Mitteilung Nr. xxx

Entwurf eines parallelisierten OSI-TP in Estelle

Jan Bredereke[†]

FBI-HH-M-xxx/93

Dezember 1993

Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Straße 30

D-22527 Hamburg

[†]Die hier beschriebenen Arbeiten fanden statt im Rahmen des DFG-Forschungsprojektes Vo 543/1-1 „Estelle für Hochgeschwindigkeitsnetze“, Standort Universität Hamburg.

Inhaltsverzeichnis

1 Einleitung

Kommunikationsprotokolle für Rechnernetze stammen weitgehend aus der Zeit der Kupferkabel (Telefondrähte, Koaxialkabel). Sie wurden entworfen für ein physikalisches Medium, das durch eine geringe Übertragungsrate und eine hohe Bitfehlerrate gekennzeichnet war. Der Leistungsengpaß bei vernetzten Rechnern waren die Leitungen, so daß ein erheblicher Protokollaufwand in den Schichten 2 bis 7 des OSI-Basisreferenzmodells ([ISO81]) in Hardware oder in Software getrieben werden konnte, ohne dadurch den Datendurchsatz auf dem Netz zu gefährden. Dies gilt sowohl für herstellerspezifische Netze als auch für Netze nach ISO/OSI und für ISDN. In den nächsten Jahren werden sich sowohl im lokalen Bereich als auch im Weitverkehr schnelle, übertragungsfehlerarme Glasfasernetze mit physikalischen Übertragungsraten von 100 MBits/s bis 1 GBit/s durchsetzen. Weder die Kommunikationsprotokolle noch die Architekturen der heutigen Komunikations-Subsysteme sind geeignet, Daten in einer solchen Geschwindigkeit durch die Schichten 3 bis 7 hindurch der Anwendung verfügbar zu machen.

Die ebenso erwartete Steigerung der Leistung von Rechnern wird nicht ausreichen, um mit der erwarteten Steigerung bei der Leistung der Leitungen mitzuhalten. Ein möglicher Ausweg ist der Entwurf leichtgewichtiger Protkolle. Dieser Ansatz wird auch verfolgt, hat aber den Nachteil der Inkompatibilität mit den bisher bestehenden Kommunikationsprotokollen. Daher ist er kurzfristig kaum eine Hilfe. Eine weitere Möglichkeit besteht darin, die Übertragungsprotokolle innerhalb eines Systems parallel zu bearbeiten (vergleiche [FaMe81, Des86, Zit89b, Zit89c, Zit89a]). Dieser Ansatz wird in dem hier beschriebenen Projekt verfolgt.

Der Einsatz einer Formalen Beschreibungstechnik (FDT) erlaubt es aufgrund ihrer formal definierten Semantik, Kommunikationsprotokolle exakt zu beschreiben, wobei dies in aufgrund des hohen Abstraktionsgrades in einer problemnahen Form geschehen kann. Ferner eröffnet er die Möglichkeit, sowohl die Spezifikation als auch eine zugehörige Implementation zu validieren. In den letzten 10 Jahren hat im internationalen Umfeld intensive Forschung stattgefunden, die u.a. zur Entwicklung und Normung der FDTs Estelle ([ISO89b]), LOTOS ([ISO88c]) und SDL ([CCI87]) durch ISO und CCITT geführt hat. Ziel ist es, Protokollnormen künftig unter Verwendung dieser FDTs zu definieren und damit eine eindeutige Grundlage für deren Implementierung durch verschiedene Hersteller zu schaffen. Diese Voraussetzung ist gegenwärtig wegen der Verwendung natürlicher Sprache noch nicht erfüllt.

Estelle basiert auf dem Grundmodell der Erweiterten Endlichen Automaten. Eine Estelle-Spezifikation beschreibt ein hierarchisch strukturiertes System von nichtdeterministischen, sequentiellen Komponenten, den Modulinstanzen. Interaktion zwischen jeweils zwei Modulinstanzen findet über asynchrone Kanäle statt, die in der Spezifikation explizit definiert werden müssen. Systemstruktur und Verbindungsstruktur können dynamisch durch Erzeugen und Beenden von Modulinstanzen bzw. durch Aufbau und Abbau von Verbindungen verändert werden.

