Die SAMSTAG Methode und ihre Rolle im OSI Konformitätstesten

Jens Grabowski, Robert Nahm, Andreas Spichiger, Dieter Hogrefe

Institut für Informatik, Universität Bern
Länggassstr. 51, CH-3012 Bern

IAM-93-024

Oktober 1993

Zusammenfassung[1]

Für das Konformitätstesten von OSI Protokollen 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 sie mit diesen Methoden nicht mehr testbar sind. Zum anderen hat man sich bei der Methodenentwicklung nur wenig an dem in der Praxis üblichen Vorgehen beim OSI Konformitätstesten orientiert. Dieses Vorgehen ist insbesondere im internationalen ISO/IEC Standard 9646 (ISO/IEC IS 9646) [10] 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 ISO/IEC IS 9646. In diesem Artikel werden die angesprochenen Probleme ausführlich diskutiert und die SAMSTAG Methode eingeführt. Zusätzlich zeigen wir, wie sich ISO/IEC IS 9646, die in wissenschaftlichen Institutionen entwickelten Methoden und die SAMSTAG Methode bei einer vollständigen formalen Erklärung des OSI Konformitätstestens ergänzen können.

CR Categories and Subject Descriptors: C.2.0 [Computer-Communication Networks]: General; C.2.2 [Computer-Communication Networks]: Network Protocols; D.2.5 [Software Engineering]: Testing and Debugging

[1]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.

1 Einleitung

Vor ungefähr 10 Jahren haben ISO/IEC und ITU-TS[2] angefangen, sich mit dem Konformitätstesten von OSI Protokollen zu beschäftigen. Ein Resultat dieser Arbeiten ist der internationale ISO/IEC Standard 9646 (ISO/IEC IS 9646) *'OSI Conformance Testing Methodology and Framework'* [10][3]. Dieser siebenteilige Standard beschreibt ein allgemeines Vorgehen, um eine Protokollimplementierung auf seine Konformität zu einer OSI Protokollspezifikation zu überprüfen. ISO/IEC IS 9646 beinhaltet u.a. die generellen Konzepte für das Konformitätstesten, verschiedene Testarchitekturen und Testmethoden, die *'Tree and Tabular Combined Notation'* (TTCN) [11] als Beschreibungssprache für Testfälle und die Testrealisierung. ISO/IEC IS 9646 scheint auch eine gute Grundlage für das in der industriellen Praxis durchgeführte Protokolltesten zu sein. Die hohe Akzeptanz und die zunehmende Verbreitung von ISO/IEC IS 9646 zeigt sich an dem wachsenden Angebot von kommerziellen Werkzeugen, die einzelne Teile des Vorgehens beim OSI Konformitätstesten unterstützen und automatisieren [15, 1].

Parallel zur Entwicklung von ISO/IEC IS 9646 haben ISO/IEC und ITU-TS die Spezifikationssprachen SDL, Estelle, LOTOS [7] und MSC [17] entwickelt. Durch die Anwendung dieser Sprachen in den Standards sollen unter anderem die Validierung und das Konformitätstesten von OSI Protokollen unterstützt werden.

Die Einführung formaler Methoden zur Protokollbeschreibung hat dazu geführt, dass man sich auch 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 [9]. 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 von realen Systemen und auf ISO/IEC IS 9646 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 Konformitätsdefinition 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 ISO/IEC IS 9646 abbilden lassen. Zusammenfassend muss daher leider gesagt werden, dass die theoretischen Methoden das in ISO/IEC IS 9646 beschriebene Vorgehen zwar befruchtet, jedoch bisher nur wenig vereinfacht haben.

Es besteht jedoch die Möglichkeit, ISO/IEC IS 9646 und die theoretischen Methoden näher zusammenzubringen. Eine wichtige Rolle könnten hierbei die Ergebnisse des Projekts *'Conformance Testing - Ein Werkzeug zur Generierung von Testfällen'* spielen. Das erwähnte Projekt hat sich eng an ISO/IEC IS 9646 orientiert. Ein Hauptziel ist es, einen wichtigen Schritt im allgemeinen Vorgehen beim OSI Konformitätstesten 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 ISO/IEC IS 9646 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

[2] Der *'Telecommunication Standards Sector of the International Telecommunication Union'* (ITU-TS) ist die Nachfolgeorganisation des CCITT. Das CCITT wurde im März 1993 aufgelöst.

[3] ISO/IEC IS 9646 wird von der ITU-TS als Empfehlung X.290 herausgegeben.

wurde die SAMSTAG Methode darüberhinaus in Form eines Prototyps implementiert.

In diesem Artikel vergleichen wir ISO/IEC IS 9646 mit den theoretischen Methoden (Abschnitt 2) und gehen dabei insbesondere auf die Gemeinsamkeiten und die Unterschiede ein. Anschliessend werden die SAMSTAG Methode und das SAMSTAG Werkzeug vorgestellt (Abschnitt 3). Die SAMSTAG Methode wird in das allgemeine Vorgehen beim OSI Konformitätstesten eingeordnet und die Testfallgenerierung anhand eines kleinen Beispiels erläutert. Im letzten Kapitel werden die formalen Aspekte von ISO/IEC IS 9646, den theoretischen Methoden und der SAMSTAG Methode noch einmal zusammengefasst (Abschnitt 4). Hieraus ergibt sich die Möglichkeit einer vollständigen formalen Erklärung des OSI Konformitätstestens.

2 Ein Vergleich von ISO/IEC IS 9646 mit den theoretischen Methoden

In diesem Kapitel wird zunächst das in ISO/IEC IS 9646 standardisierte Vorgehen für das OSI 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 OSI Konformitätstesten

In Abbildung 1 ist das allgemeine Vorgehen dargestellt, um nach ISO/IEC IS 9646 für eine OSI Protokollspezifikation[4] und eine Protokollimplementierung über einen Konformitätstest zu einer Konformitätsaussage zu gelangen.[5] Die Rechtecke bezeichnen Aktionen, während die Ellipsen Vor- und Nachbedingungen (oder Ein- und Ausgaben) der Aktionen beschreiben. Gepunktete Rechtecke und Ellipsen bezeichnen Aktionen und Bedingungen, die in ISO/IEC IS 9646 nur informal beschrieben sind. Zahlen und Buchstaben dienen als Referenzen für die nachfolgende Beschreibung.

Das Ziel des Konformitätstestens ist es nachzuweisen, dass eine Protokollimplementierung (a) das in einer Spezifikation (b) beschriebene Verhalten besitzt. Die Implementierung wird jedoch nicht direkt gegen die Spezifikation verifiziert, sondern mit einer Menge von Testfällen, einer *Testreihe*, getestet.

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 Testfällen*, erzeugt. Die Testzwecke (c), die Bestimmung der Testzwecke (1) und auch die Spezifikation der abstrakten Testreihe (2) sind in ISO/IEC IS 9646 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 Protokollda-

[4] Der hier verwendete Begriff der *OSI Protokollspezifikation* bezeichnet im allgemeinen einen ISO, ITU-TS oder ETSI Standard. Wir möchten desweiteren darauf hinweisen, das die Begriffe OSI Protokollspezifikation, Protokollspezifikation und Spezifikation in diesem Artikel synonym verwendet werden.

