K. G. Saur Verlag, München, 1994

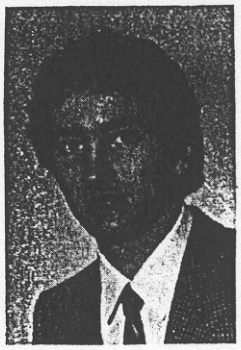
Jens Grabowski, Robert Nahm**, Andreas Spichiger***, Dieter Hogrefe**

Die SAMSTAG Methode und ihre Rolle im Konformitätstesten

Dr. Jens Grabowski, Dipl.-Inf.; Studium der Informatik an der Universität Hamburg. Seit 1990 wissenschaftlicher Mitarbeiter in der Fachgruppe „Rechnernetze und verteilte Systeme" am Institut für Informatik der Universität Bern. Promotion 1994. Hauptarbeitsgebiete: Formale Methoden für die Spezifikation von Telekommunikationssystemen, Konformitätstesten.

Dr. Robert Nahm, Dipl.-Inf.; Studium der Informatik an der Technischen Universität München. 1991-1994 wissenschaftlicher Mitarbeiter in der Fachgruppe „Rechnernetze und verteilte Systeme" am Institut für Informatik der Universität Bern. Promotion 1994. Zur Zeit Gastwissenschaftler in der Zentralabteilung für Forschung und Entwicklung der Siemens AG München. Arbeitsgebiete: Formale Methoden der Softwareentwicklung und Softwaretesten für Telekommunikationssysteme.

Dr. Andreas Spichiger, Dipl.-Inf.; Studium der Informatik an der Universität Bern. 1990-1994 wissenschaftlicher Mitarbeiter in der Fachgruppe „Rechnernetze und verteilte Systeme" am Institut für Informatik der Universität Bern. Promotion 1994. Seit Juli 1994 Mitarbeiter der Siemens-Albis AG Zürich. Arbeitsgebiete: Testmethodik für Telekommunikationssysteme, Netzwerkmanagement.



Prof. Dr. Dieter Hogrefe: Mathematik- und Informatik-Studium an der TU Hannover, 1985 Promotion. Von 1983-1986 Mitarbeiter bei der Siemens AG München, wo er sich mit der Analyse von neueren Telekommunikationssystemen beschäftigt hat. 1986 - 1989 Hochschulassistent an der Universität Hamburg mit Habilitation. Zur Zeit Forschung und Lehre auf dem Gebiet der Rechnernetze und verteilten Systeme am Institut für Informatik der Universität Bern

* Universität Bern, Institut für Informatik, Länggassstrasse 51, CH-3012 Bern
** Siemens AG München, ZFE BT SE 4, Otto-Hahn-Ring 6, D-81739 München
*** Siemens-Albis AG, EAT 3, Steinenschanze 2, CH-4051 Basel

Zusammenfassung

Für das Konformitätstesten von Kommunikationsprotokollen ist die automatische Generierung von Testfällen aus formalen Spezifikationen ein bisher nur unzureichend gelöstes Problem. Von wissenschaftlichen Institutionen wurden zwar eine ganze Reihe von Methoden zur Testfallgenerierung entwickelt; sie sind jedoch in der Praxis kaum einsetzbar. Zum einen sind reale Systeme häufig so komplex, daß diese Methoden nicht mehr testbar sind. Zum anderen hat man sich bei der Methodenentwicklung nur wenig an dem in der Praxis üblichen Vorgehen beim Konformitätstesten orientiert. Dieses Vorgehen ist insbesondere im internationalen ISO/IEC Standard 9646 [11], im folgenden CTMF genannt, beschrieben. Mit der von uns entwickelten SAMSTAG Methode lassen sich ebenfalls Testfälle für Konformitätstests generieren. Im Gegensatz zu anderen Methoden orientiert sich die SAMSTAG Methode eng an CTMF. In diesem Artikel werden die angesprochenen Probleme diskutiert und die SAMSTAG Methode eingeführt.

1 EINLEITUNG

Vor ungefähr 10 Jahren haben ISO/IEC und ITU-T[1] begonnen, sich mit dem Konformitätstesten von Kommunikationsprotokollen zu beschäftigen. Ein Resultat dieser Arbeiten ist der internationale ISO/IEC Standard 9646 *„OSI Conformance Testing Methodology and Framework"* (CTMF) [11][2]. Dieser siebenteilige Standard beschreibt ein allgemeines Vorgehen, um eine Protokollimplementierung auf seine Konformität zu einer Protokollspezifikation zu überprüfen. CTMF beinhaltet u.a. die generellen Konzepte für das Konformitätstesten, verschiedene Testarchitekturen und Testmethoden, die *„Tree and Tabular Combined Notation"* (TTCN) als Beschreibungssprache für Testfälle und die Testrealisierung. CTMF scheint auch eine gute Grundlage für das in der industriellen Praxis durchgeführte Protokolltesten zu sein. Die hohe Akzeptanz und die zunehmende Verbreitung von CTMF zeigt sich an dem wachsenden Angebot von kommerziellen Werkzeugen, die einzelne Teile des Vorgehens beim Konformitätstesten unterstützen und automatisieren [1, 18].

Parallel zur Entwicklung von CTMF haben ISO/IEC und ITU-T die Spezifikationssprachen SDL, Estelle, LOTOS [7] und MSC [14] entwickelt. Durch die Anwendung dieser Sprachen in den Standards sollen unter anderem die Validierung und das Konformitätstesten von Kommunikationsprotokollen unterstützt werden. Hierfür wird CTMF zur Zeit von ISO/IEC und ITU-T im Rahmen des gemeinsamen Normierungsprojektes *„Formal Methods in Conformance Testing"* [12, 9], kurz FMCT genannt, formalisiert und erweitert, d.h. auf das Kon-

[1] Der *‚Telecommunication Standards Sector of the International Telecommunication Union'* (ITU-T) ist die Nachfolgeorganisation des CCITT. Das CCITT wurde im März 1993 aufgelöst.

[2] CTMF wird von der ITU-T als Empfehlungen X.290-X.296 herausgegeben.

formitätstesten von formal spezifizierten Kommunikationsprotokollen zugeschnitten. Da sich dieser Artikel ebenfalls mit dem Konformitätstesten von formal spezifizierten Protokollen beschäftigt, können CTMF und FMCT nicht getrennt voneinander betrachtet werden. Um auf die Arbeiten in CTMF und FMCT zu verweisen, benutzen wir nachfolgend die Abkürzung CTMF/FMCT.

Die Einführung formaler Methoden zur Protokollbeschreibung hat dazu geführt, daß man sich im wissenschaftlichen Umfeld intensiver mit dem Konformitätstesten beschäftigt hat. Dabei sind eine ganze Reihe von Methoden zur automatischen Generierung von Testfällen entwickelt worden. Wir bezeichnen diese Methoden nachfolgend kurz als *theoretische Methoden*. Eine Diskussion solcher Methoden findet sich z. B. in [10]. Bei allen theoretischen Methoden soll eine Relation, die sog. Konformitätsrelation, zwischen einer Spezifikation und einer Implementierung über einen Test nachgewiesen werden. Die Testfallgenerierung wird daher durch die Spezifikation und durch die Konformitätsrelation bestimmt. Bezogen auf das Testen realer Systemen und auf CTMF/FMCT treten insbesondere drei Probleme auf. Das erste Problem betrifft die Konformitätsrelation. Viele der verwendeten Konformitätsrelationen sind nur für Spezifikationen und Implementierungen nachweisbar, deren Verhalten sich in Form von endlichen Automaten beschreiben lassen. In der Praxis ist diese Einschränkung fast nie gegeben. Das zweite Problem ist die Komplexität von realen Systemen. Selbst wenn eine Konformitätsrelation nachweisbar ist, so ist die Anzahl der notwendigen Tests häufig so groß, daß sie in der Praxis nie durchgeführt werden können. Ein drittes Problem besteht darin, daß sich die in den theoretischen Methoden verwendeten Begriffe und Vorgehensweisen nicht direkt auf CTMF/FMCT abbilden lassen. Zusammenfassend kann daher gesagt werden, daß die theoretischen Methoden das in CTMF/FMCT beschriebene Vorgehen zwar befruchtet, aber bisher nur wenig vereinfacht haben.

CTMF/FMCT und die theoretischen Methoden lassen sich jedoch näher zusammenbringen. Eine wichtige Rolle können hierbei die Ergebnisse des Projekts *„Conformance Testing - Ein Werkzeug zur Generierung von Testfällen"* spielen. Dieses Projekt hat sich eng an CTMF/FMCT orientiert. Ein Hauptziel ist es, einen wichtigen Schritt im allgemeinen Vorgehen beim Konformitätstesten nach CTMF/FMCT zu formalisieren und durch ein Prototypwerkzeug zu automatisieren. Der ausgewählte Schritt betrifft die Generierung von Testfällen basierend auf einer formalen Protokollspezifikation und einer Menge von Testzwecken. Wir haben den in CTMF/FMCT nur informal definierten Begriff *Testzweck* (engl. test purpose) formalisiert und mit der SAMSTAG (Sdl And Msc baSed Test cAse Generation) Methode ein Verfahren für die Testfallgenerierung entwickelt. Mit dem SAMSTAG Werkzeug wurde die SAMSTAG Methode darüberhinaus in Form eines Prototyps implementiert.

2 EIN VERGLEICH VON CTMF/FMCT MIT ANDEREN METHODEN ZUR TESTFALLGENERIERUNG

In diesem Kapitel wird zunächst das in CTMF/FMCT standardisierte Vorgehen für das Konformitätstesten erläutert (Abschnitt 2.1). Danach gehen wir auf die theoretischen Methoden ein (Abschnitt 2.2). Abschließend werden die Gemeinsamkeiten und Unterschiede der beiden Vorgehensweisen noch einmal zusammengefasst (Abschnitt 2.3).

