REIHE INFORMATIK
4/93

Integration von Darstellungs- und Kommunikationssteuerungsschicht in Estelle

B. Hofmann
Universität Mannheim
Seminargebäude A5
6800 Mannheim

Integration von Darstellungs- und Kommunikationssteuerungsschicht in Estelle

Bernd Hofmann
Universität Mannheim
hofmann@pi4.informatik.uni-mannheim.de

Zusammenfassung

Herkömmliche Implementierungen von OSI-Protokollen orientieren sich an der Schichteneinteilung des OSI-Referenzmodells. Dadurch wird die Leistung vor allem in den anwendungsorientierten Schichten durch die Kommunikation zwischen den Schichten beeinträchtigt. Der vorliegende Artikel beschreibt, wie durch eine Integration benachbarter Protokollschichten sowohl Nachrichtenaustausch als auch Zustandswechsel eingespart werden können. Erste Erfahrungen mit einer integrierten Implementierung der Darstellungs- und Kommunikationssteuerungsschicht liegen bereits vor und ergeben signifikante Geschwindigkeitssteigerungen.

1 Einleitung

Zur Kommunikation von Rechnern in offenen Netzen wurden von der ISO das Basisreferenzmodell und die zugehörigen Protokolle standardisiert [ISO84a]. Dabei wurde von relativ langsamen Übertragungsmedien ausgegangen, so daß die Abwicklung komplexer Protokolle im Vergleich zur Übertragungszeit nicht ins Gewicht fiel und dadurch möglich und sinnvoll war.

Mittlerweile aber sind schnellere Übertragungsmedien wie z. B. Glasfaserkabel verfügbar. Dadurch verschiebt sich der Engpaß vom Netz zum Host, da die Verarbeitung der Protokolle nicht mehr in einer dem Netz angepaßten Geschwindigkeit erfolgen kann. Es muß daher nach Wegen gesucht werden, wie die Protokolle – wenn man sie beibehalten will – effizienter implementiert werden können.

Häufig wird die schichtenweise Implementierung als eine Quelle für Effizienzverlust genannt [Svo89, Web91, Sok91]. Insbesondere bei den anwendungsorientierten Schichten müssen oft Dienste, die schon in tiefergelegenen Schichten angeboten werden, durchgereicht werden, um sie einer höhergelegenen Schicht anbieten zu können. Tragen die dazwischenliegenden Schichten nichts zu diesen Diensten bei, könnte einiger Kommunikationsaufwand

eingespart werden, wenn ein schichtenübergreifender Dienstzugriff möglich wäre. Dies wird aber durch die strenge Schichteneinteilung in einer Implementierung unmöglich gemacht.

Diese – für den Entwurf durchaus sinnvolle Einteilung – muß bei der Implementierung nicht unbedingt beibehalten werden. Die gerade geschilderten Nachteile können durch die Integration benachbarter Schichten bei der Implementierung vermieden werden. Dabei werden die Protokollmaschinen der betreffenden Schichten zu einer einzigen zusammengefaßt, die die äußeren Schnittstellen beibehält, sich also für die restlichen Schichten wie die ersetzten verhält. Intern werden aber weniger Zustandswechsel vorgenommen, da nun ein schichtenübergreifender Dienstzugriff möglich ist.

In diesem Artikel wird exemplarisch die Integration der Darstellungsschicht (presentation layer) und der Kommunikationssteuerungsschicht (session layer) gezeigt. Da zum einen die Protokolle in Form von endlichen Automaten standardisiert sind, zum anderen sich die Integration sehr gut mit den Mitteln der Automatentheorie beschreiben läßt, wird als Spezifikationssprache die automatenorientierte Sprache Estelle verwendet. Dies hat darüberhinaus den Vorteil, daß bestehende Estelle-Spezifikationen als Referenz zur Verfügung stehen. Die Implementierung erfolgt werkzeuggestützt mithilfe eines Estelle-C-Compilers. Erste vergleichende Messungen der Laufzeiten ergaben Geschwindigkeitssteigerungen um den Faktor 2. Dies ist aber lediglich eine untere Schranke, der tatsächliche Speedup läßt sich erst durch genauere Messungen ermitteln.

Der Artikel gliedert sich wie folgt: Im nächsten Abschnitt werden die Nachteile einer schichtenweisen Implementierung genauer erörtert und an einigen Beispielen belegt. Die Integration zweier Automaten wird in Abschnitt 3 beschrieben. In Abschnitt 4 wird die Implementierung der integrierten Darstellungs- und Kommunikationssteuerungsschicht vorgestellt. Hier werden auch die ersten Ergebnisse von Laufzeitmessungen angegeben. Schließlich werden im letzten Abschnitt die bisherigen Ergebnisse zusammengefaßt und ein Ausblick auf zukünftige Arbeiten gegeben.

2 Schichtenweise Implementierung

Der schichtenweise Aufbau des Architekturmodells legt eine schichtenweise Implementierung, d. h. unter Beibehaltung der Schnittstellen zwischen den Schichten, nahe. Es gibt im wesentlichen zwei Methoden: das *server model* und das *activity thread model* [Svo89]. Beim *server model* wird jede Schicht durch einen oder mehrere Prozesse implementiert, die durch Interprozeßkommunikation miteinander kommunizieren. Beim *activity thread model* hingegen werden mehrere Schichten durch einen Prozeß implementiert, die Kommunikation zwischen den Schichten erfolgt durch Prozeduraufrufe. In jedem Fall aber bleibt die schichtenweise Aufteilung in der Implementierung erhalten, und die Kommunikation zwischen benachbarten Schichten wird gemäß dem Standard abgewickelt.

Vorteile dieser Vorgehensweise sind zum einen die Möglichkeit, eine neue Implemen-

tierung einer Schicht oder sogar ein ganz neues Protokoll einfach in das Gesamtsystem einfügen zu können. Hierbei muß nur darauf geachtet werden, daß die Schnittstelle beibehalten wird. Zum anderen ist gar keine andere Implementierung möglich, wenn ein Teil der Protokolle in Hardware, ein anderer in Software realisiert ist (z. B. physikalische und MAC-Schicht auf einer Karte, Rest in Software). Auch wenn benachbarte Schichten aus systemtechnischen Gründen in verschiedenen Adreßräumen laufen sollen, z. B. im Anwender- und im Kernel-Adreßraum, ist es sinnvoll, die ISO-konforme Dienstschnittstelle beizubehalten.

