TTCN-3 – Eine Sprache für die Spezifikation und Implementierung von Testfällen

TTCN-3 – A Language for the Specification and Implementation of Test Cases

Jens Grabowski und Michael Schmitt

Die *Testing and Test Control Notation* (TTCN-3) ist eine universelle Sprache für die Beschreibung von Tests für verteilte Systeme. Der vorliegende Artikel beschreibt den Hintergrund für die TTCN-3 Entwicklung und erläutert ausgewählte Sprachkonstrukte anhand von einfachen Beispielen.

The *Testing and Test Control Notation* (TTCN-3) is a universal language for the description of tests for distributed systems. This article describes the background for the TTCN-3 development and presents selected language features by means of simple examples.

1 Einleitung

Die *Testing and Test Control Notation* (TTCN-3) [2; 5] wurde von 1999 bis 2001 am *European Telecommunications Standards Institute* (ETSI) als Nachfolgesprache für die zweite Edition der *Tree and Tabular Combined Notation* (TTCN) entwickelt.

TTCN wurde ursprünglich als Teil des ISO-Standards 9646 *Open Systems Interconnection (OSI) – Conformance Testing Methodology and Framework* [7] speziell für die Spezifikation von Testfällen für das Konformitätstesten von OSI-Protokollimplementierungen standardisiert. Der Begriff *Konformitätstesten* bezeichnet dabei funktionales Black-Box-Testen basierend auf der Spezifikation des zu testenden Protokolls. Beim *funktionalen Black-Box-Testen* ist das zu testende System, namentlich das *System under Test* (SUT), eine Black-Box, die eine Funktion erbringen soll, deren innere Struktur nicht bekannt ist. Beim Test wird das SUT mit Eingaben stimuliert und die darauf folgenden Reaktionen beobachtet und beurteilt. Die Eingaben und Reaktionen werden im Allgemeinen *Testereignisse* genannt.

TTCN-3 unterstützt ebenfalls funktionales Black-Box-Testen, es handelt sich aber um eine allgemeine Testfall-Beschreibungssprache, die von Grund auf neu entwickelt wurde und keine anwendungsspezifischen Sprachelemente mehr besitzt.

TTCN-3 ist eine textuelle Beschreibungssprache, die in ihrem Aufbau traditionellen Programmiersprachen ähnelt. Mittels sogenannter *Präsentationsformate* können mit TTCN-3 spezifizierte Tests jedoch auch mit *Message Sequence Chart*-ähnlichen Diagrammen [4; 6] oder Tabellen [3] graphisch dargestellt werden.

TTCN-3 besitzt ein eigenes, vollständiges Datenkonzept mit allgemein bekannten Datentypen wie zum Beispiel **integer**, **boolean** oder **record**. Häufig ist es jedoch notwendig, die Datenbeschreibungen des SUT zu benutzen. TTCN-3 unterstützt daher die Verwendung der im Telekommunikationsbereich weit verbreiteten Datenbeschreibungssprache ASN.1 [8], und für das Testen von verteilten Systemen, die auf der Middleware CORBA [10] aufsetzen, wurde eine Abbildung von der Sprache IDL auf TTCN-3-Datenbeschreibungen definiert [1].

2 Die TTCN-3 – Sprache

In diesem Abschnitt werden einige ausgewählte Sprachelemente von TTCN-3 vorgestellt. Zur Illustration werden Beispiele aus einem TTCN-3–Dokument für den in [9] beschriebenen Produktionsprozess erläutert.

Als Testkonfiguration wird der in Bild 1 dargestellte Aufbau zugrunde gelegt. Das SUT besteht aus dem Roboter, den zugeordneten Greifplätzen und den Lagern für `C`- und `D`-Teile. Das Testsystem besteht aus insgesamt drei unabhängigen Prozessen, sogenanten *Testkomponenten* (siehe Abschnitt 2.3). Zwei Testkomponenten simulieren die Fa-

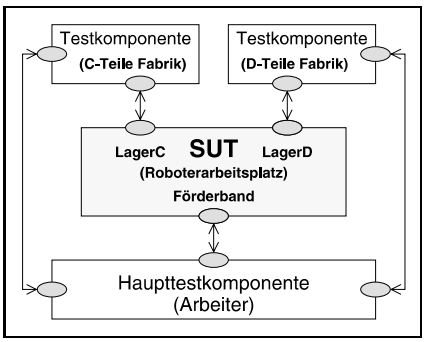

Bild 1: Testkonfiguration für das Produktionsbeispiel.

briken für C- und D-Teile, die die zugehörigen Lager auf Anfrage beliefern und einer Haupttestkomponente, die für die Testkoordination verantwortlich ist und die Rolle eines Arbeiters übernimmt, der das Förderband bestückt und wieder abräumt. Mit dieser Testkonfiguration lässt sich z. B. prüfen, ob der Roboter bei Eingabe eines A-Teils ein korrektes AC-Produkt und kein fehlerhaftes ACD-Produkt fertigt.