Die verteilte Transaktionsverarbeitung hat durch die zunehmende Verteilung von Verarbeitungsleistung eine zentrale Bedeutung erlangt. Für das ISO-Protokoll für verteilte Transaktionsverarbeitung in offenen Systemen (OSI-TP) wurde 1992 ein internationaler Standard verabschiedet ([ISO92]). Dabei wurden erstmals gleichzeitig mit den üblichen Dokumenten formale Spezifikationen des Protokolls in Estelle und in LOTOS erstellt, die als Anhänge veröffentlicht werden sollen.

Die Estelle-Spezifikation des OSI-TP-Protokolls ist mit ca. 10000 Zeilen die derzeit größte existierende Estelle-Spezifikation. Dieses Protokoll der Anwendungsschicht ist sehr komplex, auch in seiner internen Struktur von Komponenten, und bietet daher ein sehr geeignetes Feld und viel Potential für Untersuchungen zur Parallelisierung.

Für die prototypische Realisierung im Rahmen des Projekts wurde ein geeigneter Teil des sehr umfangreichen OSI-TP-Protokolls ausgewählt (siehe Kapitel 5.1). Es wurde eine Architektur für das OSI-TP-Protokoll gefunden, die eine parallele Bearbeitung in den einzelnen Komponenten ermöglicht, wobei während des Entwurfes dieser Architektur stets der Aspekt der effizienten Ausführbarkeit im Auge behalten wurde (Kapitel 4 und 5). Mit Hilfe eines geeigneten Werkzeuges (Kapitel 7.1) wurde aus der Estelle-Spezifikation automatisch eine Implementation generiert (Kapitel 7). Weiterhin wurde eine generische Last entworfen, die den OSI-TP-Dienst nutzt, um im weiteren Projektverlauf Messungen ausführen zu können (Kapitel 6).

Da es ein unmögliches Unterfangen wäre, OSI-TP auf wenigen Seiten hinreichend vollständig zu beschreiben, insbesondere auch in Bezug auf die jeweiligen Aufgaben der einzelnen Komponenten, können wir an dieser Stelle lediglich auf die einschlägige Literatur verweisen. Als ersten Einstieg kann man die sehr gut verständliche Einführung in [And91] empfehlen. Eine aktuelle, per definitionem vollständige, aber auch mühsamer zu lesende Beschreibung gibt dann der ISO-Standard für OSI-TP ([ISO92]). Daher möchten wir noch besonders auf unser Abkürzungsverzeichnis auf Seite 51 hinweisen, das die Lektüre dieses Berichtes sicherlich oft erleichtern kann, sowie auf die diversen Anhänge mit weiterer Information (vergleiche das Inhaltsverzeichnis).

Einen guten Überblick über die Formale Beschreibungstechnik Estelle gibt [DeBu89], das zugehörige Standarddokument ist [ISO89b].

2 Bewertung existierender Estelle-Spezifikationen

Aufgabenstellung des Projektes war, das *bereits existierende* Protokoll für OSI-TP in Estelle zu parallelisieren. Daher ist es wichtig, bevor wir zur Beschreibung der weiteren Entwurfsentscheidungen kommen, zuerst den Status und die Eignung der bereits bei Projektbeginn existierenden Estelle-Spezifikationen der verschiedenen

Protokolle und Komponenten zu beschreiben, die für die Arbeiten relevant sind.

2.1 Estelle-Spezifikation des TP-Protokolls

Hier waren zwei Versionen vorhanden. Parallel zur Entwicklung des TP-Standards wurde bei INRIA eine Estelle-Spezifikation des Protokolls entwickelt, wobei die Entwicklung der Estelle-Spezifikation wiederum auch eine Rückkoppelung für die Entwicklung des Standard-Dokumentes brachte ([AGS93], vergleiche auch [And91]). Diese Arbeit wurde mit Ende ihrer Finanzierung eingestellt, bevor der Standard fertiggestellt war.

Unanbhängig davon wurde im NIST eine weitere Spezifikation entwickelt, die als Anhang für das Standard-Dokument vorgesehen ist.