Für die anwendungsorientierten Schichten gilt dies aber nicht uneingeschränkt. So haben z. B. die Kommunikationssteuerungs- und die Darstellungsschicht „ähnliche" Protokollautomaten. Betrachtet man nämlich den gesamten Protokollturm, so steht den Kommunikationspartnern erstmals ab der Transportschicht eine Ende-zu-Ende-Verbindung zur Verfügung, die darunterliegenden Netze und Netzarchitekturen sind verdeckt.

An dieser grundsätzlichen Ende-zu-Ende-Struktur ändert sich bis in die Anwendungsschicht nichts, es kommen lediglich neue Funktionen wie beispielsweise Dialogmanagement, Synchronisation, Umwandlung unterschiedlicher Datenrepräsentationen etc. hinzu. Diese Funktionen sind letztlich nur Prozeduren, die bei Zustandsübergängen der Protokollautomaten aufgerufen werden.

Daneben sind diese Schichten bereits semantisch verzahnt, denn $(N+1)$-PDUs werden direkt auf entsprechende (N)-PDUs abgebildet und nicht – wie in tieferen Schichten – transparent als (N)-DATA übertragen (z. B. beim Verbindungsaufbau).

Hinzu kommt, daß Dienste, die schon in Schichten unterhalb der Anwendungsschicht voll zur Verfügung stehen, wegen der Schichtenarchitektur nur dadurch der Anwendung zugänglich gemacht werden können, indem sie von allen dazwischenliegenden Schichten unverändert durchgereicht werden.

Ein Beispiel hierfür ist die einfache Datenübertragung nach dem Verbindungsaufbau. Dieser Dienst steht bereits an der Transport-Schnittstelle zur Verfügung (T-DATA.request). Er ist aber wegen der strengen Schichteneinteilung nicht direkt an der Dienstschnittstelle der Kommunikationssteuerungsschicht verfügbar, sondern nur über ein entsprechendes Dienstprimitiv dieser Schicht (S-DATA.request). Der Protokollautomat der Kommunikationssteuerungsschicht reagiert auf den Empfang dieses Dienstprimitivs mit dem Senden eines T-DATA.requests. Außer der PDU-Kodierung fallen keine spezifischen Aufgaben der Kommunikationssteuerungsschicht an. Der Vorgang wiederholt sich analog in der Darstellungsschicht (s. Abb. 1).

Weitere Beispiele wären Verbindungsaufbau und Expedited Data (ab Transportschicht) sowie geordneter Verbindungsabbau, Typed Data, Capability Data, Dialogmanagement, Synchronisation, Aktivitätenmanagement und Ausnahmebehandlung (ab Kommunikationssteuerungsschicht).

Dieses Durchreichen von Dienstprimitiven erzeugt zweierlei Mehraufwand: zum einen müssen in jeder beteiligten Schicht zusätzliche Zustandsübergänge durchgeführt werden,

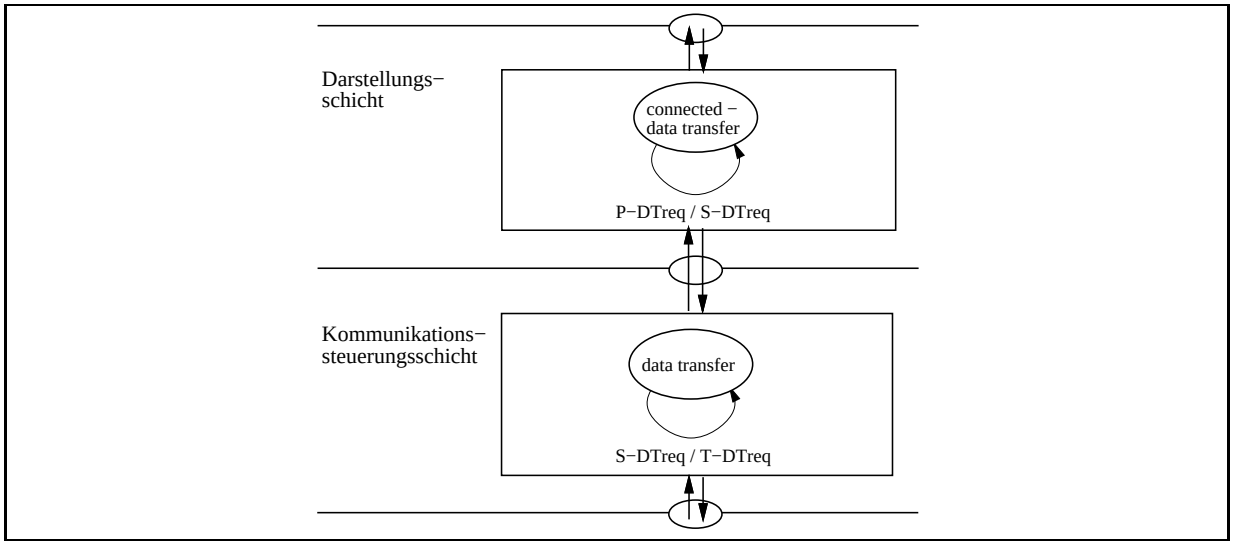

Abbildung 1: Durchreichen eines Dienstes

zum anderen Parameter weitergereicht oder – je nach Implementierung – sogar kopiert werden. Im Hinblick auf eine effiziente Implementierung bietet die Beseitigung dieses Mehraufwands ein Potential zur Geschwindigkeitssteigerung.

3 Integration

Der schichtenweise Aufbau des ISO/OSI-Basisreferenzmodells ermöglicht erst die Spezifikation eines komplexen Kommunikationsubsystems, indem das Problem in kleinere, in sich geschlossene Teilprobleme mit klaren Schnittstellen zerlegt wird. Für die Implementierung ist eine solche Aufteilung aber nicht zwingend vorgeschrieben [Svo89].

Das oben beschriebene Durchreichen von Dienstprimitiven und die damit verbundenen Effizienzverluste lassen sich durch die Integration benachbarter Schichten vermeiden. Das ist natürlich nur bei einer beschränkten Anzahl von Schichten sinnvoll, um die Vorteile der Modularisierung nicht ganz aufzugeben. Deshalb bietet sich eine solche Vorgehensweise besonders bei Schichten mit „ähnlichen“ Protokollautomaten wie z. B. Darstellungs- und Kommunikationssteuerungsschicht an. Dieser Artikel beschränkt sich auf die Integration dieser beiden Schichten, es ist aber auch eine Einbeziehung von ACSE denkbar und für die Zukunft geplant.