2.1 Module

Die oberste Strukturierungseinheit von TTCN-3 ist das *Modul*. Bild 2 zeigt ein einfaches TTCN-3–Modul mit dem Namen `Produktion`. Das Modul ist parameterisiert, um es in verschiedenen Umgebungen verwenden zu können. Ein Modul besteht aus einem *Definitionsteil* (Zeilen 2–14) und einem *Kontrollteil* (Zeilen 15–21).

Der Definitionsteil beinhaltet alle für die Ausführung einer Testreihe notwendigen Definitionen, z. B. Datentypen, Ports und Testkomponenten (Abschnitt 2.3), oder Testfälle (Abschnitt 2.4). Definitionen können auch aus anderen Modulen importiert werden. Im Definitionsteil des Moduls in Bild 2 werden der Datentyp `Basisteil` und die Konstanten `aTeil`, `bTeil`, `cTeil` und `dTeil` aus einem Modul namens `Basisteile` importiert, sowie der Datentyp `Bestellung` und die Schablone `standardBestellung` (Abschnitt 2.2) definiert.

```
 1: module Produktion(integer grosseLager) {
 2:   import from Basisteile {
 3:     type Basisteil;
 4:     const aTeil, bTeil, cTeil, dTeil;
 5:   }
 6:   type record Bestellung {
 7:     integer anzahlTeile,
 8:     integer lieferzeit optional;
 9:   }
10:   template Bestellung standardBestellung := {
11:     anzahlTeile := ?,
12:     lieferzeit := ( 10 .. infinity ) ifpresent
13:   }
14:   ... /* weitere Definitionen */
15:   control {
16:     var verdicttype gesamtUrteil := pass;
17:     gesamtUrteil := execute(acTest());
18:     if (gesamtUrteil == pass) {
19:       gesamtUrteil := execute(bdcTest())
20:     }
21:   }
22: }
```

Bild 2: TTCN-3–Modul `Produktion`.

Der Kontrollteil ist das Hauptprogramm eines TTCN-3–Moduls. Es beschreibt die Reihenfolge und Bedingungen für die Ausführung der im Definitionsteil definierten oder importierten Testfälle. Im Modul `Produktion` (Bild 2) wird der Testfall `bdcTest` nur dann ausgeführt, wenn der Testfall `acTest` erfolgreich war, d. h. der Testlauf mit dem Testurteil **pass** (siehe Abschnitt 2.7) beurteilt worden ist.

2.2 Schablonen

Für die Beschreibung von Testdaten stellt TTCN-3 das Konzept der *Schablonen* (engl. *templates*) zur Verfügung. Eine Schablone ist eine Datenbeschreibung, in der, neben konkreten Werten, auch spezielle *Matching*-Operatoren zugelassen sind. Mit ihrer Hilfe lassen sich Wertebereiche, beliebige Mengen von Werten, Komplemente von Wertemengen, optionale Werte, sowie Längenbeschränkungen für Zeichenketten und Felder spezifizieren. Während der Testausführung wird geprüft, ob die von den Testkomponenten empfangenen Daten mit der zugehörigen Schablone konform sind (engl. *matching*). In Bild 2, Zeilen 10–13, ist die Schablone `standardBestellung` für den Datentyp `Bestellung` (Zeilen 6–9), die eine mögliche Anfrage der SUT nach beliebig vielen Basisteilen und einer Lieferzeit von mindestens 10 Tagen (sofern angegeben) definiert.

2.3 Testkonfigurationen

In TTCN-3 werden Testfälle von parallel arbeitenden Testkomponenten ausgeführt. Im einfachsten Fall erfolgt die Ausführung auf einer einzelnen Testkomponente. Für den Test eines SUT, dessen Schnittstellen räumlich verteilt sind, wird ein verteilter Testfall benötigt, der von mehreren Testkomponenten ausgeführt wird.

