# DAS UML TESTING PROFILE
## Erweitertes Abstrakt

Ina Schieferdecker, Fraunhofer FOKUS, Berlin, Germany,
schieferdecker@fokus.fraunhofer.de
www.fokus.fraunhofer.de/tip

## 1 Einleitung

Obwohl Testen oftmals bis zu 50% des gesamten Entwicklungsaufwands erfordert, ist es bislang nicht bzw. wenig in die einzelnen Systementwicklungsphasen integriert. Das liegt u.a. daran, dass Architekten, Entwickler und Tester verschiedene Sprachen und Werkzeuge nutzen, so dass die Interaktion untereinander und der Austausch von Informationen und Dokumenten erschwert werden. Das UML 2.0[1] Testing Profile[2] (U2TP [1][2]) schließt diese Lücke, indem Konzepte und Wege aufgezeigt werden, wie UML auch für den Testentwurf und Testspezifikationen eingesetzt werden kann. Das ermöglicht die Wiederverwendung von Systemdokumenten für die Testentwicklung und erlaubt so eine frühzeitige und kontinuierliche Verzahnung von Systementwicklungs- und Testschritten. U2TP ist die Grundlage für modell-basiertes Testen im Kontext von UML.

Die UML (Unified Modelling Language) der OMG (Object Management Group) ist eine Familie graphischer Modellierungssprachen für komplexe, objekt-orientierte Systeme, die über ein gemeinsames Metamodell verbunden sind. Während UML sehr mächtig bzgl. objekt-orientierter Konzepte von Softwaresystemen ist, waren bislang Testkonzepte jenseits der Betrachtung. Die Entwicklung von U2TP wurde daher Ende 2001 mit einem RFP (Request for Proposal) angestoßen und führte im April 2003 zur Annahme der finalen U2TP Spezifikation des U2TP Konsortiums (mit IBM, Rational, Telelogic, Softeam, Ericsson, Motorola, Universität Lübeck, geleitet von Fraunhofer FOKUS). In der jetzigen Finalisierungsphase von U2TP werden offene Punkte und Probleme geklärt, die im Zuge der Implementierung von U2TP Werkzeugen durch IBM/Rational, Telelogic, Softeam und Microsoft auftreten.

Dieser Abstrakt beschreibt die wesentlichen Konzepte von U2TP. Auf die Präsentation eines Beispiels wird verzichtet und dazu auf [1][3] verwiesen.

## 2 Die Konzepte des UML Testing Profiles

U2TP bietet Konzepte zum Entwurf und der praktischen Entwicklung von präzisen Spezifikationen für Black-Box-Tests[3]. U2TP bietet im wesentlichen Konzepte zur Beschreibung von Testarchitekturen, Testverhalten und Testdaten. Gemeinsam mit der UML definieren diese Konzepte eine Testmodellierungssprache, mit der die Artefakte von Testsystemen spezifiziert, visualisiert, analysiert, konstruiert und dokumentiert werden können. U2TP nutzt wo immer möglich existierende UML 2.0 Konzepte, so dass eine minimale Menge von zusätzlichen Konzepten definiert wurde. Diese Menge wurden aus den Konzepten des Softwaretestens (wie TET [10] und JUnit [11]) und des Protokolltestens (wie TTCN-3 [7][8]) hergeleitet.

Mit den Testarchitekturkonzepten werden die strukturellen Aspekte einer Test Suite festgelegt (siehe Abb. 1[4]):

- Mit dem Stereotyp *SUT* (System Under Test) werden ein oder mehrere Objekte als Bestandteil des zu testenden Systems und als wichtig für eine Testkonfiguration identifiziert.

---

[1] UML 2.0 [4][5] ist die neue Version von UML mit einer engeren Integration der Sprachfamilie und einer dedizierten Semantik.

[2] UML Profile sind Erweiterungen von UML um domänenspezifische Konzepte (definiert als Stereotypen, Attribute und Relationen), die entsprechend der UML Profil-Mechanismen definiert werden und sich nahtlos in die UML einbetten. Im Fall von U2TP wird die UML um testspezifische Konzepte erweitert.

[3] Beim Black-Box Testen wird das zu testende System über seine öffentlichen Schnittstellen stimuliert und beobachtet, wobei kein Wissen über die interne Realisierung des Systems genutzt und nur das Schnittstellenverhalten und dessen Leistungsfähigkeit bewertet werden.