[5] Aufgrund der Komplexität des in ISO/IEC IS 9646 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.

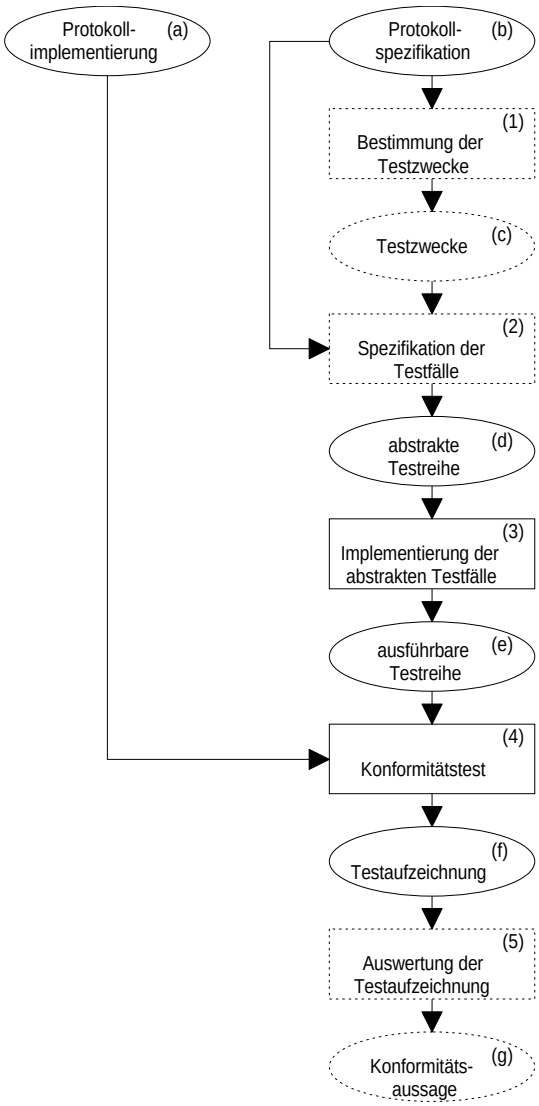


Abbildung 1: Das Vorgehen beim OSI Konformitätstesten

teneinheiten 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 ISO/IEC IS 9646 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 Bestim-

mung 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 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 von zwei Beispielen erläutert. Beispiel A bezieht sich auf die von Holzmann in [9] diskutierten Methoden zur Testfallgenerierung für endliche Automaten (z.B. UIO- oder W-Methode). In Beispiel B wird auf das von Tretmans in [18] beschriebene Verfahren zur Testfallgenerierung für LOTOS Spezifikationen eingegangen. Die geklammerten Buchstaben und Zahlen im nachfolgenden Text sind Referenzen auf Abbildung 2.

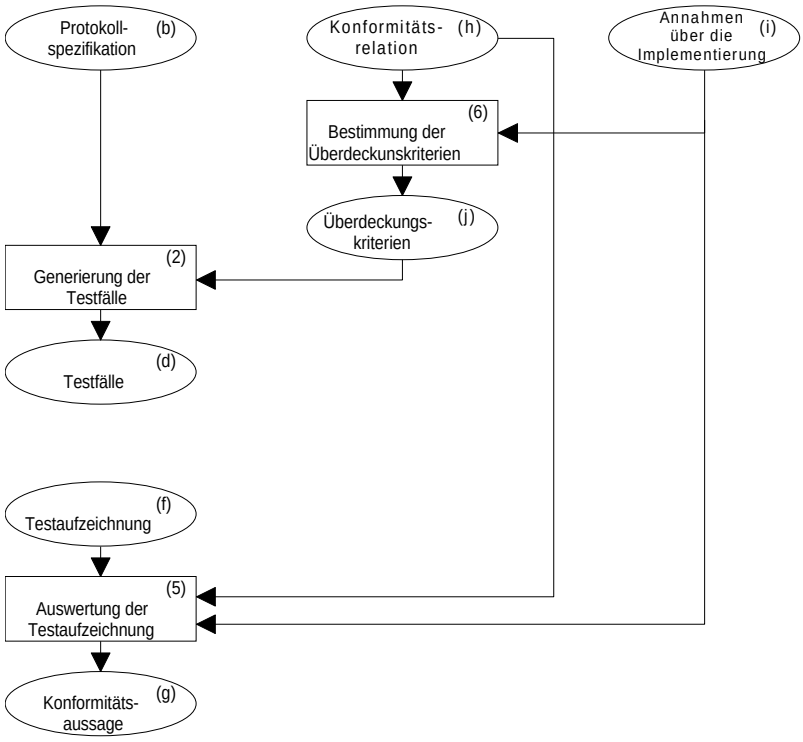


Abbildung 2: Testfallgenerierung mit theoretischen Methoden

Grundsätzlich wird bei allen theoretischen Methoden zuerst Konformitätsrelation (h) festgelegt. Sie sagt etwas über die Beziehung der Traces von Spezifikation und Implementierung aus. Die Gültigkeit der Relation soll mit einem Test nachgewiesen werden. Verschiedene Konformitätsrelationen sind jedoch nicht

für beliebige Spezifikationen und Implementierungen testbar. Eine Verhaltensäquivalenz von Spezifikation und Implementierung kann z.B. nur dann in endlicher Zeit nachgewiesen werden, wenn sich ihre Verhalten durch endliche deterministische Automaten beschreiben lassen. Durch die Konformitätsrelation wird die Menge der testbaren Spezifikationen und Implementierungen eingeschränkt.

Beim Konformitätstesten liegt die Spezifikation (b) vor. Man kann daher direkt überprüfen, ob sie in der Klasse der für eine Konformitätsrelation testbaren Spezifikationen liegt. Eine Implementierung liegt aber im allgemeinen als *Black Box* vor. Es kann daher nicht überprüft werden, ob die Spezifikation testbar ist. Statt dessen macht man Annahmen (i) über die Implementierung. Im Prinzip wird auf diese Weise die abschliessende Konformitätsaussage (g) relativiert. Zu unseren Beispielen:

Beispiel A. Bei den von Holzmann diskutierten theoretischen Methoden soll eine Verhaltensäquivalenz zwischen einer Spezifikation und einer Implementierung nachgewiesen werden. Zusätzlich wird vorausgesetzt, dass zum Testen nur eine endliche Zeit zur Verfügung steht. Zum Nachweis der Verhaltensäquivalenz müssen sich Spezifikation und Implementierung durch endliche, stark zusammenhängende und deterministische Automaten beschreiben lassen.

