse dar.

Im darauffolgenden Schritt wurden bestehende und etablierte Modellstandards selektiert und analysiert. Ziel war es, entsprechende Testkonzepte zu extrahieren und diese in das TestingMM zu integrieren. Gegenstand dieser Prüfungen waren vor allem die OMG-Standards UML und SysML sowie das Metamodell des Eclipse Test and Performance Tools Platform-Projekts (TPTP). Das TestingMM ist als konzeptionelle Integration von Teilbereichen dieser Metamodelle zu verstehen.Abbildung 2 visualisiert diese Zusammenführung schematisch.

Strukturelle Spezifikation des TestingMM

Die Einteilung der Konzepte in Abbildung 1 spiegelt sich im Aufbau des TestingMM wider. Jede der Spalten resultiert in einem eigenen Metamodell-Paket (Package). Zu den reinen Testkonzepten ist das Paket *Foundation* hinzugekommen, das die objektorientierten Grundkonzepte der UML abbildet und somit als Kern des gesamten TestingMM fungiert. Demnach besteht das gesamte TestingMM aus zehn Paketen, welche nachfolgend kurz beschrieben werden:

1. Foundation. Das *Foundation*-Paket definiert die grundlegenden Konzepte des objektorientierten Paradigmas und ist strukturell sowie semantisch eng an das Kernel-Paket der UML Superstruktur angelehnt. Es dient vor allem dazu, dem TestingMM eine paketorientierte Struktur zu verleihen (ähnlich zu UML-Modellen) sowie Typsysteme und Schnittstellenstrukturen zu beschreiben.
2. TestArchitecture. Als Testarchitektur wird der strukturelle Aufbau von Tests, Testsuiten und Testumgebungen bezeichnet. Das TestingMM übernimmt dieses Paket nahezu unverändert aus dem UTP. Zentrale Metaklasse dieses Konzeptbereichs ist der *TestContext*.
3. TestBehavior. Um Testfälle ausführen zu können, bedarf es einer eindeutig interpretierbaren Verhaltensbeschreibung. Testfallverhalten ist im TestingMM als simplifiziertes UML Sequenz-Diagramm implementiert, angereichert mit dedizierten Testkonzepten (beispielsweise Validierungs-Events).
4. TestData. Das *TestData*-Paket beschreibt Konzepte zur Erstellung von Datenpartitionen basierend auf dem Typsystem des SUT. Eine Datenpartition setzt dabei eine (oder mehrere) Parameter einer Schnittstellenoperation des SUT mit einem logischen Muster in Verbindung, welches die validen und invaliden Repräsentanten der Partition beschreibt. Auf diese Art ist es beispielsweise möglich, Äquivalenzklassen- und Grenzwertanalysen zu unterstützen, sowie konkrete Instanzen für jede identifizierte Datenpartition zu generieren. Weitere Aspekte des *TestData*-Pakets sind Wildcards für Nachrichtenargumente oder Codierungsinformationen für die technische Realisierung logischer Typsysteme.
5. TestTime. Das TestingMM implementiert einen einfachen *Timer*-Mechanismus auf Basis des UTP-Pakets *TimeConcepts*. *Timer*-Aktionen werden innerhalb der Interaktionen auf den Lebenslinien der Testkomponenten platziert und ausgeführt. Darüber hinaus ermöglicht das *TestTime*-Paket die Synchronisierung und Koordinierung von Testkomponenten in einer verteilten Testumgebung mittels Zeitzonen.
6. TestGeneration. Das *TestGeneration*-Paket wird für die Definition von Testdirektiven verwendet. Eine Testdirektive steuert oder beschreibt die Ableitung (manuell oder automatisiert) eines Systemmodells in ein oder mehrere Testkontexte des Testmodells. Die Ableitung kann dabei auf Basis von Gruppierungen und Namensänderungen erfolgen [Dai]. Eine andere Möglichkeit ist die Berücksichtigung spezifizierter Abdeckungskriterien für die Traver-