2.1 Das Konformitätstesten nach CTMF/FMCT

In Abbildung 1 ist das allgemeine Vorgehen dargestellt, um nach CTMF/FMCT für eine Protokollspezifikation[3] und eine Protokollimplementierung über einen Konformitätstest zu einer Konformitätsaussage zu gelangen[4]. Die Rechtecke bezeichnen Aktionen, während die Ellipsen Eingabe- und Ausgabedaten der Aktionen beschreiben. Gepunktete Rechtecke und Ellipsen bezeichnen Aktionen und Daten, die in CTMF/FMCT nur informal beschrieben sind. Zahlen und Buchstaben dienen als Referenzen für die nachfolgende Beschreibung.

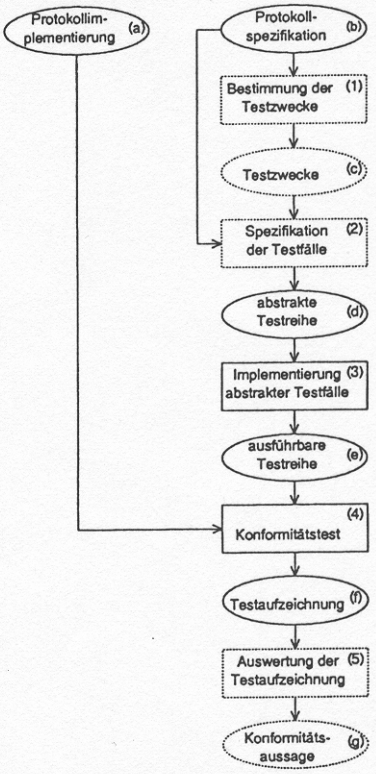


Abb. 1: Das Vorgehen beim Konformitätstesten nach CTMF/FMCT

Das Ziel des Konformitätstestens ist es, nachzuweisen, daß eine Protokollimplementierung (a) das in einer Spezifikation (b) beschriebene Verhalten besitzt. Ein Konformitätstest ist ein *Black Box Test*, d.h. die interne Struktur der Implementierung ist dem Tester nicht bekannt. Man kann den Code einer Implementierung daher nicht direkt gegen die Spezifikation verifizieren, sondern muß sie mit einer Menge von Testfällen, einer *Testreihe,* testen.

Um aus einer Spezifikation eine Testreihe zu entwickeln, müssen *Testzwecke* (c) definiert werden. Ein Testzweck ist die informale textuelle Beschreibung eines Verhaltens, das die Implementierung während des Tests zeigen muß. Aus den Testzwecken wird, unter Berücksichtigung der Spezifikation, eine *abstrakte Testreihe* (d), bestehend aus *abstrakten Test-*

[3] Der hier verwendete Begriff der Protokollspezifikation bezeichnet im allgemeinen einen ISO, ITU-T oder ETSI Standard. Die Begriffe Protokollspezifikation und Spezifikation werden in diesem Artikel synonym verwendet.

[4] Aufgrund der Komplexität des in CTMF/FMCT beschriebenen Vorgehens, kann Abbildung 1 nicht alle Einflüsse auf das Vorgehen beim Konformitätstesten beschreiben. So werden z. B. die Einflüsse von PICS und PIXIT hier nicht berücksichtigt.

fällen, erzeugt. Die Testzwecke (c), die Bestimmung der Testzwecke (1) und auch die Spezifikation der abstrakten Testreihe (2) sind in CTMF/FMCT nur sehr informal beschrieben. In der Praxis werden die Aktionen (1) und (2) von Experten per Hand durchgeführt.

Ein abstrakter Testfall beschreibt den geforderten Austausch von Protokolldateneinheiten und Dienstprimitiven unabhängig von der Testrealisierung und von der Protokollimplementierung. Um eine abstrakte Testreihe (d) in eine *ausführbare Testreihe* (e) zu transformieren (3), müssen alle Protokolldateneinheiten und Dienstprimitiven in Bitkombinationen umgesetzt werden, die von der Protokollimplementierung und den Testgeräten interpretiert werden können.

Während des Konformitätstests (4) wird das Verhalten der Protokollimplementierung auf Eingaben beobachtet. Die Eingaben und die erwarteten Ausgaben sind in den einzelnen Testfällen beschrieben. Je nachdem, welches Verhalten die Implementierung zeigt, wird pro Testfall eines von drei Testurteilen vergeben. Der gesamte Testverlauf, einschließlich der vergebenen Testurteile, wird in einer *Testaufzeichnung* (f) protokolliert.

Die Auswertung einer Testaufzeichnung (5) und die sich daraus ergebende *Konformitätsaussage* (g) sind in CTMF/FMCT nur sehr allgemein beschrieben. Ein Grund dafür besteht darin, daß eine allgemeine Konformitätsaussage neben dem Testergebnis auch andere technische und nicht technische Aspekte, wie z. B. die Seriosität des Herstellers, berücksichtigen soll.
Von den Aktionen in Abbildung 1 sind (3) und (4) automatisierbar, während (1), (2) und (5) (zur Zeit) nur informal beschrieben und daher nicht automatisierbar sind. Die Vorbedingung für die Bestimmung der Testzwecke (1) ist eine Protokollspezifikation, für die wir annehmen können, daß sie in einer standardisierten formalen Beschreibungstechnik, d.h. in LOTOS, Estelle oder SDL [7], vorliegt. Obwohl die Spezifikation das Verhalten des Protokolls eindeutig beschreibt, ist Aktion (1) nicht automatisierbar, da der zentrale Begriff des Testzwecks und seine Beziehung zur Protokollspezifikation noch nicht formal geklärt sind. Ebenso ist die Beziehung zwischen einem Testzweck und einem Testfall ungeklärt, so daß auch die Bestimmung der Testreihe (2) noch nicht automatisiert ist. Für die (manuelle) Spezifikation von abstrakten Testfällen kann mit TTCN eine standardisierte Sprache eingesetzt werden. Für die automatische Durchführung von Aktion (3) existieren kommerzielle TTCN Compiler. Auch die technische Ausrüstung für die automatische Durchführung von Konformitätstests kann oftmals käuflich erworben werden.

2.2 Die Testfallgenerierung mit theoretischen Methoden

Die im wissenschaftlichen Umfeld entwickelten Methoden, hier als theoretische Methoden bezeichnet, generieren aus einer formalen Spezifikation Testfälle, mit denen man eine sog. Konformitätsrelation zwischen der Spezifikation und einer Implementierung überprüfen kann. Das Vorgehen bei den theoretischen Methoden wird nachfolgend mit Hilfe von Abbildung 2 und anhand eines Beispiels erläutert. Das Beispiel bezieht sich auf ein von Holzmann in [10] diskutiertes Verfahren zur Testfallgenerierung für endliche Automaten, der UIO Methode kombiniert mit einer Transition Tour. Buchstaben und Zahlen im nachfolgenden Text sind Referenzen auf Abbildung 2.

Prinzipiell wird bei allen theoretischen Methoden zuerst eine Konformitätsrelation (h) festgelegt. Die Konformitätsrelation

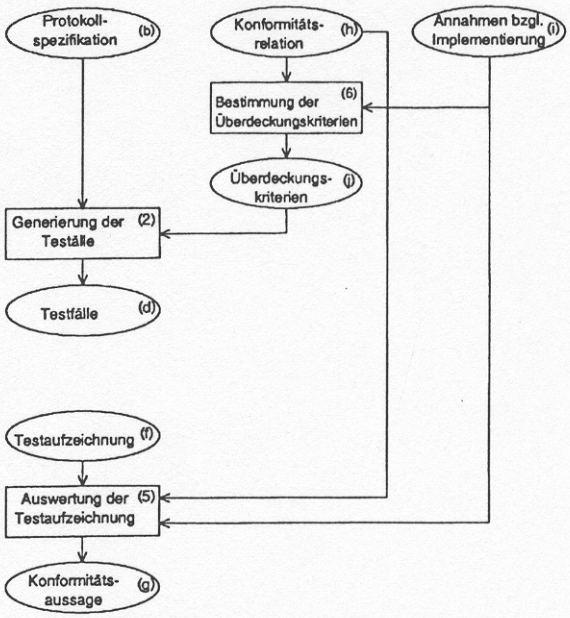


Abb. 2: Testfallgenerierung mit theoretischen Methoden

legt die Beziehung zwischen den Abläufen der Spezifikation (b) und denen der Implementierung fest. Die Gültigkeit dieser Relation soll mit einem Test nachgewiesen werden. Verschiedene Konformitätsrelationen sind jedoch nicht für beliebige Spezifikationen und Implementierungen testbar.

Bei dem von Holzmann beschriebenen Verfahren wird die *Verhaltensäquivalenz* als Konformitätsrelation verwendet. Die Relation ist gültig, wenn die Implementierung jeden Ablauf der Spezifikation durchführen kann und umgekehrt. Häufig fordern Spezifikationen unendliches Verhalten von einer Implementierung. In so einem Fall kann mittels eines endlichen Tests nicht jedes mögliche und erlaubte Verhalten der Implementierung überprüft werden. Deshalb werden bei den theoretischen Methoden nicht die Verhalten von Spezifikation und Implementierung verglichen, sondern ihre interne Struktur.