Beispiel B. Auch Tretmans will eine Verhaltensäquivalenz zwischen einer Implementierung und einer Spezifikation nachweisen. Er macht jedoch keine Annahme über eine zeitliche Beschränkung des Testvorgangs. Als Konsequenz kann Tretmans das Testen von Implementierungen und Spezifikationen behandeln, deren Verhalten sich durch Labeled Transition Systeme, d.h. durch unendliche Automaten, beschreiben lassen.[6]

Basierend auf der Konformitätsrelation (i) und den einschränkenden Annahmen wird ein Überdeckungskriterium (j) bestimmt. Dieses Kriterium definiert, wie die Spezifikation durch die Testfälle überdeckt werden soll. Ein Überdeckungskriterium könnte z.B. fordern, dass jeder Zustand der Spezifikation mindestens einmal erreicht wird. Ein Kriterium könnte z.B. sagen, dass alle Zustandsübergänge einmal durchgeführt werden sollen. Basierend auf dem Überdeckungskriterium und der Spezifikation werden die Testfälle (d) entwickelt. Der für das OSI Konformitätstesten so wichtige Begriff Testzweck (vgl. Abbildung 1 (c)) ist in den theoretischen Methoden nicht zu finden.

Beispiel A. Zum Testen der Verhaltensäquivalenz von zwei endlichen Automaten muss jeder Zustandsübergang überprüft. Es wird jedoch nicht für jeden Zustandsübergang ein eigener Testfall generiert. Stattdessen wird mittels einer sog. *Transition Tour* ein einziger grosser Testfall erzeugt, der alle Zustandsübergänge auf einmal testet.

Beispiel B. Auch bei der von Tretmans verwendeten Methode muss jeder Zustandsübergang getestet werden. Es werden jedoch keine Testfälle entwickelt, sondern ein sog. *Canonical Tester*. Der Canonical Tester ist ein Labeled Transition System, das die Umgebung der Spezifikation simuliert. Informal ausgedrückt ist ein Canonical Tester das Inverse einer Spezifikation. Man kann ihn sich aber auch als einen unendlich langen, Testfall vorstellen.

Die Durchführung der Konformitätstests wird in den theoretischen Methoden nicht behandelt. 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) kann eine abschliessende Konformitätsaussage (g) gemacht werden. Die Konformitätsaussage sagt, ob die Konformitätsrelation nachgewiesen werden konnte, oder nicht.

[6] Es sei hier angemerkt, dass Tretmans explizit LOTOS [7] als Spezifiaktionssprache benutzt.

Beispiel A. Wurden der mit der Transition Tour generierte Testfall erfolgreich durchgeführt, so ist die Konformitätsrelation nachgewiesen. Diese Aussage gilt jedoch nur, wenn die vor der Testfallgenerierung gemachten Annahmen von der Implementierung erfüllt sind. Im anderen Fall kann die Konformität nicht garantiert werden.

Beispiel B. Für die Testdurchführung werden Canonical Tester und Implementierung verbunden. Es ergibt sich ein geschlossenes System, d.h. es findet nur interne Kommunikation statt. Falls dieses System nicht terminiert, wurde die Konformitätsrelation nachgewiesen. Ended das System in einem Deadlock, so sind Implementierung und Spezifikation nicht konform.

2.3 Eine vergleichende Zusammenfassung

Durch ISO/IEC IS 9646 wird das gesamte Vorgehen beim OSI Konformitätstesten, von der Testfallentwicklung bis hin zur Konformitätsaussage, abgedeckt. Die einzelnen Vorgehensschritte sind jedoch unterschiedlich formal beschrieben und damit auch unterschiedlich schwer zu automatisieren. Hinreichend formal, d.h. automatisierbar, ist die Testdurchführung (vgl. Abbildung 1), 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 sind. Dieses ist aus mehreren Gründen problematisch.

Die Entwicklung der abstrakten Testreihe basiert auf zwei informalen Schritten, die meist von menschlichen Experten per Hand durchgeführt werden. Bei der Testreihenwicklung wird man versuchen die Funktionen des Protokolls möglichst vollständig zu testen. Wie gut dieses gelingt, hängt stark von der Bestimmung der 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 (vgl. Abbildung 2) 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 Konformiätsrelation 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 häufig 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 das 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. Andersherum erhalten Entwickler über einen Testzweck auch Hinweise darüber, welche Protokollfunktionen fehlerhaft implementiert worden sind.

3 Die SAMSTAG Methode und das SAMSTAG Werkzeug

Im vorigen Abschnitt wurde gezeigt, dass sich die im wissenschaftlichen Umfeld entwickelten Methoden zur Testfallgenerierung nur unvollständig auf das in ISO/IEC IS 9646 standardisierte Vorgehen für das OSI Konformitätstesten abbilden lassen. Insbesondere wird der für das Konformitätstesten zentrale Begriff des Testzwecks nicht erklärt. Die von uns entwickelte SAMSTAG Methode orientiert sich an dem in ISO/IEC IS 9646 beschriebenen Vorgehen und formalisiert insbesondere den Begriff des Testzwecks.

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 [16, 2] und Message Sequence Charts (MSCs) [17, 3] 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. Dieses Projektziel lässt sich in das allgemeine Vorgehen beim OSI Konformitätstesten einordnen (vgl. Abschnitt 2.1). Bezogen auf Abbildung 1 wird die Aktion 2, die Erzeugung der abstrakten Testreihe, automatisiert.

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 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 sind weitgehend ISO/IEC IS 9646 [10] und den ITU-TS Empfehlungen Z.100 [16] und Z.120 [17] entnommen. Einzig die Begriffe *Trace* und *Observable* werden zur Vereinfachung der Darstellungen in den nachfolgenden Abschnitten neu eingeführt.

SDL. Die *'Functional Specification and Description Language'* (SDL) ist eine von der ITU-TS standardisierte Spezifikationssprache für verteilte Systeme und insbesondere für Telekommunikationssysteme [16]. Eine SDL Spezifikation beschreibt eine Menge von endlichen erweiterten Automaten, die mittels Signalaustausch asynchron miteinander kommunizieren können. 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-TS standardisierte Sprache [17]. Mit einem MSC[7] lässt sich ein Ablauf eines verteilten Systems visuell und intuitiv beschreiben (vgl. Abbildung 5). 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 [4].

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 [3].

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 ISO/IEC IS 9646 sind verschiedene Architekturen für Konformitätstests beschrieben. In Abbildung 4 ist eine Testarchitektur für den Test der Initiator Instanz des Inres Protokolls [8] dargestellt. Die Testprozesse sind der Upper Tester UT und der Lower Tester LT. In ISO/IEC IS 9646 wird davon ausgegangen, dass bei einer zu testenden Protokollinstanz die Schnittstelle zur nächsttieferen Protokollschicht nicht frei zugänglich ist. Deshalb muss diese Schnittstelle über den Dienst der tieferen Protokollschicht beobachtet und kontrolliert werden. Auch in Abbildung 4 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.