Foundation	Test Architecture	Test Behavior	Test Data	Test Time	Test Purpose	Test Generation	Test Execution	Test Deployment	Test Environment
Classifier	SUT	Test Case	Data Pool	Timer	Test Purpose	Generation Directive	Execution Trace	Deployment Configuration	Setup Directive
Interface	Test Component	Message	Data Partition	Timer Events	Requirement	Coverage Goal	Verdict	Machine	Teardown Directive
Port	Test Context	Events	Partition Rules	Timezone	Requirements Coverage	Stop Criteria	Verdict Trace	Artifact	Type Mapping
Type	Test Control	Validation Action	Partition Instances		Requirement Verdict		Execution History	Manifestation	Interface Mapping
Signal	Arbiter	Default	Wildcards					Location	
Operation		Logging	Coding Schemas						
Connection									

Concepts related to

UML	UTP	Newly Introduced	SysML	TPTP

Abbildung 2: Wiederverwendung relevanter Standards für das TestingMM

sierung von Zustandsmaschinen des Systemmodells [Stefanescu].

7. TestPurpose. Jeder Testfall setzt einen bestimmten Zweck um, nämlich die Verifikation einer oder mehrerer implementierter Anforderungen des SUT. Das TestingMM beschreibt mit der Metaklasse *TestPurpose* eine n:m-Beziehung zwischen Testfällen und Anforderungen und stellt damit eine konzeptionelle Vereinigung der Stereotypen «TestObjective» von UTP und «verifies» von SysML dar.
8. TestExecution. Ein Testfall beschreibt das erwartete Verhalten eines SUT in Abhängigkeit zu den applizierten Stimuli. Im TestingMM wird die tatsächliche Ausführung eines Testfalls in Form von sequenziellen Events (engl. *Traces*) notiert. Dies ermöglicht detaillierte Analysen über den Verlauf einer Testfallausführung.
9. TestDeployment. Aus der UML wurden die Konzepte zur Beschreibung der physischen Repräsentation logischer Artefakte des Testmodells auf einem Zielsystem entliehen. Mit dem TestDeployment-Paket lässt sich die Verteilung auf diesem Zielsystem beschreiben und, bei gegebener Werkzeugunterstützung, automatisiert ausführen.
10. TestEnvironment. In Abhängigkeit zu dem gewählten Testsystem und dessen Umgebung müssen die abstrakten, logischen Artefakte auf eine konkrete technische Plattform übertragen werden. Dieses Mapping lässt sich mit den Mitteln des *TestEnvironment*-Pakets ausdrücken.

Implementierung des TestingMM

Das TestingMM repräsentiert die Realisierung des MOF-basierten Metamodells des UTP, ergänzt um die in Abbildung 1 identifizierten weiteren testrelevanten Konzepte. Implementiert wurde das TestingMM mit dem *Ecore*-Metamodell des *Eclipse Modeling Framework* [EMF] unter Zuhilfenahme MOF-spezifischer Funktionalitäten wie *Property Subsetting und Redefinitionen*, um die Nähe zu den entsprechenden UML-Konzepten zu wahren. Da EMF diese Funktionalität per se nicht bietet, wurde auf das entsprechende Listen-Framework des Eclipse UML2-Projekts zurückgegriffen.

Weitere Arbeiten

Die vorgestellten Pakete und Konzepte des TestingMM vereinigen bis dato die wesentlichen Konzepte für das Testen und werden kontinuierlich erweitert und verfeinert. So wird das TestingMM im ITEA2-Projekt VERDE [VERDE] weiterentwickelt und um spezifische Konzepte zur Unterstützung von Echtzeit- und eingebetteten Systemen sowie von risikobasiertem Testen erweitert.

Fokus!MBT – eine flexible, erweiterbare Werkzeugkette

Software-Projekte in verschiedenen technologischen Domänen (z. B. Transportwesen, Finanzbereich, Luft- und Raumfahrttechnik) unterscheiden sich in Vorgehensmodellen, Werkzeugen und späteren Einsatzzwecken der zu entwickelnden Software signifikant. Es differieren auch Testziele und Testtiefe zum Teil deutlich. Sicherheitskritische Systeme wie die Geschwindigkeitsmesser eines Flugzeugs erfordern eine andere Herangehensweise als das Entertainmentsystem desselbigen. Fallen zum Beispiel die Videobildschirme aus, ist das allenfalls ärgerlich und führt zu gelangweilten Passagieren. Versagen hingegen die Geschwindigkeitsmesser, kann dies weit verheerendere Folgen haben. Auch wenn Software-Projekte hinsichtlich ihrer intrinsischen Charakteristika differieren, so ist Testen immer ein Bestandteil, der auf generischen Konzepten, Abläufen, etc. beruht. Auf dieser Erkenntnis setzt die Fokus!MBT-Werkzeugkette auf.