Der Test des SUT und die zur Koordination von Testereignissen notwendige Kommunikation zwischen den Testkomponenten erfolgt über wohldefinierte Schnittstellen, die in TTCN-3 *Ports* heißen. Jede Testkomponente und das SUT besitzen lokale Ports. Die Kommunikationsverbindungen zwischen den Testkomponenten und dem SUT werden durch das Verbinden der lokalen Ports erzeugt.

Konzeptionell ist jeder Port durch eine unendliche Warteschlange mit einer *first-in-first-out* (FIFO) Semantik beschrieben. Als Kommunikationsmechanismen unterstützt TTCN-3 den Austausch von Nachrichten und die prozedurbasierte Kommunikation. Bei der prozedurbasierten Kommunikation werden Prozeduren bei einem Kommunikationspartner ausgeführt. Nach der Beendigung der Prozedur wird meistens ein Ergebnis zurück an den aufrufenden Partner gesendet. Aus Platzgünden werden in diesem Artikel nur Beispiele für die nachrichtenbasierte Kommunikation vorgestellt.

Bild 3, Zeile 1–4, zeigt exemplarisch die Definition eines Ports für die Kommunikation mit den Lagern des SUT. Wie durch das Schlüsselwort **message** gekennzeichnet, ist

```
 1: type port Lager message {
 2:   out Bestellung;
 3:   in Lieferung;
 4: }
 5: type component FabrikPTC {
 6:   port Lager lager;
 7:   port TestKoordination arbeiter;
 8: }
 9: type component ArbeiterMTC {
10:   timer t;
11:   port Foerderband band;
12:   port TestKoordination fabrikC, fabrikD;
13: }
14: type component TestSystem {
15:   port Lager lagerC, lagerD;
16:   port Foerderband band;
17: }
```

Bild 3: Port- und Komponenten-Definitionen.

`Lager` ein Port für nachrichtenbasierte Kommunikation. In TTCN-3 werden Nachrichten als Datentypen definiert und prinzipiell kann jeder Wert eines Typs gesendet und empfangen werden. Über einen Port des Typs `Lager` können Werte des Typs `Bestellung` gesendet und Werte des Typs `Lieferung` empfangen werden.

Testkomponenten besitzen lokale Ports und können zusätzlich noch lokale Timer, Konstanten und Variablen verwalten. Eine Testkomponente vom Typ `ArbeiterMTC` (Bild 3, Zeile 9–13) besitzt einen Timer `t` und die Ports `band`, `fabrikC` und `fabrikD`.

In TTCN-3 wird das SUT durch die Menge seiner Ports charakterisiert und daher als Testkomponente ohne lokale Timer, Konstanten und Variablen definiert. Im vorliegenden Beispiel ist das SUT durch den Komponententyp `TestSystem` (Zeilen 14–17) beschrieben.

TTCN-3 unterstützt *dynamische Testkonfigurationen*. Beim Start eines Testfalls wird zunächst eine ausgezeichnete Testkomponente, die *Haupttestkomponente* (MTC, für engl. *Main Test Component*), gestartet. Die MTC und jede einmal erzeugte Testkomponente können weitere Testkomponenten und Verbindungen erzeugen. Jede Testkomponente kann sich selbst stoppen oder terminiert automatisch, wenn die MTC terminiert.

2.4 Testfälle

Ein *Testfall* ist ein Programm, das Folgen von Testereignissen zusammen mit einer Beurteilung der einzelnen Folgen beschreibt. In TTCN-3 wird ein Testfall durch eine ausgezeichnete Funktion beschrieben, die mit einem Testurteil endet. Die im Funktionskörper enthaltene Verhaltensbeschreibung spezifiziert das Verhalten der MTC. Das Testfallbeispiel in Bild 4 prüft, ob der Roboter bei Eingabe eines A-Teils ein korrektes AC-Produkt und kein fehlerhaftes ACD-Produkt fertigt.

Die Signatur des Testfalls besteht aus dem Testfallnamen, der – hier leeren – Liste der formalen Parameter, einer Referenz auf den Typ der MTC (**runs on** `ArbeiterMTC`) und einer Referenz auf den Typ des SUT (**system** `TestSystem`). Im Testfall werden zunächst

```
 1: testcase acTest()
 2:   runs on ArbeiterMTC system TestSystem {
 3:   var default def := activate(fehlerBehandlung());
 4:   var Fabrik fabrikC_PTC := FabrikPTC.create,
 5:               fabrikD_PTC := FabrikPTC.create;
 6:   map(fabrikC_PTC:lager, system:lagerC);
 7:   map(fabrikD_PTC:lager, system:lagerD);
 8:   map(self:band, system:band);
 9:   connect(self:fabrikC, fabrikC:arbeiter);
10:   connect(self:fabrikD, fabrikD:arbeiter);
11:   fabrikC.start(fabrikation(cTeil));
12:   fabrikD.start(fabrikation(dTeil));
13:   t.start(maxZeitBearbeitung);
14:   band.send(Platzierung:aTeil);
15:   alt {
16:     [ ] band.receive(Abnahme:acTeil) {
17:         fabrikC_PTC.send(testende);
18:         fabrikD_PTC.send(testende);
19:       }
20:     [ ] band.receive(Abnahme:adcTeil) {
21:         verdict.set(fail);
22:         stop;
23:       }
24:   }
25:   all component.done;
26:   verdict.set(pass);
27:   stop;
28: }
```