2.1.1 TP-Version von INRIA

Diese Version der Estelle-Spezifikation basiert auf einem inzwischen überholten Entwurf für den TP-Standard. In dem Dokument wurden bis zur Verabschiedung als ISO-Standard noch große, auch strukturelle Veränderungen vorgenommen[1], so daß die Struktur der Estelle-Spezifikation nicht mehr die Struktur des Standard-Dokumentes wiederspiegelt.

Insbesondere aber ist diese Version stark unvollständig. Es wurde im wesentlichen nur die MACF-Komponente der TP-Protokollmaschine spezifiziert. Bereits die Einzelassoziationsobjekte SAO samt ihrer inneren Struktur fehlen, die „SAO"s der Spezifikation führen nur eine direkte Abbildung von Dienstprimitiven auf Protokolldateneinheiten aus, um Tests zu ermöglichen. Selbstverständlich fehlt auch die Recovery-Komponente vollständig, das heißt die gesamte Kanal-Protokollmaschine sowie die mit ihr kommunizierenden MACF-Transitionen der TP-Protokollmaschine. Weiterhin fehlt die Konkatenations-/Separations-Komponente, die mehrere abgehende Nachrichten zu einer einzigen zusammenfaßt und ankommende wieder trennt. Die nicht zu TP gehörenden, aber von TP benötigten CCR- und ACSE-Anwendungsdienstelemente fehlen ebenfalls, bekommen aber über Dummy-Transitionen im Dummy-Einzelassoziationsobjekt SAO immerhin

[1]Beispiele: Viele Zustände wurden umbenannt, die Anzahl der Zustände wurde verringert. Der Zustand 8 etwa hatte in der alten Version mit dem Dialogabbau zu tun, in der endgültigen Version aber hat er mit dem Handshake zu tun. Die Zustandstabellen wurden also wesentlich umgestellt. Auch der Inhalt der Tabellen-Kästchen wurde anders organisiert. In der alten Version enthielt jedes Kästchen genau eine durchnumerierte Aktion. (Dies spiegelt sich auch in der Estelle-Spezifikation wieder: Jede der numerierten Aktionen steht in einer eigenen `include`-Datei, so daß gleiche Aktionen nicht im Spezifikationstext wiederholt werden müssen.) In der endgültigen Version enthält ein Kästchen i.a. mehrere Aktionen, die mit Langnamen bezeichnet sind. Man hat also die durchnumerierten Aktionen feiner aufgeteilt, gleiche Teilstücke zusammengefaßt und ihnen Namen gegeben. Eine weitere Änderung betrifft die Namen der Dienstprimitive, die in recht großem Umfang geändert wurden.

ihren Platz in der Architektur zugewiesen.[2]

Die architekturelle Strukturierung der MACF-Komponente der TP-Protokollmaschine erscheint dagegen gut gelöst. Der TP-Standard entfernt sich hier ein Stück weit vom Konzept der kommunizierenden erweiterten endlichen Automaten. Es gibt zwar einen Hauptzustand der MACF-Komponente, aber auch für jeden von ihr erzeugten Dialog einen eigenen erweiterten endlichen Automaten. Diese erweiterten endlichen Automaten arbeiten und kommunizieren meist asynchron, doch gibt es auch bestimmte Ereignisse (Zustandsübergänge), die nur eintreten können, wenn alle Dialog-Komponenten dazu bereit sind und es synchron tun. Diese Sonderregelungen werden in der Architektur von INRIA angemessen berücksichtigt, indem die Dialog-Automaten nicht als separate Estelle-Module, sondern als ein Feld von Datenstrukturen eines einzigen MACF-Moduls abgebildet werden. Ferner werden auch die Atomaritätsbedingungen beachtet, die der TP-Protokollmaschine auferlegt sind. Sie darf an einem ihrer äußeren Interaktionspunkte erst dann wieder eine Nachricht annehmen, wenn alle aufgrund einer vorangegangenen Nachricht notwendigen Aktionen vollständig ausgeführt worden sind.

2.1.2 TP-Version von NIST

Diese Version basiert auf dem von der ISO verabschiedeten Standard für TP, ist also auf einem aktuellen Stand. Auch ist sie (im Prinzip) vollständig.