Die Integration von Darstellungs- und Kommunikationssteuerungsschicht resultiert in einer Protokoll-Instanz, die den Darstellungs-Dienst auf der Basis des Transport-Dienstes erbringt. Die „äußeren“ Schnittstellen der integrierten Schichten werden dabei gemäß dem

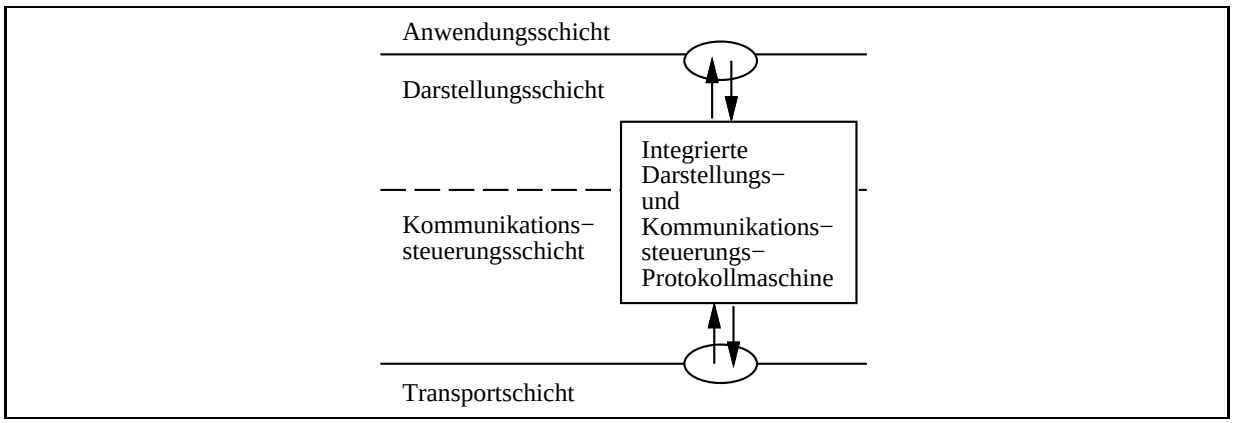

Abbildung 2: Integration von Darstellungs- und Kommunikationssteuerungsschicht

Standard eingehalten (s. Abb. 2).

Allgemein ist der Automat[1] einer integrierten Protokoll-Instanz der Produktautomat der zu integrierenden Automaten im Sinne der Automatentheorie (s. Def. 1) mit bestimmten Vereinfachungen. Diese nutzen die Tatsache aus, daß die Ausgangsautomaten Protokollmaschinen *benachbarter* Schichten sind.

Definition 1 (Produktautomat) *Seien $\alpha_j = (Z_j, E_j, A_j, \delta_j, \lambda_j, i_j)$, $j \in \{1,2\}$ zwei endliche Automaten mit den Zustandsmengen Z_j, den Eingabealphabeten E_j mit $E_1 \cap E_2 = \emptyset$, den Ausgabealphabeten A_j, den Zustandsübergangsfunktionen $\delta_j : Z_j \times E_j \to Z_j$, den Ausgabefunktionen $\lambda_j : Z_j \times E_j \to A_j$ sowie den Startzuständen i_j. Dann ist*

$$\alpha_1 \times \alpha_2 := (Z_1 \times Z_2, E_1 \cup E_2, A_1 \cup A_2, \delta, \lambda, (i_1, i_2))$$

mit

$$\delta : (Z_1 \times Z_2) \times (E_1 \cup E_2) \to (Z_1 \times Z_2),$$

$$\delta((z_1, z_2), e) := \begin{cases} (\delta_1(z_1, e), z_2) & falls \quad e \in E_1 \\ (z_1, \delta_2(z_2, e)) & falls \quad e \in E_2 \end{cases}$$

und

$$\lambda : (Z_1 \times Z_2) \times (E_1 \cup E_2) \to (A_1 \cup A_2),$$

[1] Aus Vereinfachungsgründen werden im folgenden nur Endliche Automaten und nicht *Erweiterte* Endliche Automaten beschrieben, wie sie eigentlich zur Abbildung von OSI-Protokollen nötig wären. Dies stellt keine Einschränkung dar, da die Aussagen unabhängig von der Verwendung von Variablen und Parametern sind und sich daher direkt auf Erweiterte Endliche Automaten übertragen lassen.

$$\lambda((z_1, z_2), e) := \begin{cases} \lambda_1(z_1, e) & falls \quad e \in E_1 \\ \lambda_2(z_2, e) & falls \quad e \in E_2 \end{cases}$$

Kann nun davon ausgegangen werden, daß – wie das bei Protokollautomaten benachbarter Schichten der Fall ist – Ausgaben des einen Automaten Eingaben des anderen sind, dann kann eine Transition des einen Automaten direkt eine Transition des anderen Automaten auslösen. Dieser Fall tritt ein, wenn das Ausgabezeichen der ersten und das Eingabezeichen der zweiten Transition identisch sind. Diese beiden Transitionen, die im integrierten Automaten direkt hintereinander kommen, können dann durch eine Transition ersetzt werden:

Satz 1 *Seien α_1, α_2 zwei endliche Automaten mit $A_1 \cap E_2 \neq \emptyset$. Sei weiter*

$$z_1 \overset{e_1/\lambda_1(z_1,e_1)}{\longrightarrow} \delta_1(z_1, e_1)$$

eine Transition von α_1 und

$$z_2 \overset{e_2/\lambda_2(z_2,e_2)}{\longrightarrow} \delta_2(z_2, e_2)$$

eine Transition von α_2, dann ist

$$(z_1, z_2) \overset{e_1/\lambda_1(z_1,e_1)}{\longrightarrow} (\delta_1(z_1, e_1), z_2) \overset{e_2/\lambda_2(z_2,e_2)}{\longrightarrow} (\delta_1(z_1, e_1), \delta_2(z_2, e_2))$$

eine Transitionenfolge von $\alpha_1 \times \alpha_2$.

Ist nun $e_2 = \lambda_1(z_1, e_1) \in A_1 \cap E_2$, dann läßt sich diese Folge ersetzen durch die Transition

$$(z_1, z_2) \overset{e_1/\lambda_2(z_2,\lambda_1(z_1,e_1))}{\longrightarrow} (\delta_1(z_1, e_1), \delta_2(z_2, e_2)).$$

Darüberhinaus sind in dem Produktautomaten Transitionen vorhanden, die nie ausgeführt werden können. Genaugenommen ist das neue Eingabealphabet nämlich nicht die Vereinigung von E_1 und E_2, sondern eine echte Teilmenge davon. Das rührt daher, daß bei der Integration die Dienstprimitive der zwischenliegenden Schicht entfallen.