Beim Konformitätstesten ist die interne Struktur der Spezifikation bekannt, die Implementierung liegt aber nur als Black Box vor. Um überhaupt Aussagen über das Verhalten der Implementierung treffen zu können, machen die theoretischen Methoden Annahmen über die interne Struktur der Implementierung (i). Zum Nachweis einer Verhaltensäquivalenz, muß sich die Spezifikation durch einen vollständig bekannten endlichen Zustandsautomaten beschreiben lassen. Für die Testfallgenerierung nimmt man an, daß sich das Verhalten der zu testenden Implementierung durch einen äquivalenten Automaten beschreiben läßt.

Aus der Konformitätsrelation (h) und den Annahmen (i) läßt sich ein Überdeckungskriterium (j) bestimmen. Das Überdeckungskriterium besagt, welcher Teil einer Spezifikation getestet werden muß, um die Gültigkeit der Konformitätsrelation nachzuweisen. Für die Überprüfung einer Verhaltensäquivalenz muß für jeden Zustandsübergang des Automatens, der die Spezifikation repräsentiert, ein äquivalenter Zustandsübergang in der Implementierung nachgewiesen werden.

Mittels des Überdeckungskriteriums und der Spezifikation erfolgt die Generierung der Testfälle (2). Bei dem von Holzmann beschriebenen Verfahren liefert die UIO Methode für jeden Zu-

standsübergang eine Folge von Eingabe- und Ausgabesignalen, die den Zustandsübergang eindeutig identifiziert. Es wird jedoch nicht für jeden Zustandsübergang ein eigener Testfall generiert, sondern mittels einer sog. *Transition Tour* ein einziger großer Testfall erzeugt, der alle Zustandsübergänge auf einmal testet. Die Details der Berechung von UIO Sequenzen und Transition Tour sind in [10] beschrieben.

Die Durchführung eines Konformitätstests wird von den theoretischen Methoden nicht näher beschrieben. Es wird allerdings angenommen, daß für die Auswertung der Tests (5) eine Testaufzeichnung (f) zur Verfügung steht. Basierend auf der Testaufzeichnung, den Annahmen (i), und der Konformitätsrelation (h) wird eine abschließende Konformitätsaussage (g) gefolgert.

Bei der Transition Tour ist dieses sehr einfach. Wurde der mit der Transition Tour generierte Testfall erfolgreich durchgeführt, ist die Konformitätsrelation nachgewiesen, sofern die Implemtierung die vorher gemachten Annahmen (i) erfüllt. Wurde der Testfall nicht erfolgreich durchgeführt, so ist die Konformitätsrelation nicht gültig.

Obwohl die theoretischen Methoden eine formale Konformitätsaussage liefern, haben sie doch eine Reihe von Nachteilen. Sie benutzen Annahmen über die interne Struktur der Implementierung und durchbrechen damit die Prinzipien des Konformitätstestens, das in CTMF/FMCT als reiner Black Box Test charakterisiert wird. Zudem scheitert die Anwendung der theoretischen Methoden häufig an Komplexitätsproblemen. So wächst z. B. der Zeitbedarf bei der Berechnung von UIO Sequenzen exponentiell mit der Größe des Zustandsautomatens, der das Verhalten die Spezifikation beschreibt. UIO Sequenzen können nur für kleine Spezifikationen berechnet werden. Darüberhinaus ist der für das Konformitätstesten nach CTMF/FMCT so wichtige Begriff Testzweck (vgl. Abbildung 1 (c)) von den theoretischen Methoden bisher nicht formalisiert worden.

2.3 Eine vergleichende Zusammenfassung

Das Ziel von CTMF/FMCT und den theoretischen Methoden ist es, das gesamte Vorgehen beim Konformitätstesten, von der Testfallentwicklung bis hin zur Konformitätsaussage, zu automatisieren. Dieses läßt sich nur durch eine Formalisierung der einzelnen Vorgehensschritte und der dabei anfallenden Daten erreichen.

Durch eine Zusammenfassung der Abbildungen 1 und 2 zu Abbildung 3 läßt sich der Beitrag, den CTMF/FMCT und die theoretischen Methoden für eine vollständige formale Beschreibung des Konformitätstestens leisten können, anschaulich darstellen. Auch der Beitrag, den die nachfolgend vorgestellte SAMSTAG Methode leisten soll, ist bereits angedeutet. Natürlich gibt es Überschneidungen bei den drei Methoden, die jedoch aus Gründen der Übersichtlichkeit in der Abbildung weggelassen worden sind.

Durch CTMF/FMCT wird der gesamte Prozeß des Konformitätstestens standardisiert. Dabei sind die einzelnen Vorgehensschritte jedoch unterschiedlich detailliert beschrieben und damit auch unterschiedlich schwer zu automatisieren. Hinreichend formal, d.h. automatisierbar, ist die Testdurchführung, von der abstrakten Testreihe (d) bis zur Testaufzeichnung (f), definiert. Demgegenüber sind die Begriffe Testzweck (c) und Konformitätsaussage (g), sowie die Aktionen Bestimmung der Testzwecke (1), Spezifikation der Testfälle (2) und Auswertung der Testaufzeichnung (5) nur sehr informal beschrieben. Dieses ist aus mehreren Gründen problematisch.

Die Entwicklung der abstrakten Testreihe basiert auf zwei informalen Schritten, die von menschlichen Experten per Hand durchgeführt werden müssen. Bei der Testreihenentwicklung wird man versuchen, die Funktionen des Protokolls möglichst vollständig zu testen. Wie gut dieses gelingt, hängt stark von der Bestimmung der Testzwecke (1) ab. Ohne formale Kriterien ist es jedoch sehr schwer, die Menge der Testzwecke zu beurteilen. Bei der Spezifikation der Testfälle (2) muß ein Experte die Bedeutung der Testzwecke interpretieren und in eine Testfallnotation umsetzen. Beide Tätigkeiten sind nicht trivial und dementsprechend kann eine Testreihe Fehlinterpretationen und Fehler enthalten. Die beschriebenen Probleme spiegeln sich auch bei der Auswertung der Testaufzeichnung (5) wider. Ohne ein Kriterium, wie gut eine Testreihe die Funktionen einer Protokollspezifikation überprüft, ist die abschließende Konformitätsaussage sehr wenig aussagekräftig.

Die theoretischen Methoden formalisieren die Generierung der Testfälle (2) und die Auswertung der Testaufzeichnung (5). Dieses wird insbesondere durch eine Konformitätsrelation erreicht, die das Ziel des Testens festlegt. Mit einer Konformitätsrelation kann ein Überdeckungskriterium (j) bestimmt werden. Mit Hilfe dieses Kriteriums und der Spezifikation läßt sich die Testreihe für einen Konformitätstest automatisch generieren. Bei der Auswertung einer Testaufzeichnung (5) wird überprüft, ob alle Testfälle einer Testreihe erfolgreich durchgeführt worden sind. In diesem Fall ist die Konformitätsrelation nachgewiesen.

Die Hauptprobleme der theoretischen Methoden sind die Konformitätsrelation und die Komplexität realer Systeme. Es ist nicht jede Konformitätsrelation für beliebige Spezifikationen und Implementationen testbar. Die gewählte Konformitätsrelation schränkt daher die Menge der testbaren Spezifikationen und Implementationen ein. Die meisten Konformitätsrelationen sind für reale Systeme nicht testbar. Selbst wenn eine Konformitätsrelation überprüfbar ist, so ist die Anzahl der dafür notwendigen Tests oft so groß, daß sie in der Praxis aus Zeit- und Kostengründen nicht durchgeführt werden können.

Aufgrund der Konformitätsrelation und des daraus resultierenden Überdeckungskriteriums wird der Begriff des Testzwecks bei den theoretischen Methoden nicht benötigt. Er wird deshalb auch nicht formalisiert. Aus den genannten Gründen sind die theoretischen Methoden jedoch selten auf reale Protokolle anwendbar. Für die Beurteilung von Testreihen und Testaufzeichnungen müssen deshalb andere Mechanismen benutzt werden. In der industriellen Praxis spielen hierbei die Testzwecke eine zentrale Rolle. Ein Hersteller muß z. B. seinen Kunden nachweisen, daß ein Produkt die gestellten Anforderungen erfüllt. Hierfür kann es notwendig sein, dem Kunden die Testfälle und die Testaufzeichnung zur kritischen Begutachtung zu überlassen. Ohne den Zweck der einzelnen Testfälle zu kennen, ist der Kunde kaum in der Lage, eine Begutachtung durchzuführen.

Zusammenfassend kann gesagt werden, daß sich die theoretischen Methoden nur unvollständig auf das in CTMF/FMCT standardisierte Vorgehen für das Konformitätstesten abbilden lassen. Insbesondere wird der zentrale Begriff des Testzwecks

nicht erklärt. Die von uns entwickelte SAMSTAG Methode orientiert sich eng an dem in CTMF/FMCT beschriebenen Vorgehen, formalisiert den Begriff des Testzwecks und liefert die Algorithmen, mit denen sich automatisch abstrakte Testfälle aus einer formalen Spezifikation und einer Menge von Testzwecken generieren lassen (Aktion 2 in Abbildung 3).

3 DIE SAMSTAG METHODE UND DAS SAMSTAG WERKZEUG

Die SAMSTAG Methode ist im Rahmen des von der Generaldirektion der Schweizer PTT finanzierten Forschungs- und Entwicklungsprojektes *„Conformance Testing - Ein Werkzeug zur Generierung von Testfällen"* entwickelt worden. Das Ziel dieses Projektes ist die Entwicklung einer Methode, mit deren Hilfe sich basierend auf SDL Spezifikationen [13, 3, 4] und Message Sequence Charts (MSCs) [14, 5] komfortabel TTCN Testfälle [11] generieren lassen. Dabei wird angenommen, daß das erlaubte Verhalten des zu testenden Systems durch eine SDL Spezifikation beschrieben und daß der Zweck eines zu generierenden Testfalls durch einen MSC vorgegeben ist.