Allerdings gibt es einige ernsthafte Kritikpunkte:

Die Struktur der Architektur ist unglücklich gewählt. Zur Verwaltung jedes Dialogs, den eine MACF-Komponente erzeugt, wird eine eigene Sohnmodulinstanz erzeugt. Die in Kapitel 2.1.1 erwähnte synchrone Kommunikation wird derart realisiert, daß die Vatermodulinstanz eine spontane Transition besitzt, die (in einem bestimmten Hauptzustand) mit einer `provided`-Klausel ständig bestimmte exportierte Variablen der Sohnmodulinstanzen beobachtet[3]. Bei Eintreten der erwarteten Bedingung werden dann an alle Söhne spezielle Synchronisationsnachrichten über gesonderte Nachrichtenkanäle versandt. Diese Art der Kommunikation bedeutet natürlich „busy waiting“ und ist damit unter dem Gesichtspunkt der Effizienz vollkommen unbrauchbar.

Wiederum innerhalb der Sohnmodulinstanzen, die je einen Dialog für die MACF-Komponente verwalten, sind die Einzelassoziationsobjekte SAO eingeschachtelt. Da aber in Estelle die Vater-Sohn-Beziehung zwischen Modulinstanzen statisch ist, läßt es sich bei dieser Abbildung der TP-Architektur nicht in Estelle darstellen, daß ein SAO von der MACF-Komponente freigegeben wird, einem Pool freier Assoziationen hinzugefügt wird und später einer anderen MACF zugeordnet wird. In diesen Fällen muß die SAO-Modulinstanz terminiert werden und

[2]Dies ist zum Beispiel wichtig für die im nächsten Absatz diskutierten Atomaritätsbedingungen.

[3]Siehe z.B. die Datei `tppm_m.e`, Zeile 757ff.

ihr Zustand in einer Datenstruktur gesichert werden, um später damit woanders wieder eine neue Modulinstanz zu erzeugen.

Eine weitere Kritik betrifft das TP-Anwendungsdienstelement TP-ASE. Konzeptuell ist es eine separate Komponente innerhalb eines Einzelassoziationsobjektes SAO, die bestimmte Umcodierungsaufgaben übernimmt. In der Spezifikation ist diese Funktionalität mit in der Steuerfunktion SACF des SAO aufgegangen. Anstelle der Umcodierung der von der SACF ausgesandten und empfangenen Nachrichten wurden gleich die entsprechenden Transitionen der SACF modifiziert, so daß sie bereits umcodierte Nachrichten aussenden und empfangen.

Schließlich kommen wir zum schwerwiegendsten Kritikpunkt. Die in Kapitel 2.1.1 beschriebene Atomaritätsbedingung für TP-Aktionen scheint nirgends beachtet worden zu sein. Dieser Punkt ist besonders gravierend, da somit ein korrektes Verhalten des spezifizierten Systems nicht mehr gewährleistet ist. Eine Behebung des Problems dürfte recht aufwendig sein, da vermutlich ein explizites Synchronisationsprotokoll eingeführt werden muß.

Im übrigen scheint, über eine Syntaxprüfung hinaus, keinerlei Validation der Spezifikation stattgefunden zu haben. Dies wird durch einige Beobachtungen nahegelegt. So fallen etliche Variablen auf, die zwar verwendet, aber nirgends initialisiert werden[4]. Weiterhin werden etliche Dinge als `primitive`-Funktion definiert, die in Estelle trivial auszuformulieren wären, wie etwa einige Mapping-Funktionen, die einen Parametersatz auf einen anderen abbilden.

Auch die Identifizierung der Dialoge erscheint fehlerhaft. Der TP-Standard schreibt vor, daß die einzelnen Dialoge mit lokalen Mitteln unterschieden werden können, damit z.B. eine TP-Nutzerinstanz (TPSUI) einen bestimmten Dialog gezielt ansprechen kann. Entsprechend wird in der NIST-Spezifikation in den betreffenden Nachrichten ein Parameter übergeben. Nun ist die Frage, welche Instanz diese Kennung erzeugen soll, wenn ein neuer Dialog eröffnet wird. Da die Dialogeröffnung ein möglicherweise unbestätigter Dienst ist, der auch mehrfach hintereinander zur Erzeugung mehrerer Dialoge aufgerufen werden kann, kann die TP-Protokollmaschine keine Kennung an die TPSUI zurückgeben. Also muß die TPSUI die Kennung erzeugen. In der NIST-Spezifikation wird hier zwar tatsächlich ein Parameter übergeben, aber sein Wert wird im folgenden ignoriert. Stattdessen erzeugt die MACF-Komponente einen eigenen Index zur internen Verwaltung, der aber auch später zur Kommunikation mit der TPSUI verwendet wird. Dies ist offensichtlich nicht korrekt und sollte bei jeder noch so einfachen Simulation sofort auffallen.[5]