Bild 4: TTCN-3–Testfall.

das Verhalten beim Auftreten von Sonderfällen aktiviert (**activate**-Anweisung; siehe Abschnitt 2.6), die verschiedenen Testkomponenten erzeugt (**create**-Anweisungen), die neu erzeugten Testkomponenten mit anderen Testkomponenten und dem SUT verbunden (**map**- bzw. **connect**-Anweisungen) und danach gestartet (**start**-Anweisung). Das Verhalten der einzelnen Testkomponenten wird dabei als Referenz auf eine Funktionsdefinition übergeben.

Das eigentliche Testverhalten beginnt mit dem Setzen des Timers `t` (**start**-Anweisung) und dem Senden eines Teils `aTeil` an das Förderband (**send**-Anweisung). Wie innerhalb der **alt**-Anweisung (Zeilen 15–24) spezifiziert, kann danach entweder ein korrektes Produkt `acTeil` oder ein fehlerhaftes Produkt `adcTeil` vom Förderband empfangen werden. Die beiden Alternativen werden von leeren [ ]-Klammern und **receive**-Anweisungen eingeleitet. Innerhalb der Klammern könnten boolesche Bedingungen spezifiziert werden, um einzelne Alternativen während der Ausführung ein- und auszuschalten.

Wird ein `acTeil` empfangen (Zeile 16), signalisiert die MTC das Ende des Tests an die anderen Testkomponenten. Danach wartet die MTC auf die Terminierung der Testkomponenten (**done**-Anweisung in Zeile 25), das Testurteil **pass** wird gesetzt und die MTC stoppt. Wird hingegen ein fehlerhaftes `adcTeil` empfangen (Zeile 20), setzt die MTC das Testurteil **fail** und terminiert.

2.5 Funktionen

TTCN-3–Funktionen dienen zur Strukturierung des Testverhaltens, aber auch zur Berechnung von Werten. Eine Funktion, die Testverhalten spezifiziert, besitzt eine **runs on**-Anweisung, die den Typ der Testkomponente referen-

ziert, auf der die Funktion ausgeführt werden soll. Innerhalb der Funktion können dann die im Komponententyp deklarierten Variablen, Konstanten, Timer und Ports benutzt werden. So verwendet die in Bild 5, Zeilen 1–11, definierte Funktion `fabrikation` die in dem Komponententyp `FabrikPTC` deklarierten Ports (siehe Bild 3).

```
 1: function fabrikation(Teil t) runs on FabrikPTC {
 2:   alt {
 3:     [] lager.receive(standardBestellung) {
 4:          lager.send(t);
 5:          repeat;
 6:     [] arbeiter.receive(testende) {
 7:          verdict.set(pass);
 8:        }
 9:   }
10:   stop;
11: }
12: teststep fehlerBehandlung() {
13:   [] any timer.timeout {
14:        verdict.set(fail);
15:        stop;
16:     }
17: }
```

Bild 5: TTCN-3–Funktion und TTCN-3–Testschritt.

2.6 Testschritte

TTCN-3 ermöglicht es, Abweichungen vom erwarteten Testablauf in kompakter Form durch *Testschritte* (engl. *test steps*) zu definieren. In Bild 5, Zeilen 12–17, ist der Testschritt `fehlerBehandlung` definiert. Er bewirkt, dass eine Testkomponente mit einem negativen Testurteil terminiert, sobald ein Timer abläuft. Der Testschritt wird mit Hilfe der **activate**-Anweisung in Bild 4 (Zeile 3) als *Default-Verhalten* aktiviert und damit bei Ausführung der MTC berücksichtigt. Auf diese Weise ist sichergestellt, dass der Testfall auch dann terminiert, wenn keine Nachricht über den Port *band* eintrifft (Bild 4, Zeilen 16 und 20) oder am Ende des Testfalls die übrigen Testkomponenten nicht terminieren (Zeile 25).