SAMSTAG ist eine Abkürzung für *„Sdl And Msc baSed Test cAse Generation"*. In dieser Abkürzung spiegelt sich das Projektziel wider, obwohl die Methode so verallgemeinert worden

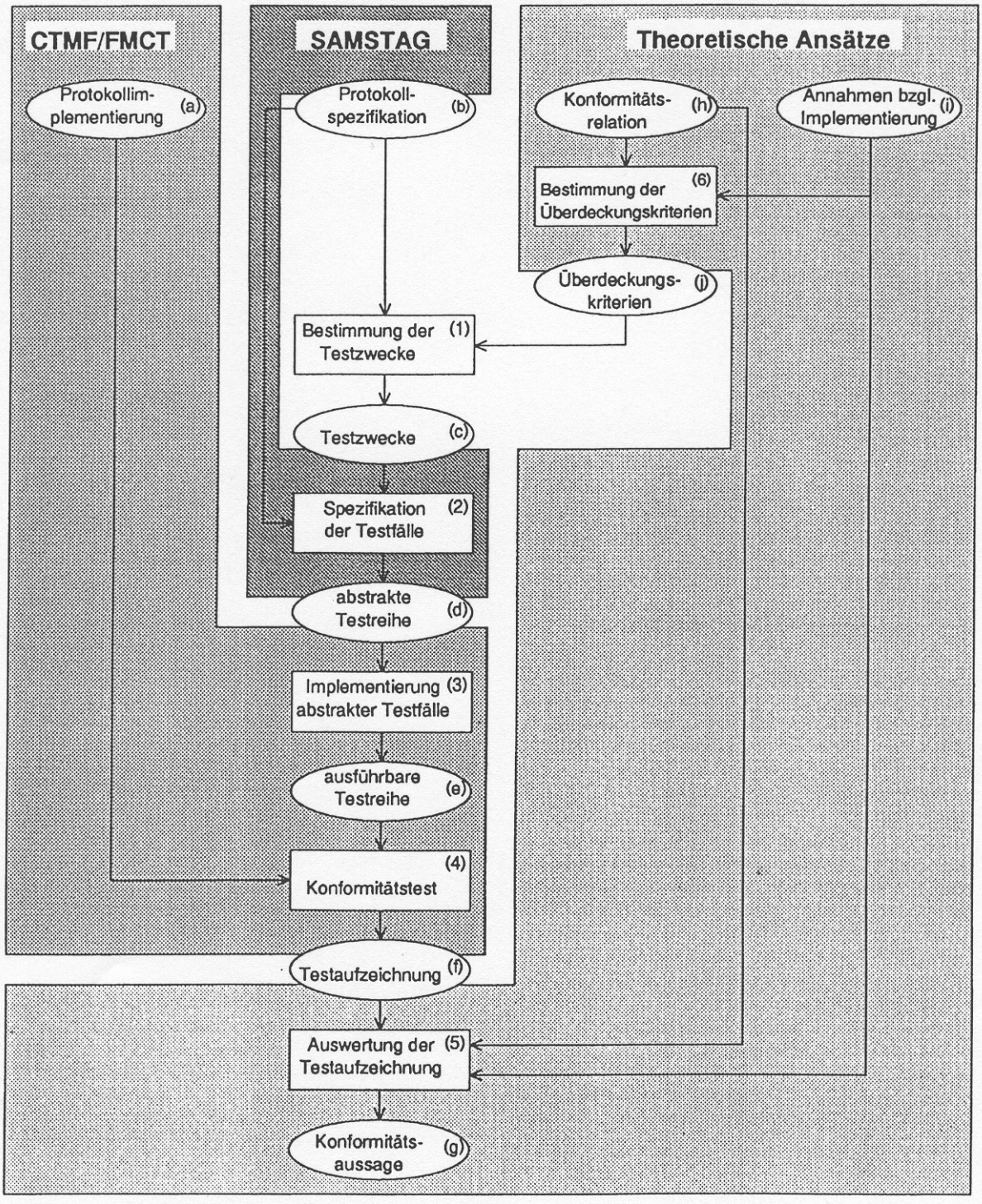


Abb. 3: Eine vollständige formale Beschreibung des Konformitätstestes

ist, daß sie auch für Protokollspezifikationen und Testzwecke, die nicht mit SDL und MSC beschrieben sind, verwendet werden kann. Zur Erklärung der SAMSTAG Methode (Abschnitt 3.2), der Testfallgenerierung (Abschnitt 3.3) und dem SAMSTAG Werkzeug (Abschnitt 3.4) führen wir zunächst einige grundlegende Begriffe ein.

3.1 Grundlegende Begriffe

Die hier eingeführten Begriffe basieren weitgehend auf CTMF/FMCT [11] und den ITU-T Empfehlungen Z.100 [13] und Z.120 [14]. Einzig die Begriffe *Ablauf* und *Beobachtung* werden zur Vereinfachung der Darstellungen in den nachfolgenden Abschnitten neu eingeführt.

SDL. Die *„Specification and Description Language"* (SDL) ist eine von der ITU-T standardisierte Spezifikationssprache für verteilte Systeme und insbesondere für Telekommunikationssysteme [13]. Eine SDL Spezifikation beschreibt eine Menge von endlichen erweiterten Automaten, die asynchron Signale austauschen. Das Gesamtverhalten einer SDL Spezifikation läßt sich formal als ein Labeled Transition System, d.h. durch einen unendlichen Automaten, auffassen. Bei den nachfolgenden Ausführungen gehen wir nicht auf SDL spezifische Besonderheiten ein. Wir gehen davon aus, daß das Verhalten einer SDL Spezifikation durch ein Labeled Transition System gegeben ist.

MSC. *„Message Sequence Chart"* (MSC) ist ebenfalls eine von der ITU-T standardisierte Sprache [14]. Mit einem MSC[5] läßt sich ein Ablauf eines verteilten Systems visuell und intuitiv beschreiben (z.B. Abbildung 6). In einem MSC repräsentieren senkrechte Achsen die verschiedenen Prozesse eines Systems, während die ausgetauschten Signale durch beschriftete Pfeile dargestellt werden. Signalaussendung und Signalverarbeitung sind durch Anfang und Ende des zugehörigen Pfeils definiert. Die entlang einer Prozeßachse definierten Sende- und Verarbeitungsaktionen sind zeitlich total geordnet. Ordnungen zwischen Aktionen auf unterschiedlichen Prozeßachsen können durch Signale vermittelt sein. Formal definiert ein MSC eine Halbordnung über der Menge der im MSC enthaltenen Signalsende- und Signalempfangsaktionen. Das mit einem MSC dargestellte Systemverhalten läßt sich durch einen endlichen Automaten beschreiben, der alle erlaubten Folgen von Signalsende- und Signalempfangsaktionen akzeptiert. Näheres hierzu findet sich in [16]. MSC ähnliche Diagramme sind sehr weit verbreitet und werden insbesondere zur Dokumentation und Spezifikation unterschiedlichster Systeme eingesetzt. Eine vergleichende Beschreibung solcher Diagramme und eine Einführung in die MSC Sprache findet sich in [5].

Testarchitektur. Eine Testarchitektur beschreibt das zu testende System einschließlich einer Testumgebung, d.h. einschließlich der Prozesse (oder Testgeräte), die den Test steuern und kontrollieren. In CTMF/FMCT sind verschiedene Architekturen für Konformitätstests beschrieben. In Abbildung 5 ist eine Testarchitektur für den Test einer Implementierung des Initiators aus dem Inres Protokoll [8] dargestellt. Die Testprozesse sind der Upper Tester UT und der Lower Tester LT. In CTMF/FMCT wird davon ausgegangen, daß bei einer zu testenden Protokollinstanz die Schnittstelle zur nächst tieferen Protokollschicht nicht frei zugänglich ist. Deshalb muß diese Schnittstelle über den Dienst der tieferen Protokollschicht beobachtet und kontrolliert werden. Auch in Abbildung 5 wird die untere Schnittstelle des Initiators über den Medium Dienst getestet. Für das SAMSTAG Werkzeug nehmen wir an, daß eine vollständige SDL Spezifikation der Testarchitektur vorliegt. Hierdurch gewinnt man die Flexibilität, auch für verschiedene Testarchitekturen Testfälle generieren zu können.

Ablauf und Beobachtung. Die Verhaltensbeschreibung eines Protokolls (z.B. in Form einer SDL Spezifikation oder eines MSCs) kann beobachtbare und nicht beobachtbare (systeminterne) Aktionen beinhalten. Aus diesem Grund unterscheiden wir zwischen einem *Ablauf* (engl. trace) und der *Beobachtung* (engl. observation) eines Ablaufs. Ein Ablauf ist eine Folge von beliebigen Aktionen eines Protokolls. Bezogen auf eine SDL Spezifikation kann ein Ablauf aus allen möglichen SDL Aktionen, wie z.B. das Verarbeiten oder Senden eines Signals, das Ausführen einer Task und das Setzen oder Rücksetzen eines Timers bestehen. Zur anschaulichen Darstellung der Abläufe einer SDL Spezifikation werden nachfolgend MSCs benutzt. Wir möchten jedoch darauf hinweisen, daß ein MSC aufgrund seiner Halbordnungssemantik im allgemeinen eine ganze Menge von Abläufen und Beobachtungen repräsentieren kann.

Die Beobachtung eines Ablaufs ist die Folge der beobachtbaren Aktionen des Ablaufs. Eine für das Konformitätstesten relevante Beobachtung kann z.B. nur die Signalsende- und Signalempfangsaktionen der Testprozesse beschreiben. In Abbildung 7 sind ein Ablauf einer SDL Spezifikation und die zugehörige Beobachtung gezeigt.