Architektur der Fokus!MBT-Werkzeugkette

Bei Fokus!MBT handelt es sich um einen flexiblen Verbund dedizierter Werkzeuge, die je nach Bedürfnis zu einer integrativen,

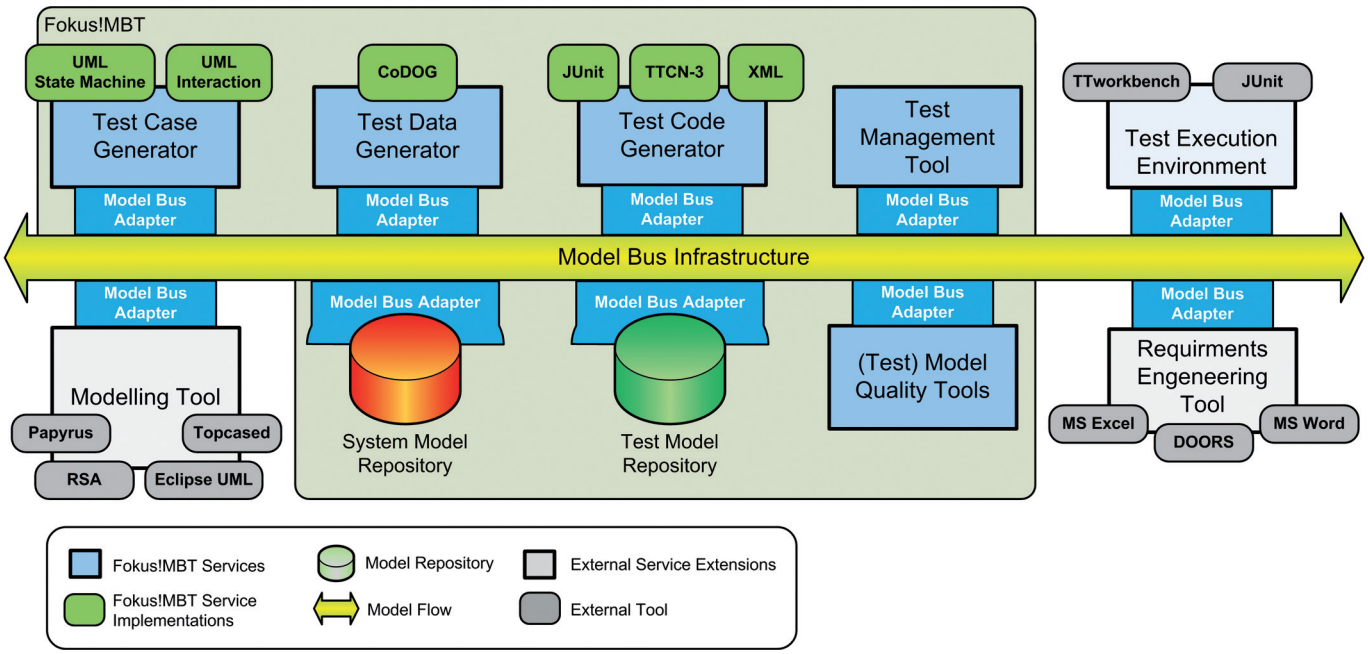

Abbildung 3: Architektur der Fokus!MBT-Werkzeugkette

domänenspezifischen Testwerkzeugkette zusammengesetzt und orchestriert werden. Abbildung 3 veranschaulicht die zugrunde liegende Architektur, die auf dem ModelBus [ModelBus] aufbaut.

Die *ModelBus*-Infrastruktur ist eine service-orientierte *Middleware*, die die Integration heterogener Datenmodelle und verschiedener Werkzeuge ermöglicht. Zu diesem Zweck werden die Werkzeuge über Adapter an den *ModelBus* angeschlossen. Das Mapping von proprietärem zu gewünschtem Datenmodell wird innerhalb der als *Web Service* implementierten Adapter realisiert. So ist es beispielsweise möglich, dass jeder Entwickler mit dem Modellierungswerkzeug seiner Wahl arbeitet. Die Konsistenz und Synchronisierung des Modells über die Werkzeuggrenzen hinweg sichert der *ModelBus* ab.