Unterscheidet man also zwischen Eingaben vom Dienstbenutzer E^U und Eingaben vom darunterliegenden Diensterbringer E^P, so gilt $E = E^U \cup E^P$ und $E^U \cap E^P = \emptyset$. Ebenso läßt sich das Ausgabealphabet partitionieren in Ausgaben an den Dienstbenutzer A^U und Ausgaben an den Diensterbringer A^P.

Dann gilt aber bei benachbarten Protokollautomaten α_1, α_2: $E_1^P = A_2^U$ und $E_2^U = A_1^P$. Da diese Zeichen beim integrierten Automaten nicht mehr gesendet bzw. empfangen werden, ist das Eingabealphabet des integrierten Automaten $E = E_1^U \cup E_2^P$, das Ausgabealphabet entsprechend $A = A_1^U \cup A_2^U$ (s. Abb. 3). Dadurch entfallen im Produktautomaten sämtliche Transitionen, deren Eingabezeichen in E_2^U oder E_1^P liegen.

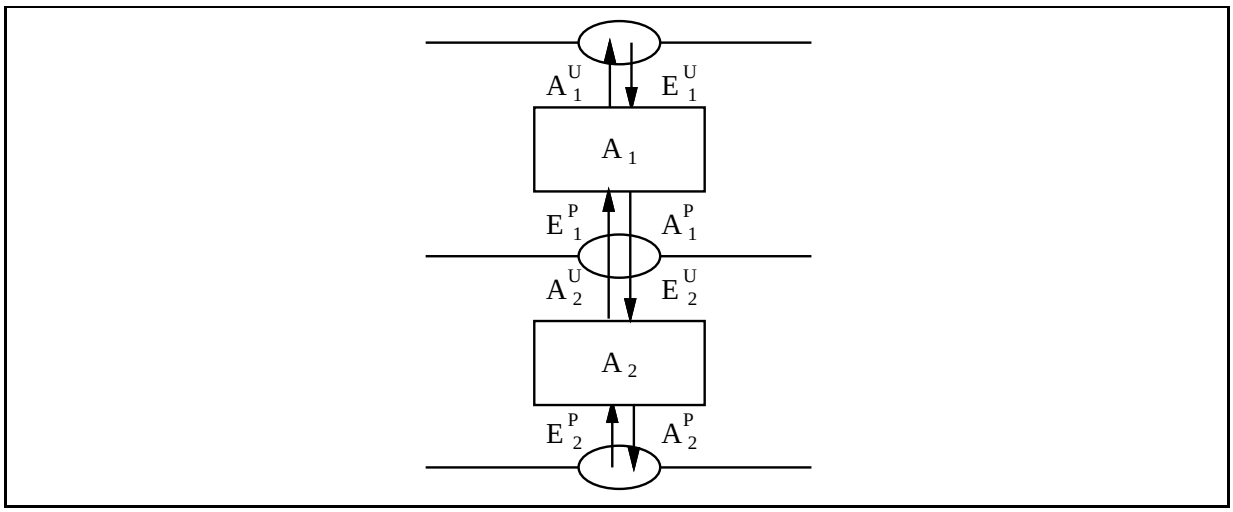

Abbildung 3: Partitionierung der Alphabete benachbarter Protokollautomaten

Ein Beispiel soll dies anhand der Darstellungs- und Kommunikationssteuerungsschicht verdeutlichen. Seien $E_P = E_P^U \cup E_P^P$ und $E_S = E_S^U \cup E_S^P$ die Eingabealphabete von Darstellungs- und Kommunikationssteuerungsschicht, $A_P = A_P^U \cup A_P^P$ und $A_S = A_S^U \cup A_S^P$ die entsprechenden Ausgabealphabete. Beim Verbindungsaufbau schickt der Darstellungsdienst-Benutzer das Dienstprimitiv P-CONNECT.request an die Darstellungs-Protokollmaschine. Diese sendet daraufhin das Dienstprimitiv S-CONNECT.request an die Protokollmaschine der Kommunikationssteuerungsschicht, was dort die Übergabe eines T-CONNECT.requests an die Transportschicht zur Folge hat (s. Abb. 4).

Im Produktautomaten entsprechen diese Transitionen dem in Abb. 5 gezeigten Ausschnitt.

Da nun S-CONNECT.request sowohl in E_S^U als auch in A_P^P liegt, können nach Satz 1 die Transitionen 1 und 2 ersetzt werden durch die Transition 1' (s. Abb. 6).

Des weiteren kann die Tatsache ausgenutzt werden, daß S-CONNECT.request nicht in E_S^P liegt, also auch nicht im Eingabealphabet des integrierten Automaten $E = E_P^U \cup E_S^P$. Da der integrierte Automat dieses Zeichen also gar nicht empfangen kann, kann die Transition 3 in Abb. 5 auch nie ausgeführt werden. Sie kann daher gestrichen werden. Dadurch wird der Zustand *idle/await T-CONcnf* unerreichbar und kann ebenfalls gelöscht werden.

Abbildung 6 zeigt also die endgültige Situation, wie sie sich im integrierten Automaten darstellt. Aus den zwei Transitionen bei Darstellungs- und Kommunikationssteuerungsschicht wurde also durch die Integration eine einzige, darüberhinaus konnte ein Nachrichtenaustausch eingespart werden.

Dies setzt natürlich stillschweigend voraus, daß durch die Integration zweier Automa-