Trace und Observable. 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 *Trace* und dem *Observable* eines Traces. Ein Trace ist eine Folge von beliebigen Aktionen eines Protokolls. Bezogen auf eine SDL Spezifikation kann ein Trace aus allen möglichen SDL Aktionen, wie z.B. INPUT, OUTPUT, TASK, DECISION, SET oder RESET, bestehen. Zur anschaulichen Darstellung der Traces einer SDL Spezifikation werden nachfolgend MSCs benutzt. Wir möchten jedoch darauf hingewiesen, dass ein MSC aufgrund seiner Halbordnungssemantik im allgemeinen eine ganze Menge von Traces und Observables repräsentieren kann.

Das Observable eines Traces ist die Folge der beobachtbaren Aktionen des Traces. Ein für das Konformitätstesten relevantes Observable könnte z.B. nur die Signalsende- und Signalempfangsaktionen der Testprozesse beschreiben. In Abbildung 6 sind ein Trace einer SDL Spezifikation und das zugehörige Observable 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[8] in einen Zustand zu bringen, von dem aus der Testbody ausführbar ist. Der Testbody beschreibt den eigentlichen Test und 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

[7] 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*.

[8] 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.

abschliessendes Testurteil vorsehen. Wir definieren einen Testfall als eine Menge von Observables, wobei jedes Observable 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[9]. 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 ISO/IEC IS 9646 [11] standardisierte Sprache, um abstrakte Testreihen für Konformitätstests zu beschreiben. Beispiele für TTCN Darstellungen sind in den Abbildungen 9 und 10 gezeigt.

In einer TTCN Tabelle werden die zu einem Testfall gehörenden Observables in einer Baumnotation beschrieben (siehe Spalte *Behaviour Description* in Abbildung 9). 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 9), während eine Nachfolgeaktion durch die nächstgrössere Einrückungstiefe beschrieben ist (z.B. die Zeilen Nr. 1 und 2 in Abbildung 9).

Die einzelnen Aktionen werden durch die beteiligte Instanz (UT oder LT in Abbildung 9), 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 9 das Senden eines ICONreq durch den Upper Tester UT an das zu testende System.

In Abbildung 10 finden sich in den Aktionsbeschreibungen OTHERWISE Konstrukte. Ein OTHERWISE kann als abürzende 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 9 werden nur die Testurteile PASS und INCONCLUSIVE vergeben. Die FAIL Fälle werden in diesem speziellen Fall durch das in Abbildung 10 abgebildete *Default* Verhalten definiert. Solche Default Beschreibungen müssen im Kopf der TTCN Testfallbeschreibung referenziert werden (siehe *Default* in Abbildung 9).

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 [12] und [11].

3.2 Die SAMSTAG Methode

Für die automatische Generierung von Testfällen wird im allgemeinen eine Testarchitektur simuliert und das Verhalten der Testprozesse protokolliert. Die Simulation wird durch eine Testfalldefinition (vgl. Abschnitt 3.1) und durch einen Testzweck gesteuert. Die aufgezeichneten Observables sind die Grundlage für die zu entwickelnde Testfallbeschreibung. Bevor wir näher auf die Testfallgenerierung eingegehen, sollen zunächst die Beziehungen zwischen Spezifikation, Testzweck und Testfall erklärt werden.

[9] ISO/IEC IS 9646 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.

Eine eigenschaftsorientierte Betrachtung des Testens. Spezifikationen und Implementierungen können als Generatoren für Traces angesehen werden. Sie definieren dadurch endliche oder sogar unendliche Mengen von Traces und Observables. Wir definieren eine Menge von Traces als eine Eigenschaft. Eine Implementierung hat genau dann die Eigenschaft einer Spezifikation, wenn die Menge der Traces der Implementierung eine Teilmenge der Menge der Traces der Spezifikation ist.

Es existieren verschiedene Klassen von Eigenschaften. In [13] werden *Guarantee Properties* (dt. Garantieeigenschaften), *Safety Properties* (dt. Sicherheitseigenschaften) und vier höhere Klassen von Eigenschaften unterschieden. Informal ausgedrückt besagt eine Garantieeigenschaft, dass in jedem Trace irgendwann einmal etwas *Erwünschtes* passiert.[10] Demgegenüber besagt eine Sicherheitseigenschaft, dass in jedem Trace niemals etwas *Unerwünschtes* passiert. Die höheren Eigenschaftsklassen beschreiben, dass sich in jedem Trace etwas Erwünschtes immer wieder, oder ab einem gewissen Zeitpunkt ständig ereignet.

Bei einem Test vergleichen wir die Traces einer Spezifikation mit den Traces einer Implementierung und versuchen etwas über die Beziehungen der beiden Tracemengen herauszufingen. 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 [14] 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 Garantie Eigenschaft 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 OSI Konformitätstesten. Sicherheits- und Garantieeigenschaften finden sich auch im allgemeinen Vorgehen für das OSI Konformitätstesten wieder. Das erlaubte Systemverhalten wird durch eine Spezifikation beschrieben. Sie kann daher als Sicherheitseigenschaft interpretiert werden.[11] Ein Testzweck definiert demgegenüber etwas, was während dem Test 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 einem Observable zugewiesen, wenn dieses die Garantieeigenschaft eindeutig nachweist und die Sicherheitseigenschaft nicht verletzt.
- **INCONCLUSIVE** wird einem Observable zugewiesen, wenn dieses die Garantieeigenschaft nicht nachweist, jedoch die Sicherheitseigenschaft auch nicht verletzt.
- **FAIL** wird einem Observable zugewiesen, wenn dieses die Sicherheitseigenschaft verletzt, egal ob die Garantieeigenschaft erfüllt wird oder nicht.[12]

[10] Eine Garantieeigenschaft kann dementsprechend auch als Erreichbarkeitskriterium aufgefasst werden.

[11] 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.

[12] Der Fall des Nachweises der Garantieeigenschaft und der Verletzung der Sicherheitseigenschaft ist zwar im FAIL ent-