Instanzen des TestingMM bilden das kanonische Datenmodell von Fokus!MBT und werden zwischen den adaptierten Werkzeugen ausgetauscht. Für den Datenfluss innerhalb der Werkzeugkette erfolgt ein Mapping aus dem TestingMM in ein bestimmtes Datenmodell und umgekehrt. Die einzelnen Werkzeuge reichern das TestingMM durch ihre domänenspezifischen Services mit Informationen an. Die Wahl der integrierbaren Werkzeuge wird lediglich durch ihre technische Integrier- und Adaptierbarkeit begrenzt. Aktuell ist Fokus!MBT in der Lage, UML- und SysML-Modelle des *IBM Rational Software Architects* (RSA), von *Papyrus* und *Topcased* zu verarbeiten und in das TestingMM zu konvertieren. Die Art der Konvertierung richtet sich wiederum nach der Methodik, welche für die Durchführung des modellbasierten Testszenarios gewählt wurde.

In [Pretschner] werden gängige Methoden für die Ableitung beziehungsweise Nutzung von Testmodellen identifiziert. Fokus!MBT fokussiert und realisiert vor allem die automatisierte Ableitung von Testmodellen aus Systemmodellen sowie die manuelle Erstellung von Testmodellen (siehe Abbildung 4). Der aktuelle Schwerpunkt liegt auf der Realisierung dieser Methoden auf Basis von UML- und SysML-Modellen. Testspezifische Informationen werden mittels UTP-Stereotypen in das UML-/SysML-Modell eingebracht. Dazu wurde das UTP um die zusätzlichen Konzepte des TestingMM (Abbildung 2) ergänzt. Dies ermöglicht die Nutzung der

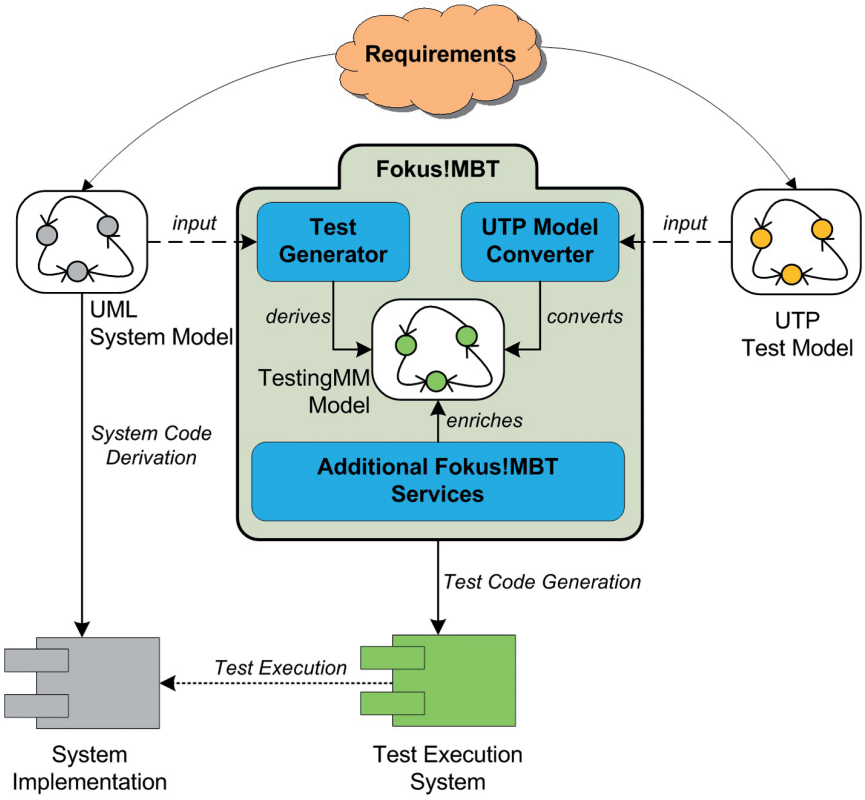

Abbildung 4: Fokus!MBT im Kontext manueller und automatisierter Ableitung des Testmodells

in Abbildung 2 aufgeführten Konzepte für die Erstellung beziehungsweise Generierung UML-basierter Testmodelle.