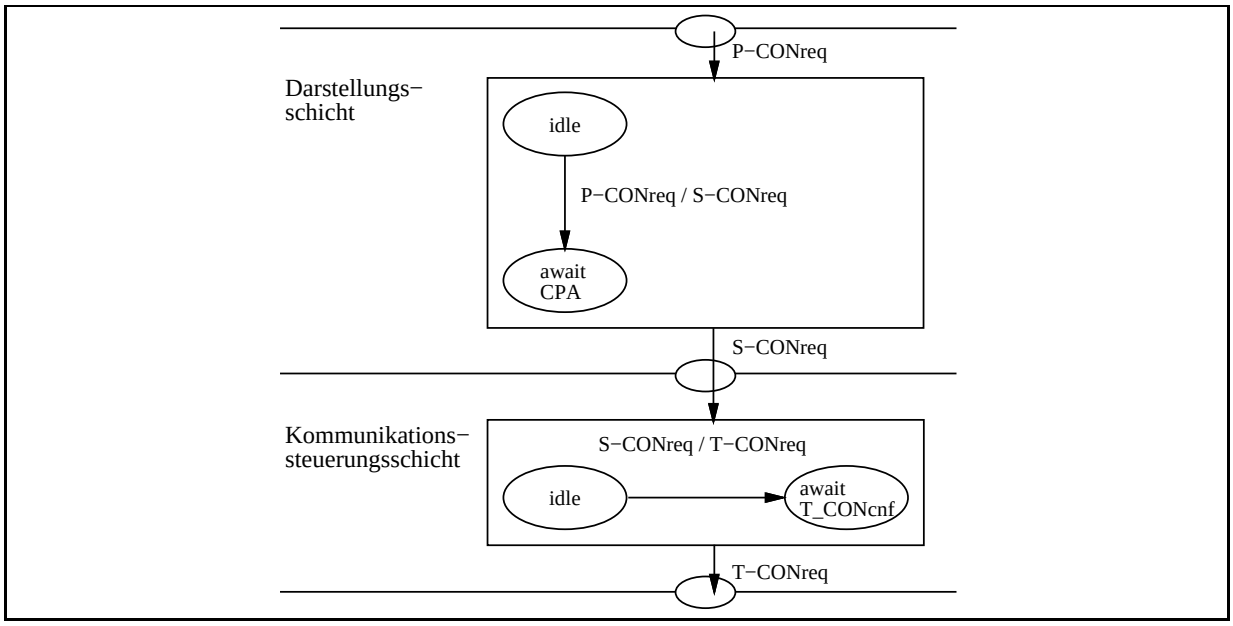

Abbildung 4: Verbindungsaufbau bei Darstellungs- und Kommunikationssteuerungsschicht

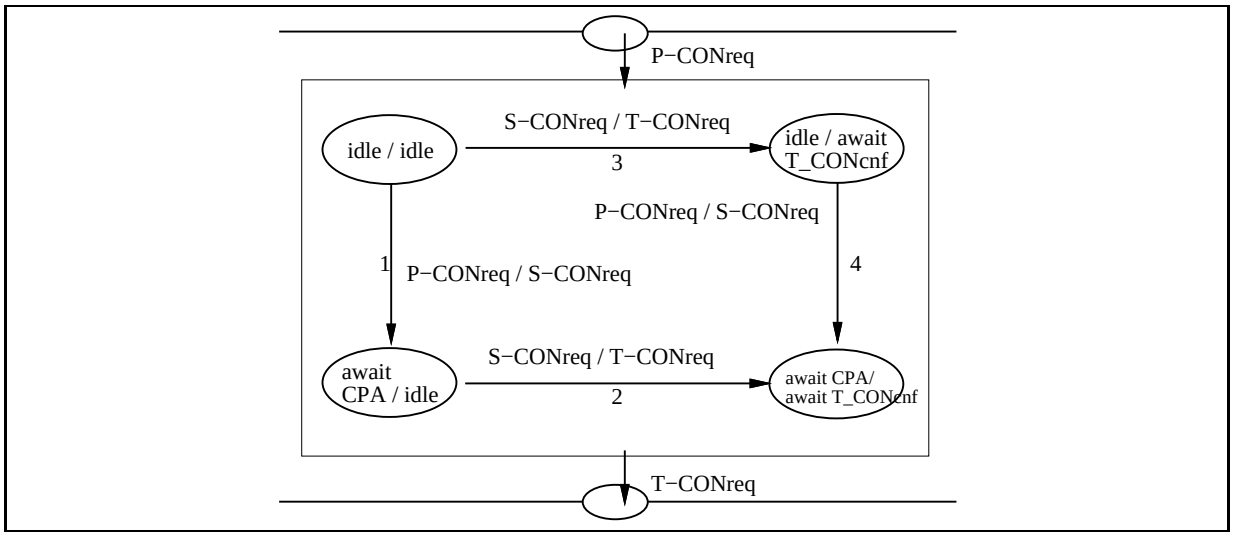

Abbildung 5: Verbindungsaufbau beim Produktautomaten aus Darstellungs- und Kommunikationssteuerungsschicht

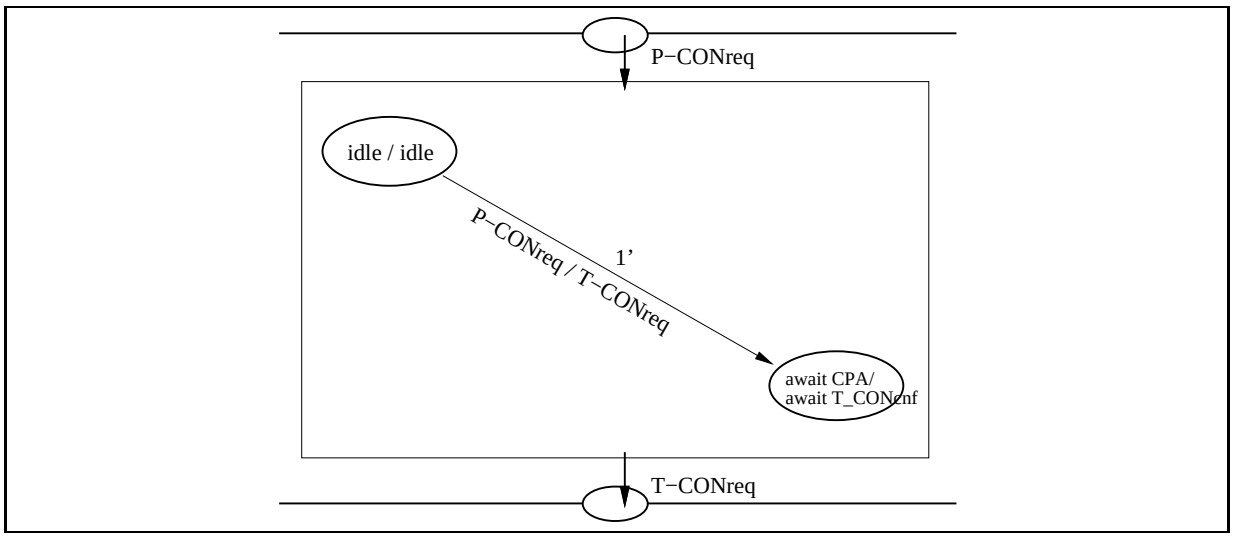

Abbildung 6: Verbindungsaufbau beim integrierten Automaten

ten keine Verfälschungen auftreten, d.h. daß das Ein-/Ausgabeverhalten nicht verändert wird. Es ist aber auf Grund der Konstruktionsvorschrift leicht einsichtig, daß der integrierte Automat und die Kombination der Ausgangsautomaten äquivalent im Sinne der Automatentheorie sind, also für gleiche Eingabefolgen gleiche Ausgaben erzeugen (zum Äquivalenzbegriff s. zum Beispiel [Sal69, Bra84, Hof91]).