Testfall. Ein Testfall besteht aus *Preamble, Testbody* und *Postamble.* Die Preamble eines Testfalls beinhaltet alle Aktionen der Testprozesse, um das zu testende System aus seinem Initialzustand[6] in einen Zustand zu bringen, von dem aus der Testbody ausführbar ist. Der Testbody beschreibt den eigentlichen Test, der den Testzweck prüft. Mittels der Postamble wird das System wieder in den Initialzustand gebracht. Ein vollständiger Testfall muß für jede mögliche Aktion des zu testenden Systems eine Reaktion der Testprozesse oder ein abschließendes Testurteil vorsehen. Wir definieren einen Testfall als eine Menge von Beobachtungen, wobei jede Beobachtung zu einem eindeutigen Testurteil führt.

Testurteile. Die möglichen Testurteile sind PASS, INCONCLUSIVE und FAIL. Für ein PASS muß der Testzweck erfüllt sein und sich das System nach dem Test wieder im Initialzustand befinden[7]. Ein FAIL wird vergeben, wenn das Verhalten des zu testenden Systems im Widerspruch zur Spezifikation steht, und ein INCONCLUSIVE wird zugewiesen, wenn die Beobachtung weder ein PASS noch ein FAIL zuläßt.

TTCN. Die *„Tree and Tabular Combined Notation"* (TTCN) ist eine in CTMF standardisierte Sprache, um abstrakte Testrei-

[5] Im allgemeinen Sprachgebrauch bezeichnet die Abkürzung MSC sowohl die MSC Sprache, als auch ein Diagramm in der MSC Sprache. Nachfolgend unterscheiden wir die Sprache von einem Diagramm durch die Verwendung des Begriffs MSC Sprache.

[6] Typischerweise besitzen verteilte Systeme einen Initial- oder Ruhezustand. Als Beispiel sei hier das Telefonsystem genannt. Das Protokoll bei einem Telefongespräch beginnt und endet in einem (Ruhe-)Zustand, bei dem die Telefonhörer bei Anrufer und Angerufenem auf dem Telefongerät liegen.

[7] CTMF/FMCT erlaubt verschiedene Alternativen für die Vergabe von PASS Urteilen. Eine andere als die von uns gewählte Möglichkeit besteht darin, ein PASS Urteil bereits bei der Erfüllung des Testzwecks zu vergeben und keinerlei Forderung an den Systemzustand nach dem Test zu stellen.

hen für Konformitätstests zu beschreiben. Beispiele für TTCN Darstellungen sind in den Abbildungen 10 und 11 gezeigt.

In einer TTCN Tabelle werden die zu einem Testfall gehörenden Beobachtungen in einer Baumnotation beschrieben (siehe Spalte *Behaviour Description* in Abbildung 10). Die Baumstruktur wird durch die Ordnung und die Einrückungstiefe der aufgeführten Aktionen festgelegt. Zwei Aktionen mit der gleichen Einrückungstiefe können alternativ stattfinden (z. B. die Zeilen Nr. 6 und 14 in Abbildung 10), während eine Nachfolgeaktion durch die nächstgrößere Einrückungstiefe beschrieben ist (z. B. die Zeilen Nr. 1 und 2 in Abbildung 10).

Die einzelnen Aktionen werden durch die beteiligte Instanz (UT oder LT in Abbildung 10), durch ihre Art (ein „!" beschreibt eine Sendeaktion und ein „?" bezeichnet eine Empfangsaktion) und durch die gesendeten bzw. empfangenen Protokolldateneinheiten oder Dienstprimitive charakterisiert. So beschreibt z. B. die Aktion „UT!ICONreq" in Zeile 1 von Abbildung 10 das Senden eines ICONreq durch den Upper Tester UT an das zu testende System.

In Abbildung 11 finden sich in den Aktionsbeschreibungen OTHERWISE Konstrukte. Ein OTHERWISE kann als abkürzende Schreibweise angesehen werden. Es erlaubt die Behandlung von beliebigen (korrekten und nicht korrekten) Protokolldateneinheiten oder Dienstprimitiven.

Testurteile werden in die *Verdict* Spalte einer TTCN Tabelle eingetragen. Im TTCN Testfall in Abbildung 10 werden nur die Testurteile PASS und INCONCLUSIVE vergeben. Die FAIL Fälle werden in diesem speziellen Fall durch das in Abbildung 11 abgebildete *Default* Verhalten definiert. Solche Default Beschreibungen müssen im Kopf der TTCN Testfallbeschreibung referenziert werden (siehe *Default* in Abbildung 10).

TTCN besitzt noch eine ganze Reihe weiterer Sprachkonstrukte, die jedoch für das Verständnis der nachfolgenden Abschnitte hier nicht eingeführt werden müssen. Nähere Informationen zu TTCN finden sich z. B. in [2] und [11].

3.2 Die SAMSTAG Methode

Die automatische Generierung eines Testfalls läßt sich auf die Simulation einer Testarchitektur zurückführen, bei der das Verhalten der Testprozesse protokolliert wird. Die Simulation wird durch eine Testfalldefinition (vgl. Abschnitt 3.1) und durch einen Testzweck gesteuert. Die aufgezeichneten Beobachtungen sind die Grundlage für die zu entwickelnde Testfallbeschreibung. Bevor wir näher auf die Testfallgenerierung eingehen, sollen zunächst die Beziehungen zwischen Spezifikation, Testzweck und Testfall erklärt werden.

Eine eigenschaftsorientierte Betrachtung des Testens. Spezifikationen und Implementierungen können als Generatoren für Abläufe angesehen werden. Wir definieren eine Menge von Abläufen als eine Eigenschaft. Eine Implementierung hat genau dann die Eigenschaft einer Spezifikation, wenn die Menge aller Abläufe der Implementierung eine Teilmenge der Menge aller Abläufe der Spezifikation ist.

In [15] beschreiben Manna und Pnueli verschiedene Klassen von Eigenschaften. Sie unterscheiden zwischen *Garantieeigenschaften* (engl. guarantee properties), *Sicherheitseigenschaften* (engl. safety properties) und vier höhere Klassen von Eigenschaften. Informal ausgedrückt besagt eine Garantieeigenschaft, daß in jedem Ablauf irgendwann einmal etwas Erwünschtes passiert[8]. Demgegenüber besagt eine Sicherheitseigenschaft, daß in jedem Ablauf niemals etwas Unerwünschtes passiert. Die höheren Eigenschaftsklassen beschreiben, daß sich in jedem Ablauf etwas Erwünschtes immer wieder, oder ab einem gewissen Zeitpunkt ständig ereignet.

Bei einem Test vergleichen wir die Abläufe einer Spezifikation mit den Abläufen einer Implementierung und versuchen eine Aussage über die Beziehungen der beiden Ablaufmengen zu machen. Im Prinzip macht man Aussagen darüber, welche Eigenschaften der Spezifikation die Implementierung besitzt und welche nicht. Man kann sich nun fragen, bzgl. welcher Eigenschaftsklassen man diese Aussagen machen kann. Wir bezeichnen diese Eigenschaftsklassen als *testbare Eigenschaften*.

In [17] werden nur *Garantie-* und *Sicherheitseigenschaften* als testbare Eigenschaften identifiziert. Informal ausgedrückt ist eine Sicherheitseigenschaft genau deshalb testbar, weil die Implementierung während des Tests etwas *Unerwünschtes* zeigen kann. In so einem Fall wird nachgewiesen, daß die Implementierung die Sicherheitseigenschaft nicht besitzt. Eine Garantieeigenschaft ist genau deshalb testbar, weil die Implementierung während eines endlichen Tests das *Erwünschte* zeigen kann. In so einem Fall wurde die Garantieeigenschaft validiert. Höhere Eigenschaften, in denen gefordert wird, daß sich etwas *Erwünschtes* beliebig oft ereignet, können mit einem endlichen Testfall weder widerlegt noch validiert werden.

Sicherheits- und Garantieeigenschaften im Konformitätstesten nach CTMF/FMCT. Sicherheits- und Garantieeigenschaften finden sich auch im allgemeinen Vorgehen für das Konformitätstesten nach CTMF/FMCT wieder. Das erlaubte Systemverhalten wird durch eine Spezifikation beschrieben. Sie kann daher als Sicherheitseigenschaft interpretiert werden[9]. Ein Testzweck definiert demgegenüber etwas, was während des Tests beobachtet werden soll. Ein Testzweck kann daher als Garantieeigenschaft interpretiert werden.

Ein Testfall ist durch eine Sicherheitseigenschaft (d.h. eine Systemspezifikation) und eine Garantieeigenschaft (d.h. ein Testzweck) bestimmt. Aus der Kombination der Aussagen, die man bzgl. dieser beiden Eigenschaften während eines Tests machen kann, ergeben sich die drei Testurteile PASS, FAIL und INCONCLUSIVE.

- **PASS** wird einer Beobachtung zugewiesen, wenn diese die Garantieeigenschaft eindeutig nachweist und die Sicherheitseigenschaft nicht verletzt.
- **INCONCLUSIVE** wird einer Beobachtung zugewiesen, wenn diese die Garantieeigenschaft nicht nachweist, jedoch die Sicherheitseigenschaft auch nicht verletzt.
- **FAIL** wird einer Beobachtung zugewiesen, wenn diese die Sicherheitseigenschaft verletzt, egal ob die Garantieeigenschaft erfüllt wird oder nicht[10].

[8] Eine Garantieeigenschaft kann dementsprechend auch als Erreichbarkeitskriterium aufgefaßt werden.