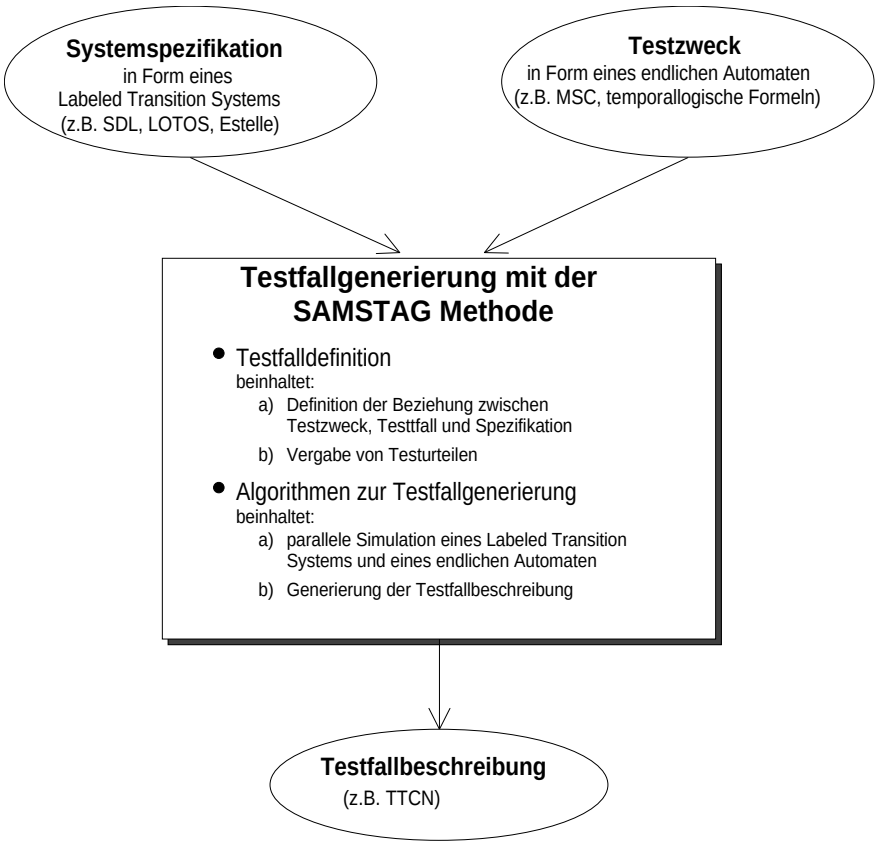


Abbildung 3: Die SAMSTAG Methode

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 Traces, 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 Traces 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 3 gezeigt. Weitere Informationen zur SAMSTAG Methode finden sich in [5], [6] und [14].

halten, 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.

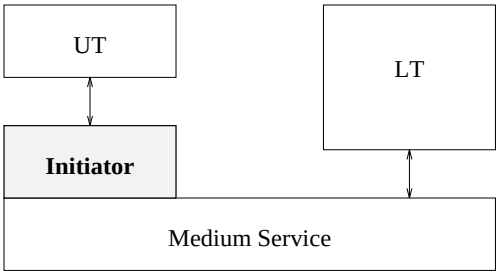


Abbildung 4: Eine Testarchitektur für die Initiator Instanz des Inres Protokolls

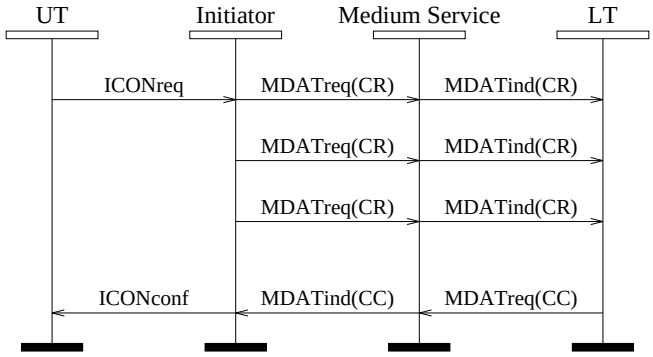


Abbildung 5: Verbindungsaufbau nach Empfang der dritten CR Protokolldateneinheit

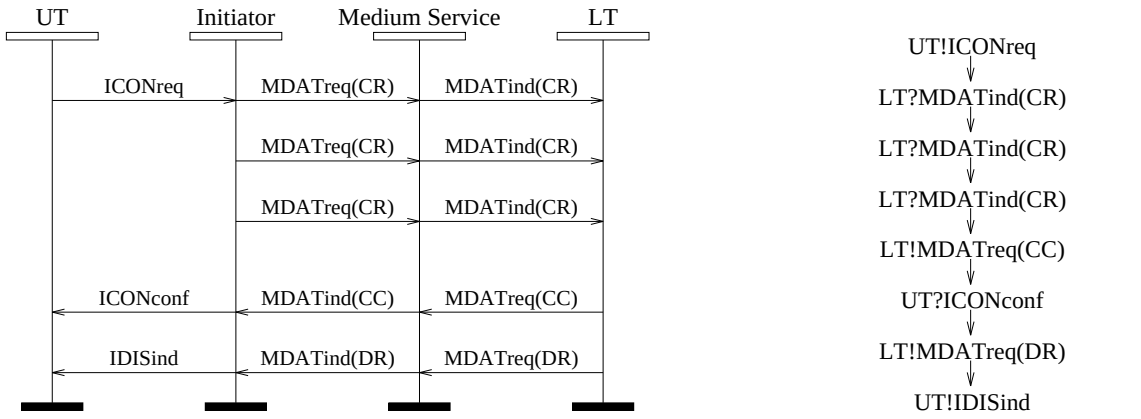
3.3 Die Testfallgenerierung mit der SAMSTAG Methode

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 4 als SDL Spezifikation vor. Getestet wird die Initiator Instanz des Inres Protokolls [8]. Der Testzweck ist durch den MSC in Abbildung 5 vorgegeben. Dieser beschreibt eine spezielle Situation bei einem Verbindungsaufbau. Der Initiator empfängt vom Upper Tester UT ein Connection Request ICONreq und übermittelt ein CR[13] 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 Initator dem Upper Tester UT durch ein Connection Indication ICONind an.

Ein Testfall besteht aus einer endlichen Menge von Observables, wobei jedem Observable ein Testurteil zugewiesen ist. Die Testurteile sind PASS, FAIL und INCONCLUSIVE. Entsprechend unterscheiden wir auch zwischen *Pass*, *Fail* und *Inconclusive Observables*. Die von uns zur Testdatengenerierung entwickelten Algorithmen basieren auf der Berechnung dieser Observables. Die Berechnung erfolgt in vier Schritten, die sich auch in der Architektur des SAMSTAG Werkzeugs widerspiegeln (Abbildung 11).

1. In einem ersten Schritt werden sogenannte *Possible Pass Observables* berechnet. Ein Possible Pass Observable ist ein Observable, zu dem es einen Trace gibt, für den folgende zwei Bedingungen gelten müssen:

[13] Da die untere Schnittstelle des Initiators nur über den Medium Dienst getestet werden kann, müssen Protokolldateneinheiten, wie z.B. CR, CC, DT, oder DR, als Parameter der Dienstprimitive MDATind und MDATreq gesendet und empfangen werden. Da der Medium Dienst die Protokolldateneinheiten transparent übermittelt und keinen Einfluss auf das Verhalten des Initiators hat, abstrahieren wir nachfolgend von den Primitven MDATind und MDATreq.