Dabei bietet Fokus!MBT selbst kein eigenes Modellierungswerkzeug an. Hintergrund dieser Entscheidung ist, dass Fokus!MBT nicht von einer bestimmten Modellierungsnotation abhängig sein soll. Essentiell ist lediglich, dass aus der verwendeten, möglicherweise proprietären Modellierungsnotation in das TestingMM konvertiert werden kann. Dadurch ist Fokus!MBT flexibel genug, um in bestehende Prozesse integriert zu werden.

Da insbesondere die automatisierte Ableitung von (Teilen von) Testmodellen aus Systemmodellen von großem Interesse und erhöhter Komplixtät ist, wird nachfolgend ein solches Verfahren beschrieben, wie es aktuell von Fokus!MBT realisiert wird.

Fokus!MBT @ Work

Um System- und Testartefakte weitestgehend getrennt zu halten, wird im Systemmodell ein neues Paket angelegt, das alle testspezifischen, für die Ableitung relevanten Informationen beinhaltet. In dem Testpaket werden verschiedene Testkontexte angelegt, die jeweils unterschiedlichen Testkonfigurationen entsprechen. Die Attribute eines Testkontexts werden mit «TestComponent» oder «SUT» markiert. Als Typen dienen die Klassifizierer (etwa Klasse, Komponente) des Systemmodells. Auf diese Weise wird das Systemmodell nicht mit testspezifischen Informationen überladen. Das separate Testpaket kann jederzeit entfernt werden, ohne die Konsistenz des Systemmodells negativ zu beeinflussen. Das zuvor angelegte Testpaket dient als Einstiegspunkt für die Fokus!MBT-Werkzeugkette. Sukzessive werden die angelegten Testkontexte verarbeitet und in Instanzen der *TestingMM::TestContext* Metaklasse transformiert. Die UTP-Stereotypen fungieren dabei als Testdirektiven und steuern die automatisierte Testableitung. In [Dai] wurde ein proprietäres Testdirektiven-Metamodell vorgestellt, welches die Erstellung von UTP-Testmodellen aus UML-Systemmodellen ermöglicht. Nachteilig bei diesem Verfahren ist, dass die Testdirektiven außerhalb des Systemmodells zu warten sind, während die Verwendung des UTP von jedem UML-kompatiblen Werkzeug unterstützt wird. Zudem hat die Autorin sich ausschließlich auf die Wiederverwendung von Sequenz-Diagrammen des

Systemmodells konzentriert. Diese wurden gemäß der Testdirektiven strukturell umgewandelt. Eine Ableitung von Testfällen aus Zustandsmaschinen auf Basis struktureller Abdeckungskriterien wurde nicht diskutiert. Fokus!MBT schließt diese Lücken, indem es neben Sequenz-Diagrammen vor allem auf die Testableitung aus Zustandsmaschinen setzt [Stefanescu].

UML kennt das Konzept des *ClassifierBehavior*, wodurch der Lebenszyklus eines Klassifizierers durch eine entsprechende Verhaltensbeschreibung spezifiziert wird. Der von uns implementierte Testfallgenerator nutzt dieses Konzept, indem er die ClassifierBehavior-Beschreibung (als Zustandsmaschine) des als <<SUT>> markierten Klassifizierers für die Ableitung heranzieht. Die Ableitung selbst wird auf Basis von strukurellen Abdeckungskriterien durchgeführt. Dadurch wird es beispielsweise möglich, alle oder nur einen bestimmten Prozentsatz der Zustände, respektive Transitionen einer Zustandsmaschine mindestens einmal zu durchlaufen.

Damit eine Transition feuern kann, ist es zwingend erforderlich, dass etwaige *Guard*-Bedingungen der Transition erfüllt sind. Um invalide Pfade a priori auszuschließen, simuliert Fokus!MBT den umgebenden Klassifizierer der Zustandsmaschine und führt sämtliche Manipulationen an dessen Attributen durch. Um eine bestimmte Transition zu feuern, analysiert Fokus!MBT die Bedingungen, die für die zu feuernde Transition gelten und übergibt diese dem integrierten Testdatengenerator. Anschließend werden potenzielle Transitionsaktionen durchgeführt, beispielsweise die Zuweisung der generierten Daten zu einem bestimmten Attribut oder das Senden eines Signals an kollaborierende Klassifizierer.