[4] Z.B. `nullBranchId`, `nullRecoveryCntxtHndl`, ... Beabsichtigt war wohl die Definition von strukturierten Konstanten, bei denen einige Komponenten vom Typ „`...`“ sind, so daß sie durch `primitive`-Funktionen initialisiert werden müssen. Die Initialisierung wurde dann vermutlich vergessen.

[5] Eine sinnvolle Lösung wäre, die von der TPSUI erzeugten Kennungen zur Kommunikation mit der TPSUI zu verwenden, die Kennungen aber für den internen Gebrauch auf dafür geeignete, intern erzeugte und verwaltete Indizes hin und zurück abzubilden, wobei die Indizes zum

Es ist zu vermuten, daß weitere, ähnliche Fehler in der NIST-Spezifikation vorhanden sind.

Die beschriebenen Probleme scheinen auch im NIST bekannt zu sein, da das französische INT einen Mitarbeiter für ein Jahr an das NIST ausgeliehen hat, um die Spezifikation validieren zu lassen. Allerdings beschränkt sich diese Tätigkeit darauf, die im Anhang C des TP-Standards [ISO92] beschriebenen Szenarien nachzuvollziehen, wodurch die Architektur-Probleme vermutlich nicht bemerkt und behoben werden. Diese Arbeit soll Ende 1993 abgeschlossen sein.

2.2 Estelle-Spezifikationen zu den von TP benutzten Diensten

Damit die Estelle-Spezifikation des TP-Protokolls ausführbar gemacht werden kann und Messungen möglich werden, müssen auch die Protokolle für alle Dienste in Estelle spezifiziert sein, die das TP-Protokoll benutzt.

2.2.1 Assoziationssteuerungsdienst – ACSE

Es sind zwei Estelle-Spezifikationen des Assoziationssteuerungsdienstelementes (ACSE, [ISO88a, ISO88b]) vorhanden; sie sind miteinander „verwandt“. Für den FTAM-Tester des NIST wurde die erste ACSE-Version entwickelt. Diese Version wurde angepaßt und weiterentwickelt für die Spezifikation des Remote-Operations-Dienstelementes (ROSE).

Bei beiden Versionen ist die äußere Schnittstelle spezifisch an den jeweiligen Dienstaufrufer angepaßt, was sich aber recht leicht wiederum an die eigenen Bedürfnisse anpassen läßt. Zu der FTAM-Version existiert eine Testshell. Die ROSE-Version besteht nur aus einer Moduldefinition und den Typ- und Kanaldefinitionen, sie ist keine vollständige Estelle-Spezifikation. Dies sollte für eine Verwendung im Rahmen der TP-Spezifikation aber nicht schaden.

Die ROSE-Version wurde näher auf ihre Korrektheit untersucht. Offensichtlich ist die Spezifikation des Protokolls *nicht korrekt* in Bezug auf den ISO-Standard für ACSE. Sie hat einen Hauptzustand weniger, und aufgrund der dadurch fehlenden Transitionen verhält sie sich nicht korrekt, wenn es bei einem „weichen Verbindungsabbau“ zu einer Überkreuzung von Abbauwünschen beider Partner kommt. Die FTAM-Version wurde noch nicht näher betrachtet, aber zumindest fehlt ihr ebenso ein Hauptzustand.

Eine Korrektur und anschließendes Testen der korrigierten Version sind also erforderlich.

Management der Dialoge dienen.

2.2.2 Darstellungsdienst

Die ISO-Entwicklungsumgebung ISODE würde einen vollständigen Darstellungsdienst enthalten. Allerdings lief zum Zeitpunkt der Bewertung des Darstellungsdienstes[6] erst eine Studienarbeit an der Universität Mannheim, die eine Anbindung von ISODE an Estelle erreichen soll. Daher wurde ISODE nicht weiter berücksichtigt.