[4] Abb. 1-3 sind vereinfachte Ausschnitte aus dem U2TP Metamodell und zeigen die wesentlichen Stereotypen, ihrer Relationen und Ableitung aus UML 2.0.

- Objekte des Testsystems werden über *Testkomponenten*-Klassen mit ihren Schnittstellen, Attributen und Operationen definiert.

- Objekte der Testkomponenten-Klassen bilden zusammen mit den SUT Objekten *Testkonfigurationen*, wobei Verbindungen zwischen den Schnittstellen der Testkomponenten und der SUT die Interaktion[5] zwischen Testsystem und zu testendem System ermöglichen.

- Testkonfigurationen charakterisieren zusammen mit den Testfällen eine *Test Suite*. Dabei ist die Testkonfiguration die interne Struktur der Test Suite, die für jeden Testfall der Test Suite genutzt wird. Die Testfälle sind Operationen der Test Suite und werden durch die in der Testkonfiguration enthaltenen Testkomponenten realisiert.

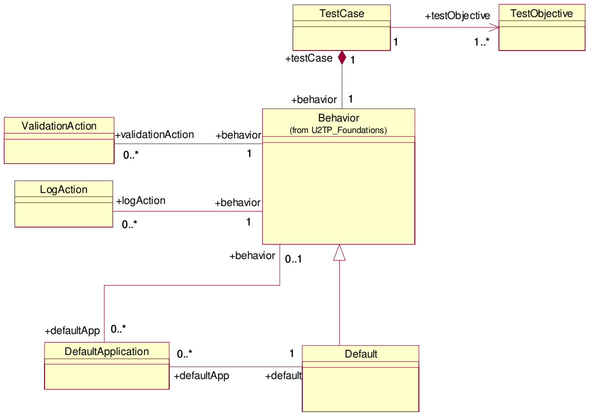
TestCase
+testObjective
TestObjective
1
1..*
+testCase
1
+behavior
1
ValidationAction
+validationAction
+behavior
Behavior
(from U2TP_Foundations)
0..*
1
LogAction
+logAction
+behavior
0..*
1
+behavior
0..1
+defaultApp
0..*
DefaultApplication
0..*
1
Default
+defaultApp
+default

**Abb. 1**: Architekturelle U2TP Konzepte

- Testkomponenten liefern während der Testausführung ihre lokalen *Testergebnisse* (sogenannte Verdicts, die die klassischen Werte pass, fail oder inconclusive[6] annehmen können).

- Zur Auswertung der individuellen Testergebnisse jeder Testkomponente wird eine spezielle Komponente im Test System – der *Arbiter* – genutzt. Der Arbiter ist in U2TP aus technischen Gründen als vordefinierte Schnittstelle realisiert. Der Arbiter berechnet aus den Einzelergebnissen das Gesamtergebnis für einen Testfall. U2TP bietet einen vordefinierten funktionalen Arbiter, bei dem schlechte Testergebnisse Priorität erhalten: wenn eine Testkomponente ein fail zuweist, kann das Gesamtergebnis nicht besser als fail sein. Andere Arbiter wie z.B. für Echtzeit-, Leistungs- oder Stabilitätstests können durch den Nutzer definiert oder durch Werkzeuge bereitgestellt werden.

Das Verhalten für Test Suites, Testfälle und Testkomponenten wird im wesentlichen unter Nutzung der verhaltensorientierten UML Diagramme für Zustandsautomaten, Interaktionen und Aktivitäten definiert (siehe Abb. 2). Diese können genutzt werden, um Stimuli an die und Beobachtungen von der SUT zu definieren, um die Koordination und Synchronisation zwischen Testkomponenten oder aber die Reihenfolge der Testausführung festzulegen.