All diese Schritte werden in einer an das *UML Common Behavior Domain Model* angelehnten Form aufgezeichnet und in einer Ereignisliste (*Event-Traces*) gespeichert. Dementsprechend resultiert aus jedem Pfad durch die Zustandsmaschine eine eigenständige Ereignisliste, welche abschließend in TestingMM-Testfälle überführt werden. Ist das gewünschte Abdeckungsmaß erreicht, wird die Testfallerstellung abgeschlossen und der entsprechende Testkontext im TestingMM finalisiert. *Model-to-Text*-Transformatoren erlauben daraufhin die Generierung ausführbaren Testkodes. Standardmäßig wird ein TTCN-3-Testskript-Generator mitgeliefert, andere Transformatoren lassen sich problemlos integrieren. Eine weitere zentrale Rolle in der Ableitung von Testfällen spielt die Testdatengenerierung, welche nachfolgend beschrieben wird.

Constraint-basierte Testdatengenerierung

Fokus!MBT verfügt über einen Constraint-basierten Testdaten- und Testorakelgenerator (kurz: CoDOG [Wendland]), welcher zu einer gegebenen Problembeschreibung valide Daten generiert. Dabei bedeutet *Constraint-basiert* zweierlei: zum einen werden die Testdaten im Modell nur indirekt in Form von *Constraints* beschrieben, zum anderen basiert CoDOG intern auf dem *Constraint-Programming*-Paradigma. Zur Auflösung der Problembeschreibungen kapselt CoDOG einen *Constraint Solver*.

Die Fokus!MBT-Werkzeuge kommunizieren mit CoDOG über ein weiteres proprietäres, domänenspezifisches Metamodell, dem *Constraint Satisfactory Model* (CSM). Das CSM erlaubt die abstrakte Beschreibung von *Constraint*-Problemen, welche in das geforderte Format des *Constraint Solvers* transformiert wird. Sollte zukünftig ein anderer *Constraint Solver* eingesetzt werden, ist lediglich das Mapping von CSM zum Eingabeformat des neuen *Constraint Solvers* anzupassen. Die Fokus!MBT-Werkzeuge arbeiten hingegen ausschließlich auf dem CSM.

Desweiteren bietet CoDOG eine generische *Parser*-Komponente an. Dadurch wird es möglich, jede gewünschte Sprache (möglicherweise komplementär) zu verwenden, um das Modell mit *Constraints* anzureichern. Bedingung hierfür ist die Realisierung eines entsprechenden *Parsers*. Jedes *Constraint* wird von seinem dedizierten *Parser* in das korrelierende CSM-Konstrukt übertragen, welches abschließend an CoDOG zur Datengenerierung übermittelt wird. Aktuell liegen *Parser* für OCL und einer limitierten Untermenge von Java-Ausdrücken vor, sowie für reguläre Ausdrücke für die Definition valider Zeichenketten.

Aussagen zur Anforderungsabdeckung

Ein primäres Ziel des Testens ist die Abschätzung der Qualität der getesteten Software. Dies beinhaltet die Validierung und Verifikation der erhobenen Anforderungen an das System. Da Testen immer in Hinblick auf zu prüfende Anforderungen durchgeführt wird, ist es unerlässlich, eindeutige Aussagen sowohl über die anforderungsspezifische Testabdeckung als auch über den Status der Anforderungen zu treffen. Dies ist Teil der Ergebnisanalyse und fließt unter anderem in den Testreport ein, nach welchem entschieden wird, ob das Produkt zur Auslieferung freigegeben oder einem Korrekturlauf unterzogen wird. Das TestingMM definiert im Paket *TestPurpose* die dazu notwendigen Konzepte.