4 Integrierte Implementierung

Um quantitative Aussagen über die Effizienzsteigerung durch die Integration benachbarter Schichten machen zu können, wurde damit begonnen, eine integrierte Implementierung zunächst der Darstellungs- und Kommunikationssteuerungsschicht zu erstellen.

Als Spezifikationssprache wurde Estelle [ISO89, BD87, Hog89] gewählt. Estelle bietet mehrere Vorteile: Da es eine von der ISO genormte Sprache ist, ist die Verwendung von und die Anbindung an bestehende Spezifikationen leicht möglich. Sie ist sehr eng an das Modell der Erweiterten Endlichen Automaten angelehnt und ermöglicht dadurch eine einfache Umsetzung von Verfahren, die mit Hilfe der Automatentheorie formuliert wurden. Des weiteren existiert eine Reihe von Werkzeugen für Estelle-Spezifikationen, unter anderen auch der Estelle-C-Compiler des *National Institute of Standards and Technology (NIST)* [FHSW89, NIS87]. Dieser erlaubt die halbautomatische Implementierung von Estelle-Spezifikationen. Dabei werden die automatenspezifischen Teile direkt in C-Code umgesetzt, für die benutzerspezifischen Teile werden Prozedurrahmen generiert, die dann von Hand

ergänzt werden müssen.

Unsere Entwicklungsumgebung an der Universität Mannheim ist ein DECsystem 5400 unter Ultrix 4.0, die Implementierungssprache ist C [KR83].

Ausgangspunkt für die Erstellung einer integrierten Spezifikation und einer Vergleichsimplementierung waren die Spezifikationen der Darstellungs- und Kommunikationssteuerungsschicht, die freundlicherweise [Web91] entnommen werden durften. Sie gehen letztlich auf [FLGL89] und [MM89] zurück. Die Kommunikationssteuerungsschicht umfaßt das *Basic Combined Subset*, der Funktionsumfang der Darstellungsschicht beschränkt sich auf die Funktionseinheit *kernel* ohne ASN.1-Encoding/Decoding.

Die Implementierung der ASN.1-Codierung kann prinzipiell auf zwei Arten erfolgen: in der Darstellungsschicht als Routine, die alle möglichen ASN.1-Datenstrukturen verarbeiten kann, oder mittels spezieller, ausschließlich auf die von der Anwendungsschicht verwendeten Datenstrukturen beschränkter Umwandlungsprozeduren. Letztere werden meistens aus ASN.1-Spezifikationen automatisch generiert. Im ersten Fall müssen die der Darstellungsschicht zur Laufzeit übergebenen Daten zusätzliche Typ-Informationen enthalten, um eine Codierung zu ermöglichen. Da dies im zweiten Fall entfallen kann, ist diese Variante wesentlich effizienter und wird auch in den meisten Implementierungen verwendet, so z. B. in ISODE [Ros90a] und in [Web91]. In diesem Fall werden die Codierungsroutinen von der Anwendung aus aufgerufen, und die Darstellungsschicht erhält die bereits umgewandelten Daten, die dann nur noch transparent übertragen werden müssen. Daher wurde in der vorliegenden Implementierung die ASN.1-Codierung nicht als interner Bestandteil der Darstellungsschicht angesehen.

Zunächst wurde ein Testrahmen erstellt, in den sowohl die Vergleichsspezifikation als auch die integrierte Spezifikation eingebettet werden können. Er ermöglicht Simulation und Test von Darstellungs- und Kommunikationssteuerungs-Instanzen zweier verschiedener Netzknoten. Der eigentliche Datentransfer geschieht durch ein Estelle-Modul, das den Transportdienst (Klasse 0) [ISO84b] beschreibt. Auf dieses setzen die beiden Kommunikationssteuerungs- und auf diesen wiederum die beiden Darstellungs-Instanzen auf. Die beiden Darstellungs-Dienstzugangspunkte werden durch das Modul `shell` dem Benutzer am Terminal zugänglich gemacht. Er kann also die Darstellungs-Dienstprimitive auf beiden Seiten abschicken und erhält ankommende Darstellungs-Dienstprimitive am Bildschirm angezeigt (s. Abb. 7a).

Im zweiten Schritt wurden die Darstellungs- und die Kommunikationssteuerungs-Module der einen Seite ersetzt durch das integrierte Darstellungs-/Kommunikationssteuerungs-Modul, die andere Seite dient als Referenz sowohl in Bezug auf Interoperabilität als auch für Laufzeiten (s. Abb. 7b). Für eine erste prototypische Spezifikation wurde der integrierte Automat zunächst nur auf Zeichenebene spezifiziert, d. h. es werden nur Dienstprimitive berücksichtigt, nicht jedoch ihre Parameter (ein Eingabezeichen im Sinne der formalen Beschreibung Endlicher Automaten entspricht einer eintreffenden Protokolldateneinheit bzw. einem eintreffenden Dienstprimitiv einer OSI-Spezifikation). Damit lassen sich erste