[9] Eine Spezifikation kann im allgemeinen auch andere Eigenschaften beschreiben. Würde man temporallogische Formeln als Spezifikationssprache verwenden, so könnte man z. B. Lebendigkeitseigenschaften spezifizieren. Für das Konformitätstesten interessiert uns jedoch nur, daß man eine Spezifikation als Sicherheitseigenschaft interpretieren kann.

Die Darstellung von Sicherheits- und Garantieeigenschaften. Für die Testfallgenerierung werden Möglichkeiten zur Beschreibung und Darstellung von Sicherheits- und Garantieeigenschaften benötigt. Hierfür bieten sich z. B. Petri Netze, Automatenmodelle oder temporallogische Formeln an. Für die SAMSTAG Methode haben wir ein Automatenmodell gewählt. Wir nehmen an, daß uns eine Sicherheitseigenschaft als Labeled Transition System, d.h. als unendlicher Automat, vorliegt. Das Labeled Transition System akzeptiert alle Abläufe, die die Sicherheitseigenschaft nicht verletzen. Dieser Ansatz ist so allgemein, daß Sicherheitseigenschaften z. B. mit den standardisierten Spezifikationssprachen LOTOS, Estelle und SDL [7] beschrieben werden können. Eine Garantieeigenschaft fassen wir als endlichen Automaten auf, der alle Abläufe akzeptiert, die diese Eigenschaft nachweisen. Mit diesem Ansatz können Garantieeigenschaften z. B. mit der MSC Sprache, oder mit temporallogischen Formeln [19] spezifiziert werden.

Ein Überblick über die SAMSTAG Methode. Die SAMSTAG Methode bietet, neben der hier beschriebenen Theorie, Algorithmen an, mit denen man für eine Spezifikation, in Form eines Labeled Transition Systems, und für einen Testzweck, in Form eines endlichen Automaten, Testfälle generieren kann. Hierauf wird im nächsten Abschnitt näher eingegangen. Ein Überblick über die SAMSTAG Methode ist in Abbildung 4 gezeigt. Weitere Informationen zur SAMSTAG Methode finden sich in [6], [16] und [17].

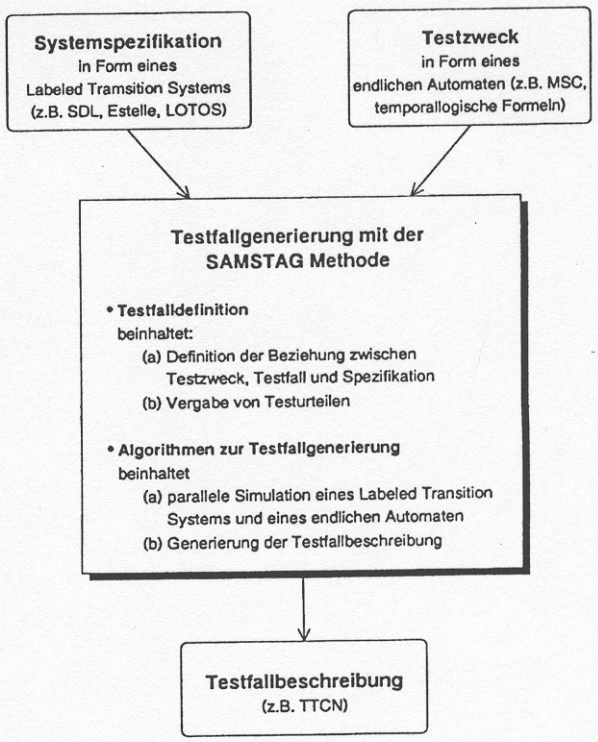


Abb. 4: Die SAMSTAG Methode

3.3 DIE TESTFALLGENERIERUNG MIT DER SAMSTAG METHODE

In diesem Abschnitt wird die Testfallgenerierung mit der SAMSTAG Methode beschrieben. Zur Veranschaulichung benutzen wir hierfür ein einfaches Beispiel.

Uns liegt die Testarchitektur in Abbildung 5 als SDL Spezifikation vor. Getestet wird die Initiator Instanz des Inres Protokolls [8]. Der Testzweck ist durch den MSC in Abbildung 6 vorgegeben. Dieser beschreibt eine spezielle Situation bei einem Verbindungsaufbau. Der Initiator empfängt vom Upper Tester UT ein Connection Request ICONreq und übermittelt ein CR[11] an eine Partnerinstanz, deren Funktion durch einen Lower Tester LT simuliert wird. Sodann wartet der Initiator auf eine Verbindungsbestätigung CC. Trifft das CC nicht innerhalb eines bestimmten Zeitlimits ein, so kann das Senden eines CR bis zu dreimal wiederholt werden. In unserem Fall antwortet der Lower Tester LT nach der Beobachtung des dritten CR. Den Empfang des CC zeigt der Initiator dem Upper Tester UT durch ein Connection Indication ICONind an.

Ein Testfall besteht aus einer endlichen Menge von Beobachtungen, wobei jeder Beobachtung ein Testurteil zugewiesen ist. Die Testurteile sind PASS, FAIL und INCONCLUSIVE. Entsprechend unterscheiden wir zwischen *Pass, Fail* und *Inconclusive Beobachtungen*. Die von uns zur Testdatengenerierung entwickelten Algorithmen basieren auf der Berechnung dieser Beobachtungen. Die Berechnung erfolgt in vier Schritten, die sich auch in der Architektur des SAMSTAG Werkzeugs widerspiegeln (Abbildung 12).

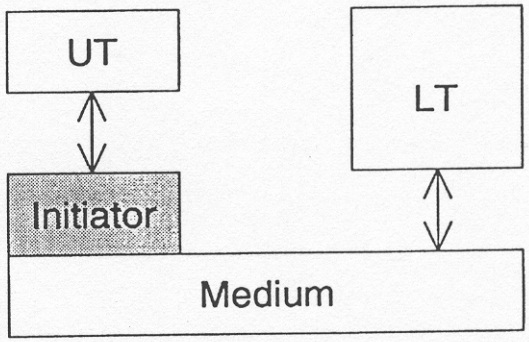


Abb. 5: Eine Testarchitektur für die Initiator Instanz des Inres Protokolls

1. In einem ersten Schritt werden sogenannte potentielle Pass Beobachtungen (engl. possible pass observations) berechnet. Eine potentielle Pass Beobachtung ist eine Beobachtung zu der es einen Ablauf gibt, für den folgende zwei Bedingungen gelten müssen:

   (a) Der Ablauf beginnt und endet im Initialzustand des SDL Systems.

   (b) Der Ablauf führt den durch den MSC geforderten Signalaustausch durch.

   Bei der Berechnung einer potentiellen Pass Beobachtung muß zunächst eine Preamble gefunden werden, die das zu testende System in einen Zustand bringt, von dem aus der MSC beobachtet werden kann. Für unser Beispiel (vgl.

[10] Der Fall des Nachweises der Garantieeigenschaft und der Verletzung der Sicherheitseigenschaft ist zwar im FAIL enthalten, sollte jedoch beim Konformitätstesten nicht vorkommen. Es wäre sinnlos, einen Testzweck (d.h. eine Garantieeigenschaft) nachweisen zu wollen, der aufgrund der Spezifikation (d.h. der Sicherheitseigenschaft) verboten ist.

[11] Da die untere Schnittstelle des Initiators nur über den Medium Dienst getestet werden kann, müssen die Protokolldateneinheiten CR, CC, DT und DR des Inres Protokolls als Parameter der Dienstprimitive des Medium Dienstes gesendet und empfangen werden. Wir abstrahieren von diesen Primitiven, da der Medium Dienst die Protokolldateneinheiten transparent übermittelt und keinen Einfluß auf das Verhalten des Initiators hat.

Abbildung 6) ist die Preamble leer, da der MSC im Initialzustand beginnt. Nach Beobachtung des Testzwecks muß das zu testende System wieder zurück in den Initialzustand