(a) MSC

(b) Possible Pass Observable

Abbildung 6: MSC aus Abbildung 5 mit möglicher Postamble und Observable

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

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

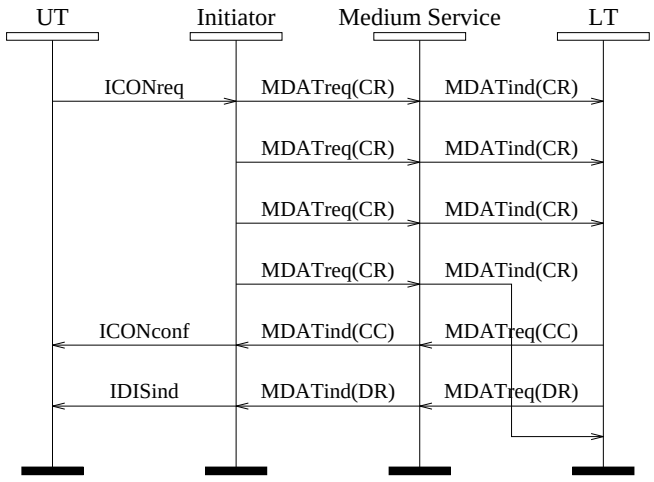
Bei der Berechnung eines Possible Pass Observables muss zunächst eine Preamble gefunden werden, die das zu testende System in einen Zustand bringt, von dem aus der MSC beobachtet werden kann. Für unser Beispiel (vgl. Abbildung 5) ist die Preamble leer, da der MSC im Initialzustand beginnt. Nach Beobachtung des Testzwecks muss das zu testende System wieder zurück in den Initialzustand geführt werden. Eine mögliche Postamble ist ein normaler Verbindungsabbau (vgl. Abbildung 6), der vom Lower Tester LT durch ein Disconnection Request DR eingeleitet und vom Initiator dem Upper Tester UT duch ein Disconnection Indication IDISind angezeigt wird. Das Observable des beschriebenen Traces ist ein Possible Pass Observable.

Im allgemeinen existiert kein eindeutiger Zusammenhang zwischen einem Trace und einem Observable. Mehrere Traces können das gleiche Observable besitzen. Der MSC in Abbildung 7 repräsentiert mehrere Traces und Observables. Das in (b) dargestellte Observable ist bis auf das abschliessende MDATind(CR) mit dem Observable in Abbildung 6 (b) identisch.

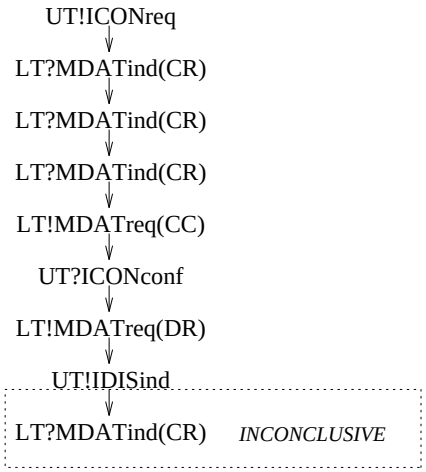
Die SDL Semantik erlaubt keine Annahmen darüber, wie lange man nach dem Empfang eines IDISind auf ein evtl. noch ausstehendes MDATind(CR) warten muss. Aufgrund der Beobachtung von IDISind kann daher nicht garantiert werden, dass der Testzweck in Abbildung 5 durchgeführt worden ist. Wird nach dem IDISind ein viertes MDATind(CR) beobachtet, muss das Testurteil INCONCLUSIVE vergeben werden (vgl. Abbildung 7 (b)).

2. In einem zweiten Schritt wird deshalb geprüft, ob alle gefundenen Possible Pass Observables eindeutig sind, d.h. dass für alle Traces des Possible Pass Observables die Bedingungen (a) und (b) (siehe Schritt 1) gelten. In diesem Fall bezeichnen wir einen Possible Pass Observable als *eindeutig* oder als *Unique Pass Observable*. Im allgemeinen kann es für einen Testfall mehrere Unique Pass Observables geben. Als Pass Observables des zu generierenden Testfalls wählen wir eine Teilmenge der kürzesten Unique Pass Observables aus.

   Der Trace, der für unser Beispiel zu einem Unique Pass Observable führt, ist in Abbildung 8 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



(a) MSC



(b) Ein Observable von (a)

Abbildung 7: Unerwünschter Systemablauf

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 das Observable des beschriebenen Traces als das Pass Observable des zu generierenden Testfalls aus (vgl. Abbildung 8 (b)). Daraus ergibt sich der TTCN Testfall in Abbildung 9. Das Pass Observable ist in den Zeilen 1 bis 12 zu sehen.

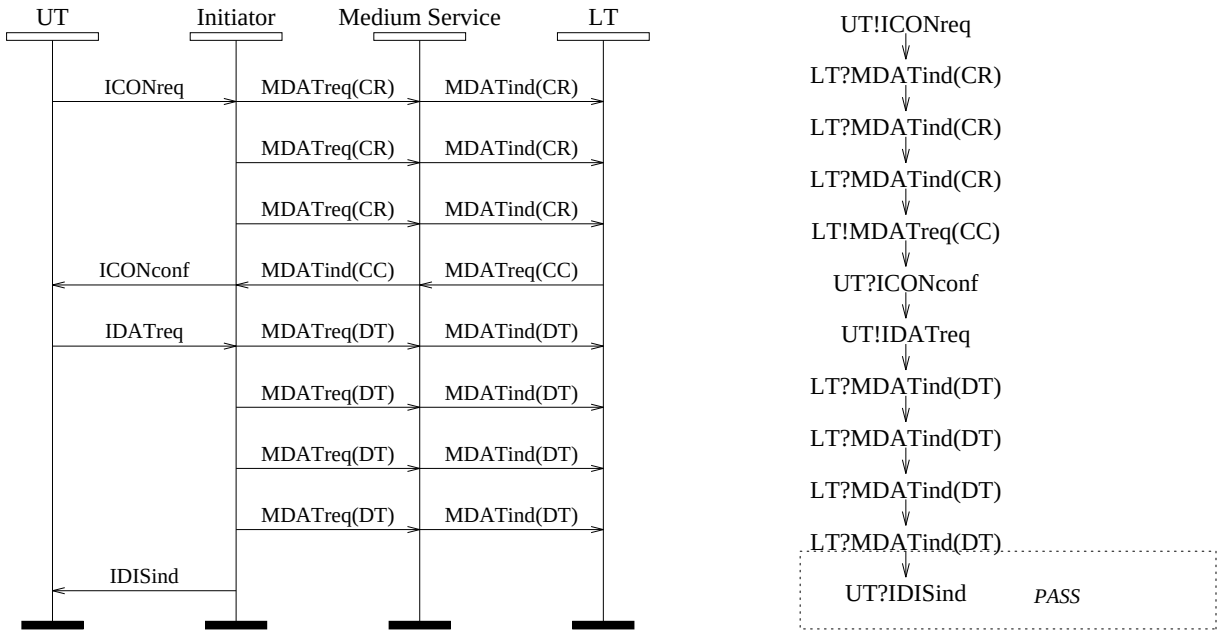
3. In einem dritten Schritt werden für die gewählten Pass Observables die zugehörigen *Inconclusive Observables* berechnet. Ein Inconclusive Observable ist ein Observable, das mit einem Pass Observable einen gewissen Zeitraum übereinstimmt, jedoch mit einer zum Pass Observable unterschiedlichen Ausgabe des SDL Systems endet, die jedoch erlaubt ist. Die Inconclusive Observables für unser Beispiel sind in der TTCN Beschreibung in Abbildung 9 enthalten.