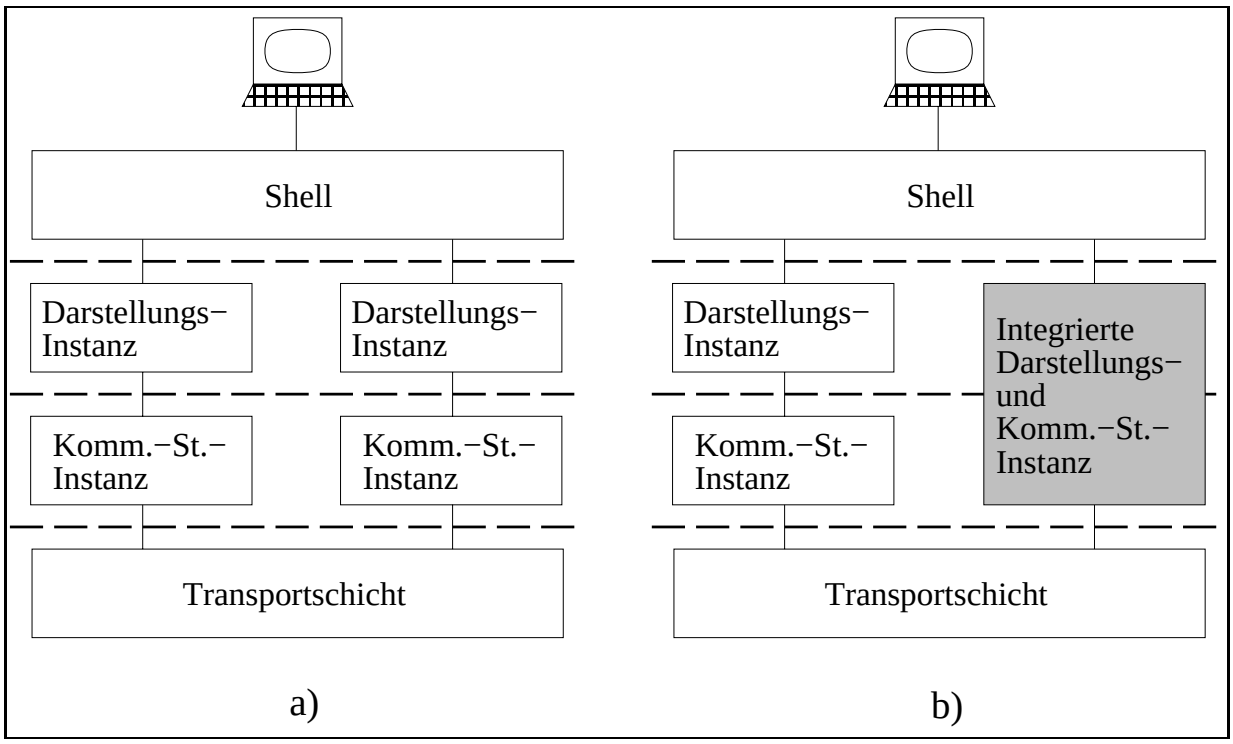

Abbildung 7: Testrahmen für die integrierte Spezifikation

Vergleiche mit der Referenzspezifikation ziehen, da hier ebenfalls die Parameterbehandlung ausgeblendet werden kann.

Für die Laufzeitmessungen wurde der NIST-Compiler erweitert [Lie92]. Er setzt jetzt in jede generierte Prozedur am Anfang und am Ende des Prozedurrumpfs Befehle, die einen Zeitstempel mit Angabe des Prozedurnamens auf Platte schreiben. Solche Befehle wurden auch in sämtliche Routinen des Laufzeitsystems eingefügt, so daß beim Ablaufen einer mit dem erweiterten Compiler übersetzten Spezifikation eine Datei erzeugt wird, in der sämtliche Prozeduraufrufe protokolliert sind. Daneben ist ein weiteres Werkzeug in Arbeit, das eine übersichtliche Auswertung dieser Protokolldatei in Form von Statistiken und Diagrammen erlaubt [Far92].

Das Schreiben der Zeitstempel beeinflußt zwar die Messung, bei einem Vergleich zweier Implementierungen werden jedoch beide gleichermaßen beeinflußt. So sind zwar keine exakten Angaben über die tatsächliche Laufzeit möglich, die Größenordnung der Effizienzsteigerung ist aber dennoch erkennbar.

Die Messungen zeigen, daß weitaus die meiste Zeit vom Scheduler des Estelle-Laufzeitsystems aufgebracht wird; die Ausführungszeiten der einzelnen Transitionen fallen demge-

genüber kaum ins Gewicht. Dies hat seine Ursache darin, daß der Scheduler als zentrale ausführende Einheit sämtliche Module nach ausführbaren Transitionen absuchen muß. Das bedeutet, daß Zustandstabellen durchsucht und Prädikate über Variablen berechnet werden müssen, um die schaltbaren Transitionen ermitteln zu können. Daher steigt der Aufwand des Schedulers mit der Anzahl der Transitionen einer Spezifikation.

Eine durch unseren Ansatz erreichte Reduzierung der Anzahl der Transitionen hat demnach nicht nur eine Vereinfachung der Transitionsroutinen im Automaten zur Folge, sondern verkürzt zugleich die von Scheduler zu durchsuchende Liste der aktivierbaren Routinen. Erste Messungen zeigen, daß der letztere Effekt erheblich größere Laufzeitverbesserungen bewirkt als der erstere.

Da die im Moment erstellte integrierte Spezifikation noch nicht ganz vollständig ist, sind zur Zeit noch keine genauen Zahlen verfügbar. Außerdem muß der Scheduler bei der vorliegenden Implementierung der Testanordnung als *ein* Betriebssystemprozeß die schaltbaren Transitionen aller Module ermitteln und führt so zu verfälschten Meßergebnissen. Es wird daher im Moment daran gearbeitet, die Testanordnung so aufzuteilen, daß die beiden Protokolltürme in zwei verschiedenen Prozessen ablaufen. Der Transportdienst wird dann von Interprozeßkommunikationsmechanismen (sockets) erbracht. Durch diese Anordnung sind dann separate Messungen der beiden Protokolltürme möglich, da dann ein Scheduler jeweils nur einen Protokollturm bearbeitet.

Es kann aber jetzt schon festgestellt werden, daß die Integration von Darstellungs- und Kommunikationssteuerungsschicht eine Effizienzsteigerung in der Größenordnung Faktor 2 bringt.

5 Zusammenfassung und Ausblick

In diesem Artikel wurde die integrierte Implementierung benachbarter Protokollinstanzen als Möglichkeit vorgeschlagen, effizientere Implementierungen von Kommunikationsprotokollen zu erzielen. Eine prototypische Implementierung der Darstellungs- und Kommunikationssteuerungsschicht zeigt, daß sich auf diesem Weg die Nachteile einer schichtenweisen Implementierung umgehen lassen. So sind einerseits weniger Zustandswechsel der Instanzen nötig, andererseits entfällt die gesamte Kommunikation der durch die Integration überbrückten Schnittstelle.

Wegen der noch ungenauen Messungen sind nur grobe Angaben über die zu erwartetende Effizienzsteigerung möglich, es läßt sich aber jetzt schon ein Speedup von mindestens 2 erkennen. Dieser Wert zeigt, daß die Integration ein Schritt in die richtige Richtung ist. Um genauere Zahlen zu erhalten, wird im Moment daran gearbeitet, die Testanordnung so aufzuteilen, daß die beiden Protokolltürme in zwei verschiedenen Prozessen ablaufen.