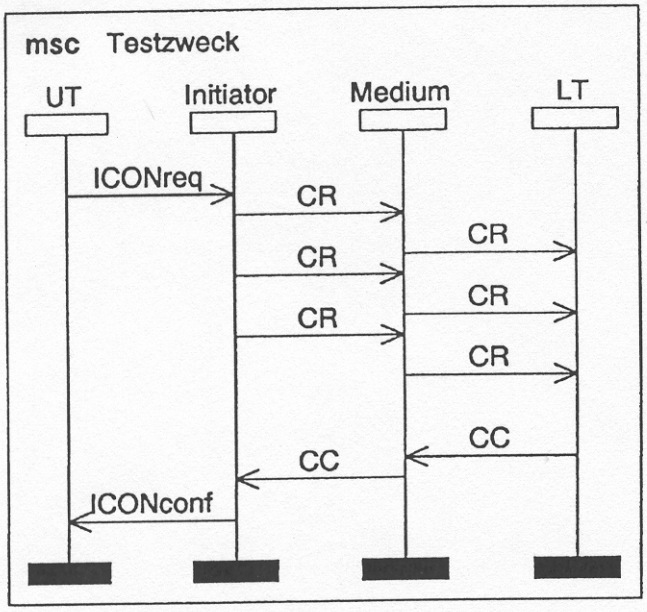


Abb. 6: Verbindungsaufbau nach Empfang der dritten CR Protokolldateneinheit

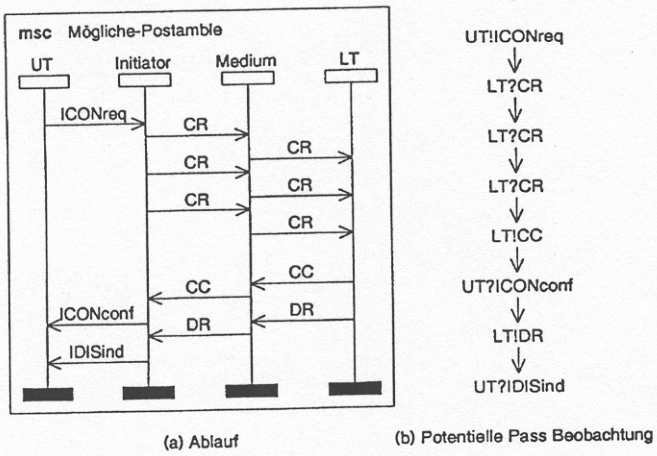


Abb. 7: MSC aus Abbildung 6 mit möglicher Postamble und Beobachtung

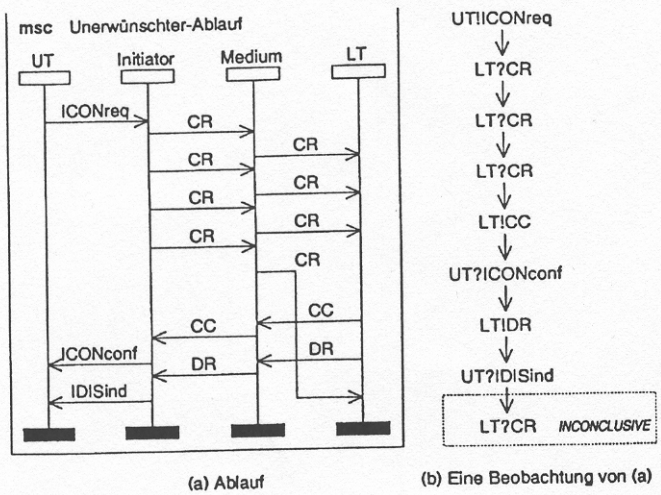


Abb. 8: Unerwünschter Systemablauf

geführt werden. Eine mögliche Postamble ist ein normaler Verbindungsabbau (vgl. Abbildung 7), der vom Lower Tester LT durch ein Disconnection Request DR eingeleitet und vom Initiator dem Upper Tester UT durch ein Disconnection Indication IDISind angezeigt wird. Die Beobachtung des beschriebenen Ablaufs ist eine potentielle Pass Beobachtung.

Im allgemeinen existiert kein eindeutiger Zusammenhang zwischen einem Ablauf und einer Beobachtung. Mehrere Abläufe können die gleiche Beobachtung besitzen. Der MSC in Abbildung 8 repräsentiert mehrere Abläufe und Beobachtungen. Die in Abbildung 8 (b) dargestellte Beobachtung ist bis auf das abschließende CR mit der Beobachtung in Abbildung 7 (b) identisch.
Die SDL Semantik erlaubt keine Annahmen darüber, wie lange man nach dem Empfang eines IDISind auf ein evtl. noch ausstehendes CR warten muß. Aufgrund der Beobachtung von IDISind kann daher nicht garantiert werden, daß der Testzweck in Abbildung 6 durchgeführt worden ist. Wird nach dem IDISind ein viertes CR beobachtet, muß das Testurteil INCONCLUSIVE vergeben werden (vgl. Abbildung 8 (b)).

2. In einem zweiten Schritt wird deshalb geprüft, ob die gefundenen potentiellen Pass Beobachtungen eindeutig sind, d.h. daß für alle Abläufe einer potentiellen Pass Beobachtung die Bedingungen (a) und (b) in 1. gelten. In diesem Fall bezeichnen wir eine potentielle Pass Beobachtung als *eindeutig* oder als *eindeutige Pass Beobachtung* (engl. unique pass observation). Im allgemeinen kann es für einen Testfall mehrere eindeutige Pass Beobachtungen geben. Als Pass Beobachtung des zu generierenden Testfalls wählen wir eine Teilmenge der kürzesten eindeutigen Pass Beobachtungen aus.

   Der Ablauf, der für unser Beispiel zu einer eindeutigen Pass Beobachtung führt, ist in Abbildung 9 gezeigt. Anstatt eines normalen Verbindungsabbaus wird vom Upper Tester UT ein Datentransfer mit einem IDATreq eingeleitet. Der Initiator sendet dem Lower Tester ein DT und wartet auf eine Empfangsbestätigung. Da der Lower Tester nicht antwortet, wird das Senden von DT dreimal wiederholt. Danach zeigt der Initiator den mißglückten Datentransfer mit einem IDISind an und kehrt in den Initialzustand zurück.

   Wir wählen die Beobachtung des beschriebenen Ablaufs als die Pass Beobachtung des zu generierenden Testfalls aus (vgl. Abbildung 9 (b)). Daraus ergibt sich der TTCN Testfall in Abbildung 10. Die Pass Beobachtung ist in den Zeilen 1 bis 12 zu sehen.

3. In einem dritten Schritt werden für die gewählten Pass Beobachtungen die zugehörigen *Inconclusive Beobachtungen* berechnet. Eine Inconclusive Beobachtung ist eine Beobachtung, das mit einer Pass Beobachtung einen gewissen Zeitraum übereinstimmt, jedoch mit einer zur Pass Beobachtung unterschiedlichen Ausgabe des SDL Systems endet, die jedoch erlaubt ist. Die Inconclusive Beobachtungen für unser Beispiel sind in der TTCN Beschreibung in Abbildung 10 enthalten.
4. Im vierten und letzten Schritt werden die *Fail Beobachtungen* definiert. Fail Beobachtungen werden nicht berechnet, da sie in TTCN mittels einer Default Beschreibung definiert werden können. Das Default für unser Beispiel findet sich in Abbildung 11.

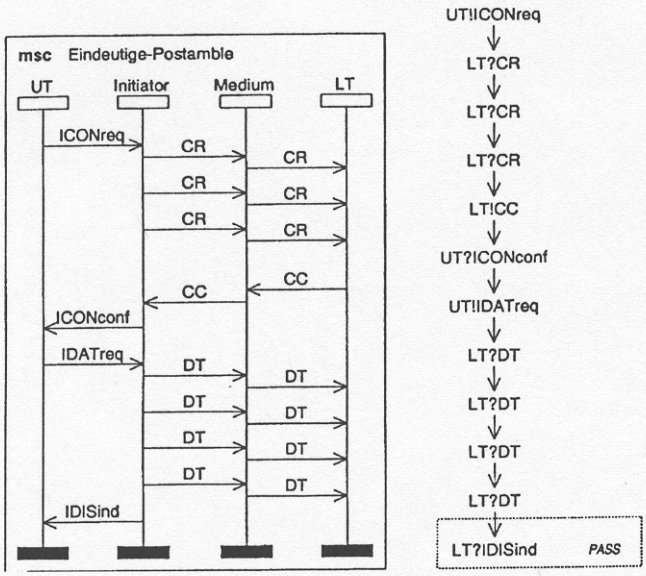


Abb. 9: MSC aus Abbildung 6 mit eindeutiger Postamble

3.4 Das SAMSTAG Werkzeug

Das SAMSTAG Werkzeug realisiert die SAMSTAG Methode für in SDL geschriebene Spezifikationen und für mit MSCs definierte Testzwecke. Die Werkzeugarchitektur ist in Abbildung 12 dargestellt. Das SAMSTAG Werkzeug besteht aus einem MSC Simulationswerkzeug, einem SDL Simulationswerkzeug und einem Testfallgenerator. Die Front- und Backends des SAMSTAG Werkzeugs sind kommerzielle SDL, MSC und TTCN Editoren.
Das MSC Simulationswerkzeug besteht aus einem Transformator, der aus einem MSC eine Datenstruktur generiert, die bei der Testfallgenerierung vom MSC Interpreter interpretiert wird. Aus Gründen der Performance wurde das SDL Simulationswerkzeug anders implementiert. Es besteht aus einem Transformator, der aus einer SDL Spezifikation ein ausführbares C++-Programm, den eigentlichen SDL Simulator, generiert. Der Testfallgenerator steuert den SDL Simulator und den MSC Interpreter. Bei der Testfallgenerierung berechnet er die potentiellen Pass Beobachtungen, die eindeutigen Pass Beobachtungen und die Inconclusive Beobachtungen. Abschließend definiert er die Fail Beobachtungen und gibt den generierten Testfall in der TTCN Notation aus.

Berechnung von potentiellen Pass Beobachtungen. Die Berechnung von potentiellen Pass Beobachtungen ist ein typisches Suchproblem. Das SAMSTAG Werkzeug sucht die Abläufe eines SDL Systems, die im Initialzustand beginnen und enden und zusätzlich den Signalaustausch des MSC durchführen. Die zu diesen Abläufen gehörenden Beobachtungen sind die potentiellen Pass Beobachtungen. Das Auffinden von potentiellen Pass Beobachtungen ist für beliebige SDL Systeme unentscheidbar, da dieses Problem äquivalent zum Halteproblem für Turingmaschinen ist. Das SAMSTAG Werkzeug bricht daher eine ergebnislose Suche nach Ablauf einer gewissen Zeitschranke ab.

Es gibt unterschiedliche Suchmethoden, wie z.B. Tiefensuche oder Breitensuche. Eine Breitensuche kann nicht angewendet werden, da es unmöglich ist, alle erreichten Zustände abzuspeichern. Eine Tiefensuche kann nicht angewendet werden, da eine Terminierung der Suche nicht gewährleistet ist. Aus diesem Grund verwendet das SAMSTAG Werkzeug eine *k-beschränkte Tiefensuche*, die alle Abläufe der Länge k berechnet. Wird kein geforderter Ablauf gefunden, so kann die Suche abgebrochen oder mit einer größeren Schranke wiederholt werden.