4. Im vierten und letzten Schritt werden die *Fail Observables* definiert. Fail Observables werden nicht berechnet, da sie in TTCN mittels einer Default Beschreibung definiert werden können. Das Default für unser Beispiel findet sich in Abbildung 10.

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 11 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.



(a) MSC

(b) Unique Pass Observable

Abbildung 8: MSC aus Abbildung 5 mit eindeutiger Postamble

Der Testfallgenerator steuert den SDL Simulator und den MSC Interpreter. Bei der Testfallgenerierung berechnet er die Possible Pass Observables, die Unique Pass Observables und die Inconclusive Observables. Abschliessend definiert er die Fail Observables und gibt den generierten Testfall in der TTCN Notation aus.

Berechnung der Possible Pass Observables. Die Berechnung eines Possible Pass Observables ist ein typisches Suchproblem. Das Testfallgenerierungstool muss Traces eines SDL Systems finden, welche im Initialzustand beginnen und enden und zusätzlich den Signalaustausch des MSC durchführen. Die zu diesen Traces gehörenden Observables sind die Possible Pass Observables. Unglücklicherweise ist das Auffinden der Possible Pass Observables für SDL Systeme nicht entscheidbar, da dieses Problem äquivalent zum Halteproblem für Turingmaschinen ist. Man kann deshalb nur suchen und hoffen, dass man die gewünschten Observables findet. Wir suchen, indem wir die SDL Spezifikation und den MSC gleichzeitig simulieren.

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. Deshalb verwenden wir eine *k-beschränkte Tiefensuche*, welche alle Traces der Länge k evaluiert. Wird kein geforderter Trace gefunden, so kann die Suche abgebrochen oder mit einer grösseren Schranke k wiederholt werden.

Berechnung der Unique Pass Observables. Für jedes Possible Pass Observable wird geprüft, ob es ein Unique Pass Observable ist. Dabei werden alle Traces des SDL Systems simuliert, welche den Possible Pass Observable als Observable besitzen. Wenn Traces existieren, die nicht die Bedingung (a) und (b) auf Seite 13 erfüllen, so ist das Possible Pass Observable nicht eindeutig. Werden mehrere Unique Pass

Test Case Dynamic Behaviour					
Test Case Name : Test_Case_Example Group : Inres_Protocol/Initiator_Test/Connection_Establishment Purpose : Connection Establishment after the third retransmission of a Connection Request Default : Unexpected Events Comments :					
Nr.	Label	Behaviour Desription	Constraint Ref.	Verdict	Comments
1		UT!ICONreq			
2		LT?MDATind(CR)			
3		LT?MDATind(CR)			
4		LT?MDATind(CR)			
5		LT!MDATreq(CC)			
6		UT?ICONconf			
7		UT!IDATreq			
8		LT?MDATind(DT)			
9		LT?MDATind(DT)			
10		LT?MDATind(DT)			
11		LT?MDATind(DT)			
12		UT?IDISind		PASS	
13		LT?MDATind(CR)		INCONC	
14		LT?MDATind(CR)		INCONC	
Detailed Comments :					

Abbildung 9: Ein TTCN Testfall

Default Dynamic Behaviour					
Test Step Name : Unexpected Events Group : Inres_Protocol/Initiator_Test/Connection_Establishment Objective : Handle unexpected Events Comments :					
Nr.	Label	Behaviour Desription	Constraint Ref.	Verdict	Comments
1		UT?OTHERWISE		FAIL	
2		LT?OTHERWISE		FAIL	
Detailed Comments :					

Abbildung 10: TTCN Default für den TTCN Testfall in Abbildung 9

Observables gefunden, so wählen wir als Pass Observables des Testfalles eine Teilmenge der kürzesten Unique Pass Observables aus. Die Existenz von Unique Pass Observables kann leider nicht garantiert werden. Unter Umständen existieren zwar Possible, jedoch keine Unique Pass Observables.

Berechnung der Inconclusive Observables. Für die gewählten Pass Observables werden die Inconclusive Observables berechnet. Hierbei werden mittels einer Tiefensuche erneut alle Traces simuliert, deren Observable bis zur vorletzten Aktion einem Pass Observable entspricht, dann aber in einer unterschiedlichen Ausgabe des Systems enden.

Definition der Fail Observables und Ausgabe des TTCN Testfalles. Abschliessend werden das Default für die Fail Observables erzeugt, sowie die Pass und Inconclusive Observables in die TTCN Notation transformiert. Als Ergebnis wird die TTCN Beschreibung des generierten Testfalls als ASCII File ausgegeben.

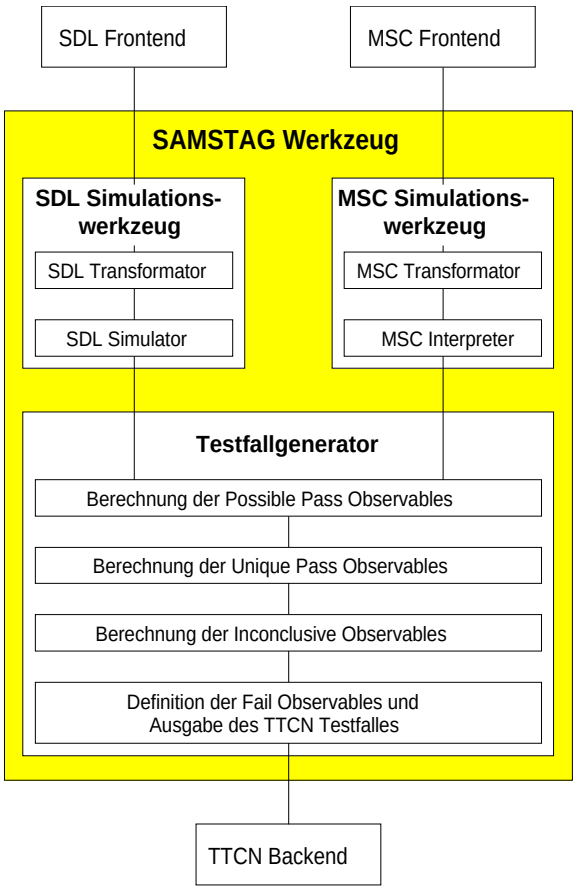


Abbildung 11: Die Architektur des SAMSTAG Werkzeugs

4 Zusammenfassung und Ausblick

Im letzten Abschnitt sind die von uns entwickelte SAMSTAG Methode und das SAMSTAG Werkzeug vorgestellt worden. Mit der SAMSTAG Methoden können, basierend auf einer formalen Protokollspezifikation und einer Menge von Testzwecken, Testfälle für Konformitätstests generiert werden. Hierfür musste insbesondere der für das OSI Kormiätstesten wichtige Begriff des Testzwecks formalisiert werden. In Kapitel 2 sind zudem das Konformitätstesten nach ISO/IEC IS 9646 und die im wissenschaftlichen Umfeld entwickelten theoretischen Methoden diskutiert worden.