Daneben ist die Anbindung des integrierten Moduls an ISODE [Ros90a, Ros90b] in Arbeit. Dadurch sind weitere Tests auf Interoperabilität und Laufzeitvergleiche möglich.

Als Nachteil der Integration wäre zu nennen, daß es nun nicht mehr ohne weiteres möglich ist, eine Implementierung einer Schicht durch eine andere oder sogar durch ein vollkommen neues Protokoll zu ersetzen. Hier müssen bei der integrierten Implementierung alle beteiligten Schichten komplett ersetzt werden. Es ist daher geplant, ein Werkzeug zu entwickeln, das die Integration zweier Automaten vollautomatisch vornimmt. Als Basis dienen dabei die Estelle-Spezifikationen der einzelnen Automaten. Dann kann aus dem neuen Protokoll zusammen mit dem Protokoll der beizubehaltenden Schicht die neue, integrierte Schicht generiert werden.

Für die Zukunft ist vorgesehen, weitere Funktionseinheiten der verwendeten Protokolle, wie z. B. Synchronisation, zu implementieren. Dabei soll versucht werden, diese von dem eigentlichen Protokollautomaten zu trennen (s. [Sok91]) und als Prozedursammlung der Anwendung zur Verfügung zu stellen. Dadurch bleibt der Protokollautomat weitgehend unverändert, so daß keine Effizienzeinbußen durch hinzukommende Zustandswechsel etc. eintreten. Darüberhinaus soll versucht werden, ACSE [ISO88] in die Integration mit einzubeziehen. Auf diese Weise entsteht aus den anwendungsorientierten Schichten eine monolithische Implementierung, eine sog. Anwendungskomponente (s. a. [Zit90]), in der die Protokollaspekte sich im Automaten widerspiegeln und die Funktionseinheiten als Prozeduren verfügbar sind.

Danksagung

Meinen besonderen Dank möchte ich an dieser Stelle den Herren Prof. Dr. W. Effelsberg von der Universität Mannheim und Prof. Dr. H. Krumm von der Universität Dortmund ausprechen, die mir immer als kritische Diskussionspartner zur Verfügung standen. Dank gebührt auch Herrn S. Weber von der Universität Bern für die freundliche Überlassung der Estelle-Spezifikationen. Weiter möchte ich mich bei Frau C. Farrenkopf und Herrn R. Lienhart bedanken, deren tatkräftige Mitarbeit die Messungen erst ermöglichten sowie bei Herrn Dr. R. Gotzhein von der Universität Hamburg für die Weitergabe des NIST-Compilers.

Literatur

[BD87] S. Budkowski und P. Dembinski. An Introduction to Estelle: A Specification Language for Distributed Systems. *Computer Networks and ISDN Systems*, 14(1):3–23, 1987.

[Bra84] W. Brauer. *Automatentheorie.* B. G. Teubner, Stuttgart, 1984.

[FHSW89] J. Favreau, M. Hobbs, B. Strausser und A. Weinstein. User Guide for the NIST Prototype Compiler for Estelle. Interner Bericht No. ICST/SNA–87/3,

Institute for Computer Science and Technology, National Institute of Standards and Technology, February 1989.

[FLGL89] J.-P. Favreau, R. J. Linn, J. Gargulio und J. Lindley. A Test System for Implementations of FTAM/FTP Gateways. National Institute for Standards and Technology, USA, 1989.

[Far92] C. Farrenkopf. Implementierung eines Werkzeugs zur Analyse von Trace-Dateien. Studienarbeit, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 1992.

[Hof91] B. Hofmann. Analyse und Optimierung von Protokoll-Spezifikationen. In W. Effelsberg, H. W. Meuer und G. Müller, Hrsg., *Kommunikation in verteilten Systemen*, Seiten 568–584, Mannheim, Februar 1991. GI/ITG-Fachtagung, Springer Verlag.

[Hog89] D. Hogrefe. *Estelle, LOTOS und SDL. Standard-Spezifikationssprachen für Verteilte Systeme.* Springer Verlag, Berlin, Heidelberg, New York, 1989.

[ISO84a] Information processing systems – Open Systems Interconnection – Basic Reference Model. International Standard ISO 7498, 1984.

[ISO84b] Information processing systems – Open Systems Interconnection – Transport Service Specification. International Standard ISO 8072, 1984.

[ISO88] Information processing systems – Open Systems Interconnection – Protocol specification for the Association Control Service Element. International Standard ISO 8650, 1988.

[ISO89] Information processing systems – Open Systems Interconnection – Estelle: A formal description technique based on an extended state transition model. International Standard ISO 9074, 1989.

[KR83] B. W. Kernighan und D. M. Ritchie. *Programmieren in C.* Carl Hanser Verlag München Wien, 1983.

[Lie92] R. Lienhart. Erweiterung des NIST-Estelle-Compilers zur Analyse des Laufzeitverhaltens übersetzter Module. Studienarbeit, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 1992.

[MM89] P. Mondain-Monval. Estelle Description of the ISO Session Protocoll. In M. Diaz, J.-P. Ansart, J.-P. Courtiat, P. Azema und Vijaya Chari, Hrsg., *The Formal Description Technique Estelle*, Seiten 229–269. Elsevier Science Publishers B. V., Amsterdam, 1989.

[NIS87] Internals Guide for the NIST Prototype Compiler for Estelle. Report No. ICST/SNA–87/4, U.S. Department of Commerce, National Institute of Standards and Technology, September 1987.

[Ros90a] M. T. Rose. *The ISO Development Environment: User's Manual.* Performance Systems International, Inc., February 1990.

[Ros90b] M. T. Rose. *The Open Book.* Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[Sal69] A. Salomaa. *Theory of Automata.* Pergamon Press, Oxford, 1969.

[Sok91] Gerd Sokolies. Effiziente Implementierung der Kommunikationssteuerungsschicht des ISO/OSI-Referenzmodells. Diplomarbeit, Lehrstuhl für Informatik IV, Universität Dortmund, 1991.

[Svo89] L. Svobodova. Implementing OSI Systems. *IEEE Journal on Selected Areas in Communications*, 7(7):1115–1130, September 1989.

[Web91] S. Weber. Spezifikation und Implementation eines Datenkommunikationssystems mit Estelle. Diplomarbeit, Institut für Informatik und angewandte Mathematik, Universität Bern, April 1991.

[Zit90] M. Zitterbart. *Funktionsbezogene Parallelität in transportorientierten Kommunikationsprotokollen.* Dissertation, Universität Karlsruhe, 1990.