2.7 Testurteile

Zur Beurteilung von Testfällen bietet TTCN-3 den speziellen Datentyp **verdicttype** mit den Werten **pass**, **fail**, **inconc** und **error** an. Ein **pass** beschreibt den Fall, dass der Testzweck erreicht wird. Das Testurteil **fail** bedeutet, dass das SUT aufgrund der Testfolge als fehlerhaft beurteilt werden kann. Das Urteil **inconc** (für engl. *inconclusive*) beschreibt den Fall, dass der Testzweck nicht erreicht wird, das SUT jedoch auch nicht fehlerhaft ist. Ein **error** beschreibt das Scheitern des Testfalls aufgrund eines Fehlers in den Testgeräten.

In TTCN-3 besitzt jede Testkomponente ein implizites lokales Testurteil (engl. *verdict*), auf das mit den Operationen **get** und **set** zugegriffen werden kann (z. B. Bild 4, Zeile 21). Terminiert ein Testfall, wird automatisch aus den lokalen Urteilen der einzelnen Testkomponenten ein abschließendes Urteil für den Testfall ermittelt. Für das Setzen von Testurteilen definiert TTCN-3 spezielle Regeln, die z. B. verhindern, dass ein zuvor vergebenes **fail** nachträglich zu einem **pass** aufgewertet wird.

3 Ausblick

TTCN-3 wurde im Juni 2001 als Standard veröffentlicht und obwohl der Standard noch sehr neu ist, sind bereits kommerzielle Werkzeuge für die TTCN-3 und die standardisierten Präsentationsformate verfügbar. Sie unterstützen das Editieren, Kompilieren, Debuggen und die Ausführung von TTCN-3-Modulen. Erste Anwendungen von TTCN-3 haben gezeigt, dass die Sprache sehr ausdrucksstark, flexibel und einfach zu erlernen ist. TTCN-3 wird am ETSI gepflegt und weiterentwickelt. Basierend auf den Anforderungen der Anwender werden in regelmäßigen Abständen Erweiterungen und Korrekturen des Standards veröffentlicht. Für die Zukunft werden ein UML-basiertes Präsentationsformat und Spracherweiterungen für den Test von Realzeit-Anforderungen diskutiert.

In diesem Artikel konnten aus Platzgründen nur einige wichtige Elemente von TTCN-3 vorgestellt werden. Weitere Informationen und die vollständige TTCN-3–Sprachdefinition finden sich auf dem WWW-Server des ETSI (`http://www.etsi.org/ptcc`).

Literatur

[1] M. Ebner: *A Mapping of OMG IDL to TTCN-3*. Technischer Bericht SIIM-TR-A01-11 der Institute für Informatik/Mathematik, Universität Lübeck, Juli 2001.

[2] ETSI EN 101873-1 (2001-06): *Testing and Test Control Notation (TTCN-3); Part 1: TTCN-3 Core Language*.

[3] ETSI EN 101873-2 (2001-06): *TTCN-3; Part 2: Tabular Presentation Format for TTCN-3 (GFT)*.

[4] ETSI DES 101873-3 (2001-10): *TTCN-3; Part 3: Part 3: Graphical Presentation Format for TTCN-3 (GFT)*.

[5] J. Grabowski, A. Wiles, C. Willcock, D. Hogrefe: *On the Design of the new Testing Language TTCN-3*. In: *Testing of Communicating Systems – Tools and Techniques* (Herausgeber: H. Ural, R. L. Probert, G. von Bochmann). Kluwer Academic Publishers, August 2000.

[6] J. Grabowski, E. Rudolph, M. Schmitt: *Die Spezifikationssprachen MSC und SDL – Teil 1: Message Sequence Chart (MSC)*. In: at-Automatisierungstechnik 49, Heft 12, Dezember 2001.

[7] ISO/IEC IS 9646 (1994–1998): *Information Technology – Open Systems Interconnection – Conformance Testing Methodology and Framework*.

[8] ITU-T Rec. X.680 (1997): *Abstract Syntax Notation One (ASN.1) Specification of Basic Notation*.

[9] S. Kowalewski: *Modellierungsmethoden aus der Informatik*. In: at-Automatisierungstechnik 49, Heft 9, Sept. 2001.

[10] A. Pope: *The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture*. Addison-Wesley, 1998.

Manuskripteingang: 23. November 2001.

Vorstellung der Autoren in Teil 1, at 12(2001) A19.