Die vorgestellten Ansätze formalisieren unterschiedliche Schritte beim Konformitätstesten. Der Beitrag, den die drei Anätze bei einer vollständigen formalen Beschreibung des OSI Konformitätstestestens leisten können, ist als Überblick in Abbildung 12 dargestellt. Dabei gibt es sicherlich Überschneidungen der drei Methoden, die jedoch aus Gründen der Übersichtlichkeit in der Abbildung weggelassen worden sind.

Die Abbildung zeigt aber auch, dass die theoretischen Grundlagen für die Bestimmung der Testzwecke (1) 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 (j), formalisieren könnte. Für die praktische Anwendbarkeit eines vollständigen formalen Modells für das Konformitätstesten müssten zusätzlich noch die Begriffe Überdeckungskriterium (j) und Konformitätsrelation (h) näher untersucht und verallgemeinert werden.

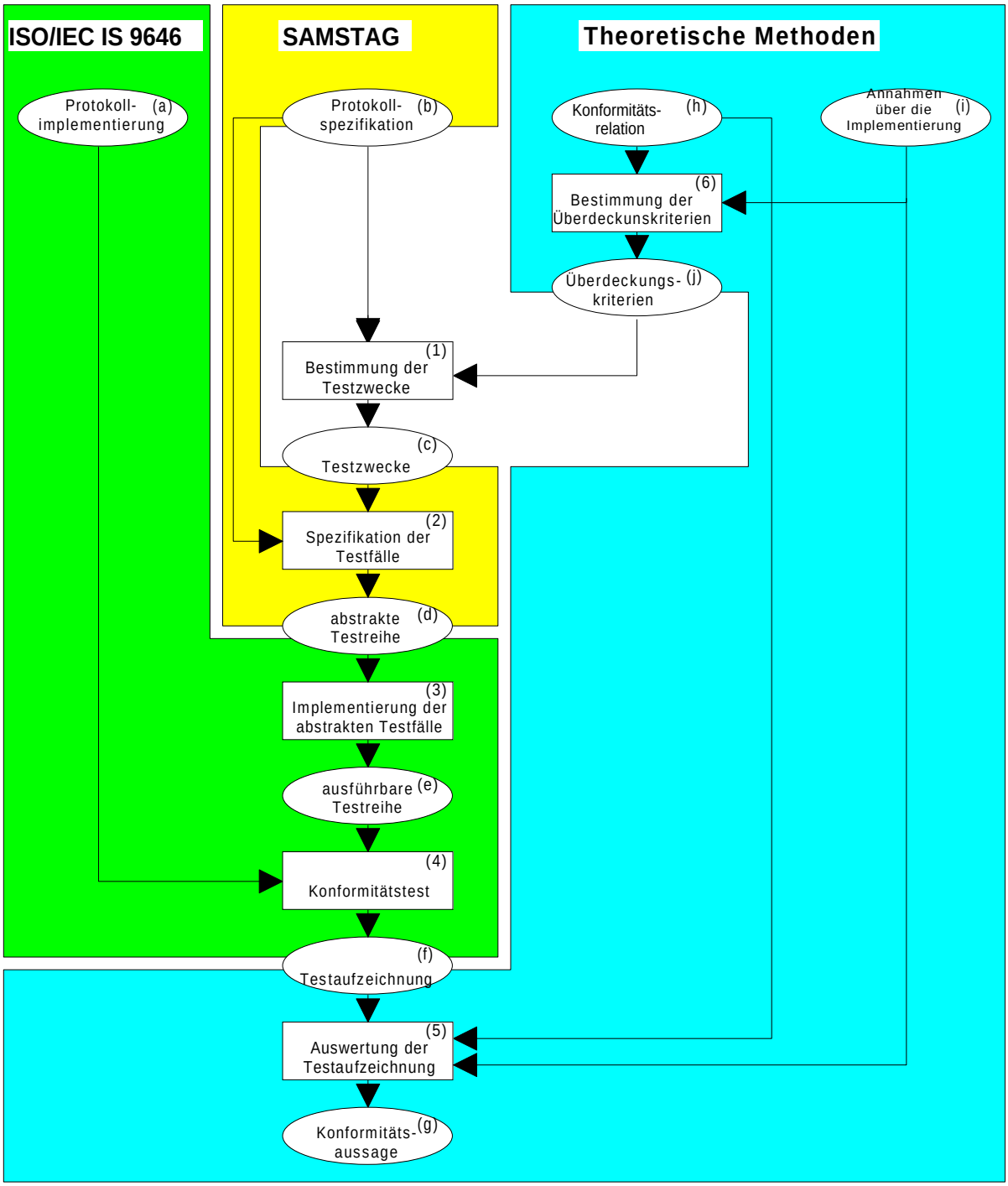


Abbildung 12: Vollständige formale Beschreibung des OSI-Konformitätstestens

Literatur

[1] Siemens AG. Produktbeschreibungen K1197, K1103. Siemens AG Berlin, 1993.

[2] F. Belina, D. Hogrefe, and A. Sarma. *SDL with Applications from Protocol Specification.* The BCS Practitioner Series, Series editor: R. Welland. Carl Hanser Verlag and Prentice Hall International, 1991.

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

[4] J. Grabowski, D. Hogrefe, P. Ladkin, S. Leue, and R. Nahm. Conformance Testing - A Tool for the Generation of Test Cases. First Interim Report of the F & E Project, Contract No. 233/257, Funded by Swiss PTT, May 1992.

[5] J. Grabowski, D. Hogrefe, and R. Nahm. A Method for the Generation of Test Cases Based on SDL and MSCs. Technical Report IAM-93-010, Universität Bern, Institut für Informatik, April 1993.

[6] J. Grabowski, D. Hogrefe, and R. Nahm. Test Case Generation with Test Purpose Specification by MSCs. In *Proceedings of the 6th SDL Forum in Darmstadt (Germany)*, October 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). Technical Report IAM-91-012, Universität Bern, Institut für Informatik, May 1991, Update May 1992.

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

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

[11] ISO/IEC JTC 1/SC21. Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework - Part 3: The Tree and Tabular Combined Notation. International Standard 9646-3, ISO/IEC, 1992.

[12] J. Kroon and A. Wiles. A Tutorial on TTCN. In *Proceedings of the 11th International IFIP WG 6.1 Symposium on Protocol, Specification, Testing and Verification*, 1991.

[13] Z. Manna and A. Pnueli. A Hierarchy of Temporal Properties. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing.* ACM Press, 1990. 1990 ACM-0-89791-404-X/90/0008/3777.

[14] R. Nahm, J. Grabowski, and D. Hogrefe. Test Case Generation for Temporal Properties. Technical Report IAM-93-013, Universität Bern, Institut für Informatik, June 1993.

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

[16] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.100: Functional Specification and Description Language (SDL) (formerly CCITT Recommendation Z.100). ITU, Geneva, June 1992.

[17] ITU Telecommunication Standards Sector SG 10. ITU-T Recommendation Z.120: Message Sequence Chart (MSC). ITU, Geneva, June 1992.

[18] J. Tretmans. *A Formal Approach to Conformance Testing*. PhD thesis, University of Twente (The Netherlands), December 1992.

[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.