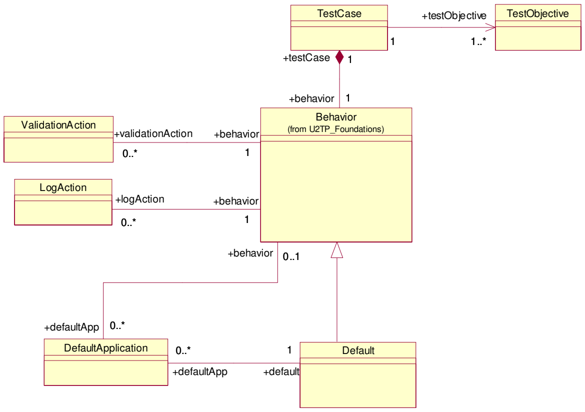
TestCase
+testObjective
TestObjective
1
1..*
+testCase
1
+behavior
1
ValidationAction
+validationAction
+behavior
Behavior
(from U2TP_Foundations)
0..*
1
LogAction
+logAction
+behavior
0..*
1
+behavior
0..1
+defaultApp
0..*
DefaultApplication
0..*
1
Default
+defaultApp
+default

**Abb. 2**: U2TP Konzepte für Testverhalten

Da es beim Testen aber i. allg. nicht nur auf die Beschreibung des „normalen“ erwarteten Verhaltens der SUT ankommt, sondern insbesondere auch auf die Bewertung unerwarteten Verhaltens und die Beachtung von wesentlichen, aber nicht zum Test beitragenden Verhalten der SUT ankommt, wurde mit den *Defaults* eine Erweiterung der Verhaltensdiagramme vorgenommen. Defaults dienen zur Spezifikation des Testverhaltens für alle, durch den Test nicht vordergründig behandelten Verhalten der SUT und dienen damit zur Komplettierung einer Testdefinition. Defaults können hierarchisch definiert werden und an einzelne Ereignisse, Verhaltensabläufe, Testkomponenten oder aber ganze Test Suiten gehängt werden. Sie erlauben die Auslagerung von „Nebenverhalten“ und die Konzentration auf das Wesentliche in der eigentlichen Testdefinition. Defaults verbessern die Lesbarkeit der Tests und vereinfachen deren Validierung, doch wo die Grenze zwischen Haupt-

---

[5] Die Verbindungen zur Kommunikation zwischen SUT und Testsystem können mit Kodierungsregeln charakterisiert werden, so dass alle auf einer Verbindung übertragenen Daten entsprechend dieser Regel kodiert und dekodiert werden müssen.

[6] Ein weiteres mögliches Testergebnis error weist auf einen Fehler im Testsystem hin und kennzeichnet nicht die SUT.

und Nebenverhalten gezogen – also welche Defaults wo genutzt – werden, entscheidet der Testentwickler.

Neben den Defaults führt U2TP noch weitere Konzepte für das Testverhalten ein:

- *Testzwecke* (Test Objectives genannt) können als Text, Use Case oder eines der anderen UML Diagramme definiert und an einen Testfall oder eine Test Suite gebunden werden, um das Ziel und die Intention eines Tests zu beschreiben.
- *Validierungsaktionen*, die von einer Testkomponente ausgeführt werden, um an den Arbiter lokale Testergebnisse zu übertragen.
- *Log-Aktionen*, die zur Ablage von zusätzlichen Informationen über die Testausführung, die ggfs. für eine spätere Analyse genutzt werden sollen
- Eine *Ende-Aktion* zur Kennzeichnung des Endes eines Testfalls auf einer Testkomponente, ohne dabei jedoch die Testkomponente zu terminieren. Damit kann in U2TP die sequentielle Abarbeitung von Testfällen auf einer einmal erzeugten (und u.U. während der Testausführung veränderten) Testkonfiguration gesteuert werden.

Ein weiterer wichtiger Aspekt in der Testspezifikation ist der der Testdaten (siehe Abb. 3), die meistens nicht statisch fixiert, sondern flexibel definierbar sein müssen. U2TP unterstützt dabei zwei Ansätze:

- Die Definition von *Data Pools*, die u.a. als Abbildung für Äquivalenzklassen in den Testdaten genutzt werden.
- Die Definition von *Wildcards*, die beispielsweise für die Beschreibung einer Bandbreite von unerwarteten Reaktionen der SUT genutzt werden können. Zusätzlich zu dem omit-wildcard von UML definiert U2TP ein any-wildcard und ein any_or_omit-wildcard.