Berechnung von eindeutigen Pass Beobachtungen. Für jede potentielle Pass Beobachtung wird geprüft, ob sie eine eindeutige Pass Beobachtung ist. Dabei werden alle Abläufe des SDL Systems simuliert, welche die potentielle Pass Beobachtung als Beobachtung besitzen. Wenn Abläufe existieren, die nicht die Bedingungen (a) und (b) auf Seite 222 erfüllen, so ist die potentielle Pass Beobachtung nicht eindeutig. Werden mehrere eindeutige Pass Beobachtungen gefunden, so wählen wir eine Teilmenge der kürzesten eindeutigen Pass Beobachtungen als Pass Beobachtungen des Testfalls aus. Die Existenz von eindeutigen Pass Beobachtungen kann nicht garantiert werden. Unter Umständen existieren zwar potentielle, jedoch keine eindeutigen Pass Beobachtungen.

Berechnung von Inconclusive Beobachtungen. Für die gewählten Pass Beobachtungen werden die Inconclusive Beobachtungen berechnet. Hierbei werden mittels einer Tiefensuche erneut alle Abläufe simuliert, deren Beobachtung bis zur vorletzten Aktion einer Pass Beobachtung entspricht, dann aber in einer unterschiedlichen Ausgabe des Systems enden.

Definition der Fail Beobachtungen und Ausgabe des TTCN Testfalles. Abschließend werden das Default für die

Test Case Dynamic Behaviour				
Test Case Name : Test_Fall_Beispiel Group : Purpose : Verbindungsaufbau nach Empfang der dritten CR Protokolldateneinheit Default : Unerwartete_Ereignisse Comments :				
Nr	Label	Behaviour Description	Constraints Ref	Verdict
1		UT!ICONreq		
2		LT?CR		
3		LT?CR		
4		LT?CR		
5		LT!CC		
6		UT?ICONconf		
7		UT!IDATreq		
8		LT?DT		
9		LT?DT		
10		LT?DT		
11		LT?DT		
12		UT?IDISind		PASS
13		LT?CR		INCON
14		LT?CR		INCON
Detailed Comments:				

Abb. 10: Ein TTCN Testfall

Default Dynamic Behaviour				
Default Name : Test_Fall_Beispiel Group : Objective : Behandlund nicht erwarteter Ereignisse Comments :				
Nr	Label	Behaviour Description	Constraints Ref	Verdict
1		UT?OTHERWISE		FAIL
2		LT?OTHERWISE		FAIL
Detailed Comments:				

Abb. 11: TTCN Default für den TTCN Testfall in Abbildung 10

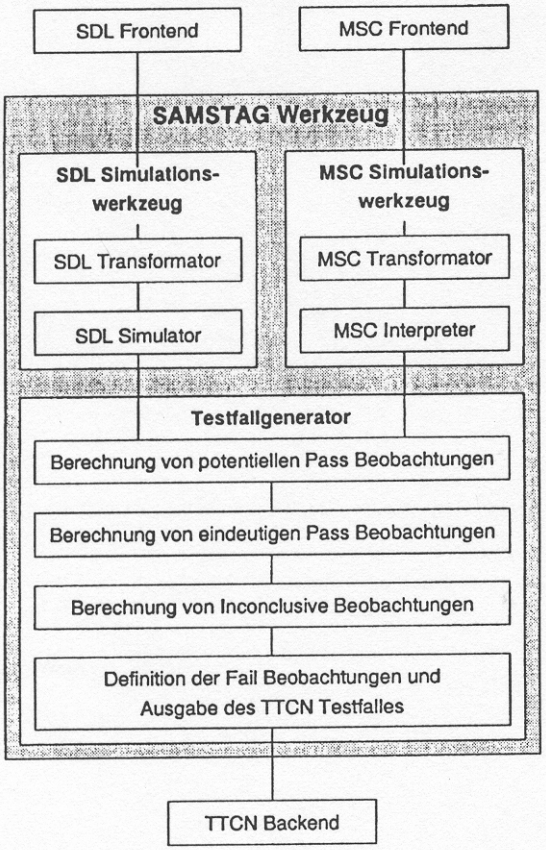


Abb. 12: Die Architektur des SAMSTAG-Werkzeugs

Fail Beobachtungen erzeugt, sowie die Pass und Inconclusive Beobachtungen in die TTCN Notation transformiert. Als Ergebnis wird die TTCN Beschreibung des generierten Testfalls als ASCII Datei ausgegeben, die mit kommerziell verfügbaren TTCN Werkzeugen weiterbearbeitet werden kann.

4 ZUSAMMENFASSUNG UND AUSBLICK

Mit der SAMSTAG Methode können, basierend auf einer formalen Protokollspezifikation und einer Menge von Testzwecken, Testfälle für Konformitätstests generiert werden. Hierfür mußte insbesondere der für das Konformiätstesten wichtige Begriff des Testzwecks formalisiert werden. In Kapitel 2 sind zudem das Konformitätstesten nach CTMF/FMCT und theoretischen Methoden zur Testfallgenerierung diskutiert und miteinander verglichen worden.

Die SAMSTAG Methode, CTMF/FMCT und die theoretischen Methoden formalisieren unterschiedliche Schritte beim Konformitätstesten. Der Beitrag, den die drei Ansätze bei einer vollständigen formalen Beschreibung des Konformitätstestens leisten können, ist als Überblick in Abbildung 3 dargestellt. Die Abbildung zeigt auch, daß die theoretischen Grundlagen für die Bestimmung der Testzwecke aus einer Protokollspezifikation bisher noch nicht erarbeitet worden sind. Wir meinen jedoch, daß man diesen Schritt, mit Hilfe der in den theoretischen Methoden verwendeten Überdeckungskriterien, formalisieren könnte. Für die praktische Anwendbarkeit eines vollständigen formalen Modells für das Konformitätstesten müßten zusätzlich noch die Begriffe Überdeckungskriterium und Konformitätsrelation näher untersucht und verallgemeinert werden.

Danksagungen

Die hier vorgestellte Arbeit wurde im Rahmen des von der Schweizer PTT finanzierten F & E Projektes Conformance Testing - Ein Werkzeug zur Generierung von Testfällen, Vertragsnummer 233/257, durchgeführt. In diesem Projekt haben neben uns auch Matthias Günter, Peter Gurtner, Kurt Neuenschwander und Daniel Toggweiler mitgearbeitet. Wir danken allen Beteiligten für zahlreiche Diskussionen, Anregungen und Kommentare. Die Ausarbeitung dieses Artikels wurde zum Teil durch das KWF Projekt Nr. 2555.1 „Graphische Methoden im Testprozeß" unterstützt.

Literatur

[1] Alcatel Network Systems. Alcatel 8650 - Conformance Test System - GSM. Produkt Information, Alcatel STR AG (Zürich), 1993.

[2] B. Baumgarten, A. Giessler. OSI-Testmethodik und TTCN. Berichte der Gesellschaft für Mathematik und Datenverarbeitung Nr. 202. R. Oldenburg Verlag, 1992.

[3] F. Belina, D. Hogrefe, A. Sarma. SDL with Applications from Protocol Specification. Carl Hanser Verlag and Prentice Hall International, 1991.

[4] R. Braek, O. Haugen. Engineering Real Time Systems - An Object-Oriented Methodology using SDL. Prentice Hall, Hemel Hempstead, UK, 1993.

[5] J. Grabowski, P. Graubmann, E. Rudolph. The Standardization of Message Sequence Charts. In Proceedings of the IEEE Software Engineering Standards Symposium 1993, September 1993.

[6] J. Grabowski, D. Hogrefe, R. Nahm. Test Case Generation with Test Purpose Specification by MSCs. In SDL'93 - Using Objects. North-Holland, Oktober 1993.

[7] D. Hogrefe. Estelle, LOTOS und SDL - Standard Spezifikationssprachen für verteilte Systeme. Springer Verlag, 1989.

[8] D. Hogrefe. OSI Formal Specification Case Study: The Inres Protocol and Service (revised). Technischer Bericht IAM-91-012, Universität Bern, Institut für Informatik, Mai 1991, Neue Version Mai 1992.

[9] D. Hogrefe. On the Development of a Standard for Conformance Testing based on Formal Specifications. Computer Standards Interfaces 14, 1992.

[10] G. J. Holzmann. Design and Validation of Computer Protocols. Prentice-Hall International, Inc., 1991.

[11] ISO/IEC JTC 1/SC 21. Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework. International Multipart Standard 9646, ISO/IEC, 1992.

[12] ISO/IEC JTC 1/SC 21. Open Systems Interconnection - Formal Methods in Conformance Testing. Working Draft, ISO/IEC, Juli 1993.

[13] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.100: Specification and Description Language (SDL) (formerly CCITT Recommendation Z.100). ITU, Genf, Juni 1992.

[14] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.120: Message Sequence Chart (MSC). ITU, Genf, Juni 1992.

[15] Z. Manna, A. Pnueli. A Hierarchy of Temporal Properties. In Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing. ACM Press, 1990.

[16] R. Nahm. Conformance Testing Based on Formal Description Techniques and Message Sequence Charts. Dissertation, Universität Bern, Institut für Informatik, Februar 1994.

[17] R. Nahm, J. Grabowski, D. Hogrefe. Test Case Generation for Temporal Properties. Technischer Bericht IAM-93-013, Universität Bern, Institut für Informatik, Juni 1993.

[18] Siemens AG. Produkt Information K1197, K1103. Siemens AG Berlin, 1993.

[19] P. Wolper. On the Relations of Programs and Computations to Models of Temporal Logic. In Proceedings Temporal Logic in Specification, Lecture Notes in Computer Science 398, 1989.