Fokus!MBT unterscheidet zwischen drei Arten von Testfallausführungsbewertungen (*Verdict*), dem Testfall-*Verdict*, dem lokalen Anforderungs-*Verdict* und dem globalen Anforderungs-*Verdict*. Das Testfall-*Verdict* repräsentiert den Ausführungsstatus des Testfalls, das heißt mit welchem Ergebnis der Testfall abgeschlossen wurde. Aus TTCN-3 hat das TestingMM die *Verdicts* none, pass, inconclusive, error und fail übernommen. Die Anforderungs-*Verdicts* werden aus dem Testfall-*Verdict* abgeleitet, allerdings existieren hier lediglich drei verschiedene *Urteile*, nämlich pass, fail und not executed. Die Ableitung des lokalen Anforderungs-*Verdict* erfolgt nach folgender Strategie: ein Testfall-*Verdict* pass produziert ein ebensolches lokales Anforderungs-*Verdict*. Alle anderen Testfall-*Verdicts* werden auf das Anforderungs-*Verdict* fail abgebildet, und not executed weist darauf hin, dass ein assoziierter Testfall der Anforderung bislang noch nicht ausgeführt wurde.

Da zwischen Testfällen und Anforderungen logisch eine n:m-Beziehung besteht – jede Anforderung kann von mehreren Testfällen verifiziert werden, gleichsam kann ein Testfall eine unbestimmte Menge von Anforderungen, mindestens jedoch eine, verifizieren – ist ein Gesamturteil aus den lokalen Anforderungs-*Verdicts* zu ermitteln. Dieses *Verdict* bezeichnen wir als globales Anforderungs-*Verdict*. Sobald für eine Anforderung das lokale Anforderungs-*Verdict* fail berechnet wird, ist die Verifikation der gesamten Anforderung fehlgeschlagen (fail). Wurde jedes lokale Anforderungs-*Verdict* erfolgreich bewertet (pass), gilt die gesamte Anforderung als verifiziert und erhält das Verdict (pas)s. Wurden die Testfälle einer Anforderung noch nicht ausgeführt, ist das Gesamturteil für diese Anforderung not

Requirement ID	Test Case	Verdicts		
		Test Case Verdict	Local Req. Verdict	Global Req. Verdict
Req. 001	1	Fail	Fail	Fail
Req. 002	1	Fail	Fail	Fail
	2	Pass	Pass	
	3	Inconclusive	Fail	
Req. 003	4	Pass	Pass	Pass
	5	Pass	Pass	
Req. 004	5	Pass	Pass	Partial Pass
	6	-	Not Executed	

Tabelle 1: Beispielhafte Ableitung der Anforderungs-Verdicts

executed. Zusätzlich zu diesen *Verdicts* kennt das TestingMM auf globaler Anforderungsebene noch das *Verdict* partial_pass. Dieses neue *Verdict* besagt, dass der bisherige Status der Anforderung auf Basis aller durchgeführten Testfälle pass lautet, es allerdings noch nicht ausgeführte Testfälle gibt, die diesen Status noch in ein fail abwandeln könnten. Dadurch wird verhindert, dass Testfälle zu einer Anforderung vergessen werden. Auf ein *Verdict* partial_fail wurde bewusst verzichtet, da ein fail per Definition nicht mehr zu einem pass führen kann. Die Information, dass mindestens ein Testfall einer Anforderung fehlschlug, aber noch nicht alle Testfälle ausgeführt wurden, bringt daher aus Testmanagement-Sicht keinen Mehrwert. Tabelle 1 veranschaulicht das Prinzip der *Verdicts* auf unterschiedlichen Ebenen.

Referenzen

[Utting] Utting, M.; Pretschner, A., Legeard, B.: A Taxonomy of Model-Based Testing. ISSN 1170-487X, 2006. URL: http://www.cs.waikato.ac.nz/pubs/wp/2006/uow-cs-wp-2006-04.pdf.