Zur Komplettierung der Konzepte zur Testmodellierung hat U2TP auch zusätzliche Konzepte zur Zeitmodellierung und zeitlichen Steuerung von Tests eingeführt, die so in UML 2.0 nicht definiert sind:

- *Timer*, die zur zeitgesteuerten Beeinflussung und Kontrolle von Testverhalten u.a. auch zur Terminierung von Testfällen genutzt werden können
- *Zeitzonen*, die zur zeitlichen (und damit konzeptionell räumlichen) Gruppierung von Testkomponenten dienen, so dass zeitliche Abläufe von Testkomponenten in derselben Zeitzone verglichen werden können.

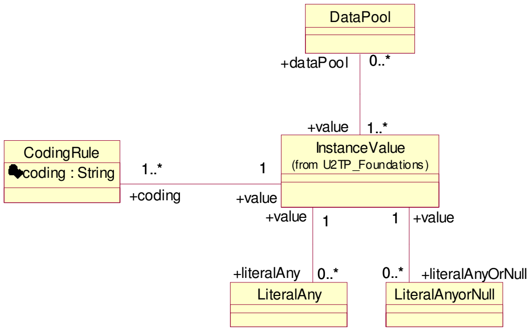

**Abb. 3**: U2TP Konzepte für Testdaten

## 3 Die Einbettung in UML

Das UML 2.0 Testing Profile ist in Form zweier Metamodelle definiert:

- Das *UML 2.0-basierte Metamodell* ist die eigentliche U2TP Definition. Es ist als Spezialisierung des UML 2.0 Metamodells realisiert und dient insbesondere UML Werkzeugen zur Umsetzung von U2TP mittels verfeinerter, in UML eingebetteter Konzepte.
- Das selbständige *MOF(Meta Object Facility [6])-Metamodell* ist eine alternative, weniger mächtige Definition von U2TP und dient der Umsetzung von U2TP unter Umgehung einer UML 2.0 Basis. Dies ist insbesondere in der Übergangsphase von UML 1.x nach UML 2.0 sinnvoll, da es bis dato keine bzw. nur eingeschränkte UML 2.0 Werkzeuge gibt. Mit dem MOF Metamodell wird die Umsetzung von Repositories zur Ablage und Manipulation von U2TP Artefakten ermöglicht.

Sogenannte *Compliance Points* definieren die Anforderungen an reguläre U2TP Werkzeuge: es kann eines der beiden Metamodelle genutzt, im Falle des UML 2.0 basierten Metamodells muss die von UML 2.0 und U2TP definierte Syntax umgesetzt und in beiden Fällen müssen die Datenstrukturen und die Semantik der Konzepte korrekt realisiert werden. Es gibt dabei Abstufungen, auf die jedoch hier nicht weiter eingegangen wird.

Derzeit werden Werkzeuge für U2TP unter Nutzung beider Metamodelle realisiert: beispielsweise entwickelt Telelogic im Rahmen ihrer Tau G2 Werkzeuge ein U2TP Werkzeug unter Nutzung des UML 2.0 basierten Metamodells, während IBM/Rational im Kontext des

Eclipse Hyades Projekts das MOF-Metamodell als Ausgangspunkt nutzt.

## 4 Die Abbildungen auf Testinfrastrukturen

Zur direkten Umsetzung der abstrakten U2TP Spezifikationen als ausführbare Tests bietet U2TP zwei Abbildungen auf Testinfrastrukturen an:

- JUnit [11] ist eine Open Source Testinfrastruktur für Unit-Tests und wird vielfach von Programmierern zur Analyse von Java Code eingesetzt. Wegen der Fokussierung auf Unit-Tests bietet JUnit nicht alle von U2TP definierten Konzepte zur Testmodellierung an, so dass nur eine eingeschränkte Abbildung möglich ist.

- Die Testing and Test Control Notation TTCN-3 [7][8][9] ist eine Testspezifikations- und –implementierungstechnologie, die zum Testen reaktiver, möglicherweise verteilter Systeme auf verschiedenen Ebenen wie Modul-, Komponenten- und Systemebene entwickelt wurde. Obwohl sich TTCN-3 und U2TP in einigen Grundkonzepten unterscheiden, ist eine vollständige Abbildung von U2TP auf TTCN-3 möglich.

Mit den für U2TP definierten Abbildungen auf Testinfrastrukturen werden U2TP Testspezifikationen direkt ausführbar (unter der Annahme der Existenz entsprechender Werkzeuge). Dabei sind diese Abbildungen als repräsentative, aber beispielhafte Abbildungen zu verstehen: es können sowohl andere Zieltechnologien genutzt werden – zum Beispiel eine direkte Realisierung in einer Programmiersprache wie Java – als auch können die definierten Abbildungen modifiziert und angepasst werden. Sie stellen eine mögliche Abbildung unter Weglassung weiterer Alternativen dar.