[Pretschner] Pretschner, A.; Phillips, J.: Methodological Issues in Model-based Testing. In: *Model-based Testing of Reactive Systems*. LNCS 3472, pp. 281 – 291, Springer Berlin /Heidelberg, 2004. ISBN: 978-3-540-26278-7

[UTP] Object Management Group (OMG): UML2 Testing Profile, Version 1.0. URL: http://www.omg.org/cgi-bin/doc?formal/05-07-07

[UML] Object Management Group (OMG): OMG Unified Modeling Language (OMG UML) Superstructure, Version 2.3. URL: *http://www.omg.org/spec/UML/2.3/*

[TPTP] Eclipse Foundation: Test & Performance Tools Platform (TPTP). URL: http://www.eclipse.org/tptp

[TTCN-3] European Telecommunications Standards Institute (ETSI): The Testing and Test Control Notation version 3 (TTCN-3). URL: http://www.ttcn-3.org

[Dai] Dai, Z.R.: An Approach to Model-Driven Testing – Functional and Real-Time Testing with UML 2.0, U2TP and TTCN-3. PhD thesis, TU Berlin (June 2006)

[Stefanescu] Stefanescu, A.; Wendland, M.-F.; Wieczorek, S.: Using the UML testing profile for enterprise service choreographies. 36th Euromicro Conference on Software Engineering and Advanced Applications, (SEAA 2010), Lille, France, September 2010.

[Modelplex] Modelplex Project. URL: http://www.modelplex.org

[EMF] Eclipse Modeling Framework. URL: http://www.eclipse.org/emf

[ModelBus] ModelBus Infrastructure, URL: http://www.modelbus.org.

[VERDE] VERDE Project. URL: http://www.itea-verde.org/

[UTP_RTF] Object Management Group (OMG): OMG UML Testing Profile 1.1 (UTP) RTF. URL: http://www.omg.org/techprocess/meetings/schedule/UML_Testing_Profile_1.1_%28UTP%29_RTF.html.

[Wendland] Wendland, M.-F.: Testdaten- und Testorakelgenerierung aus OCL-Ausdrücken im Kontext modellbasierten Testens. Masterthesis, Beuth Hochschule für Technik Berlin, 2009.

Ausblick

Die Diskussionen um das TestingMM sowie offene Punkte und Ungereimtheiten im UTP waren der Auslöser für die Bildung einer *UTP Revision Task Force* bei der OMG im Juni 2010. Fraunhofer FOKUS übernimmt bei der Revision des UTP, wie auch schon bei der initialen Spezifikation, den Vorsitz eines namhaften, internationalen Komitees. Das wichtigste Ziel der Revision ist vor allem die Angleichung des UTP an die aktuelle UML-Version 2.3. Die Revision wurde offiziell am 25. Juni 2010 gestartet, der Aufruf zu Kommentaren läuft noch bis Ende Januar 2011. Über die Mailadresse issues@omg.org können Vorschläge eingebracht und Probleme gemeldet werden, wozu wir Sie herzlich einladen. Weiterführende Informationen finden Sie auf [UTP_RTF].

Die Arbeiten in der UTP RTF werden unter anderem auf den Ergebnissen des TestingMM und der Fokus!MBT Werkzeugkette aufbauen. In unserem Test-Metamodell und der Fokus!MBT-Werkzeugkette haben wir insbesondere auf die Flexibilität und Offenheit gegenüber verschiedenen Testprozessen, Testartefakten, Notationen und Werkzeugen geachtet. Die dabei umgesetzten Prinzipien zur Anpassbarkeit und Einbettung in bestehende Umgebungen haben sich bereits in Projekten mit THALES, SAP, Telefonica und Deutsche Telekom bewährt, für die MBT-Lösungen für verschiedene Anwendungsgebiete in verschiedenen Prozess- und Werkzeuglandschaften realisiert wurden.

Wir sind davon überzeugt, dass eine internationale Einigung auf die gemeinsamen, allgemeingültigen Konzepte modellbasierter Testprozesse und der dabei genutzten Artefakte einen weiteren Schritt in der Effektivierung und Effizienzsteigerung von Testprozessen ergeben wird – vergleichbar zur Allgemeingültigkeit und Übertragbarkeit der generischen Testausführungskonzepte in TTCN-3. Auch wenn das TestingMM und UTP sich derzeit auf die ausschließlich testbezogenen Artefakte konzentrieren, ist perspektivisch zu untersuchen, inwiefern die Modellierung der Testprozesse und ihre Steuerung (und Optimierung) mittels der Testgenerierungs- oder Testausführungsdirektiven eine zusätzliche Abstraktion, weitere Automatisierung und damit eine noch bessere Konzentration auf den eigentlichen Testentwurf und die Bewertung der Testergebnisse erlauben. ■