## 5 Ausblick

Dieser erweiterte Abstrakt bietet einen kurzen Überblick zum UML 2.0 Testing Profile, welches UML um testspezifische Konzepte wie Test Suite, Testfall, Testkomponente, Testergebnis etc. erweitert, wobei wo immer möglich UML 2.0 direkt genutzt wird. Zur Definition von Testverhalten können die UML 2.0 Diagramme für Zustandsmaschinen, Interaktionen und Aktivitäten genutzt werden, die u.a. um testspezifische Aktionen wie das Weiterleiten eines Testergebnisses an den Arbiter erweitert wurden. Nach einem Überblick zu den U2TP Konzepten wurden die Definition der U2TP in Form von Metamodellen und die Abbildung von U2TP Testspezifikationen auf JUnit- und TTCN-3-Tests angerissen. Die Konzepte von U2TP sind so definiert, dass eine direkte (wenn auch im Fall von JUnit eingeschränkte) Abbildung möglich ist.

Zusammenfassend bietet U2TP eine kohärente Menge von dedizierten Testkonzepten als Erweiterung der UML an. Diese Testkonzepte werden bereits seit langem im Rahmen von Software- und Protokolltests genutzt. Jedoch steht ein abschließender Nachweis der Praktikabilität der gewählten Konzeptmenge aus. Dieser Nachweis wird im Zuge der Erstellung und Nutzung von U2TP Testwerkzeugen erwartet. In der derzeitigen Finalisierungsphase von U2TP sind Vorschläge und Kommentare jeder Form an u2tp@fokus.fraunhofer.de willkommen.

Des weiteren ist U2TP mit dem Anspruch der integrierten UML-basierten System- und Testentwicklung erarbeitet worden. Methoden, die die Abläufe in einem solchen integrierten Entwicklungsprozess, den Einsatz der UML und des U2TP erläutern und den Informationsfluss zwischen den Phasen und Artefakten festlegen, sind zu erarbeiten und in Fallstudien zu analysieren.

## 6 Literatur

[1] OMG ADTF: final submission by the U2TP partners to OMG RFP on a UML Testing Profile, ad/01-07-08: *UML 2.0 Testing Profile*, ad/03-03-26, 2003.

[2] *UML 2.0 testing profile home page*: http://www.fokus.fraunhofer.de/U2TP/

[3] I. Schieferdecker, Z. R. Dai, J. Grabowski, A. Rennoch: *The UML 2.0 Testing Profile and its Relation to TTCN-3*. IFIP 15th Intern. Conf. on Testing Communicating Systems - TestCom 2003, Cannes, France, May 2003.

[4] OMG ADTF: 2nd revised submission by the UML 2.0 partners to OMG RFP ad/00-09-02: *Unified Modelling Language: Superstructure*, version 2.0, ad/03-02-01, 2003.

[5] OMG ADTF: 3rd revised submission by the UML 2.0 partners to OMG RFP ad/00-09-01: *Unified Modelling Language: Infrastructure*, version 2.0, ad/03-01-01, 2003

[6] OMG: *Meta Object Facility*, Version 1.4, OMG document formal/00-04-0, 2000.

[7] ETSI: The Testing and Test Control Notation TTCN-3, Part 1: *TTCN-3 Core Language*, ETSI ES 201873-1 V2.2.1 (2002-10), also ITU-T Recommendation Z.140.

[8] ETSI: The Testing and Test Control Notation TTCN-3, Part 3: TTCN-*3 Graphical Presentation Format*, ETSI ES 201873-3 V2.2.1 (2002-10), also ITU-T Recommendation Z.142.

[9] J. Grabowski, D. Hogrefe, G. Rethy, I. Schieferdecker, A. Wiles, C. Willcock: *An Introduction into the Testing and Test Control Notation (TTCN-3)*. - Computer Networks Journal, Vol.42, Issue 3, 2003

[10] The Open Group: *Test Environment Toolkit*, TETware User Guide, 1999

[11] *JUnit*: http://www.junit.org.