Über die Universität Mannheim wurden Spezifikationen der Protokolle der Darstellungs- und der Sitzungsschicht beschafft. Es existieren dort insgesamt drei Versionen:

- Beide Protokollmaschinen sind als eigenes Estelle-Modul spezifiziert. Die ASN.1-Umwandlung in der Darstellungsschicht wurde einst mit dem MAVROS-Compiler erstellt, ist inzwischen aber auskommentiert. Es findet also keine ASN.1-Umwandlung mehr statt.[7]

- Beide Protokollmaschinen sind als eigenes Estelle-Modul spezifiziert. In der Darstellungsschicht wurde die ASN.1-Umwandlung mit dem ISODE-Werkzeug PEPSY erstellt. Die Parameter des Darstellungsdienstes werden kodiert, lediglich das Nutzerdatenfeld wird unverändert als String übertragen.

- Eine Variante der vorigen Version, bei der beide Protokollmaschinen in eine einzige Protokollmaschine verschmolzen sind. Diese Version wurde von der Universität Mannheim erzeugt, sie ist aufgrund der eingesparten internen Kommunikation zwischen den Maschinen etwas effizienter als die beiden vorigen Versionen ([Hof93]).

Der Sitzungsdienst gliedert sich in verschiedene Funktionseinheiten (FUs), die in den ISO-Standards ISO 8326 und ISO 8326/Amd4 definiert werden ([ISO87]). In allen drei obigen Versionen wird bisher nur die Minimalversion mit der Kernel- und der Duplex-FU realisiert. Dem Dienst der unterliegenden Transportschicht wird keine weitere Funktionalität hinzugefügt, er wird im wesentlichen lediglich durchgereicht. Nichtsdestotrotz besitzt die Protokollmaschine bereits eine ganze Reihe verschiedener Zustände, um den aktuellen Zustand der Verbindung verfolgen zu können. Laut Auskunft aus Mannheim ist es relativ einfach möglich, weitere Funktionseinheiten zu ergänzen. Hierfür wären dann weitere Zustände und Transitionen zu ergänzen.

Für das TP-Protokoll ist die Minimalversion des Sitzungsdienstes ausreichend, solange nicht die Commit-FU des TP-Dienstes realisiert wird. Dann werden (mindestens) alle FUs des Sitzungsdienstes notwendig, die das CCR-Protokoll (in der Version 2) benötigt, also die Minor-Synchronize- und die Resynchronize-FU. Einen Überblick gibt Tabelle 1.

---

[6] Juli '93

[7] Diese Version ist in Hamburg vorhanden.

Funktionseinheit	vorhanden	TP braucht	
		ohne Commit	mit Commit
Kernel	•	•	•
Duplex	•	•	•
Halbduplex			
Negotiated Release			
Expedited Data			
Minor Synchronize			•
Major Synchronize			
Resynchronize			•
Activity Management			

Tabelle 1: Die Funktionseinheiten des Sitzungsdienstes

Soweit bekannt ist, wurde die Korrektheit der Estelle-Spezifikationen des Darstellungs- und Sitzungsprotokolls bisher nicht validiert, zum Beispiel durch Tests. Allerdings sollte dies eigentlich auch kein großes Problem darstellen, da diese Protokolle (bisher) keine wesentliche Funktionalität zum unterliegenden Dienst hinzufügen.

Fehlerfrei können diese Spezifikationen dennoch nicht sein, da zumindest eine nicht initialisierte Variable bereits entdeckt wurde.[8] (Bei den Mannheimer Arbeiten fiel sie vermutlich deshalb nicht auf, weil das dortige Estelle-Werkzeug sie offensichtlich automatisch mit dem richtigen Wert initialisiert hat.)

2.2.3 Transportdienst

Dieser Dienst wird nicht vom TP-Protokoll direkt genutzt, sondern nur vom Protokoll der Sitzungsschicht. Diese wird aber über die Darstellungsschicht schließlich doch von den Protokollen der Anwendungsschicht genutzt, so daß auch der Transportdienst benötigt wird.

Es existiert an der Universität Mannheim eine Implementation außerhalb von Estelle, die die Funktionalität als Estelle-`primitive`-Funktionen zur Verfügung stellt. Die Kommunikation findet dort über Unix-Sockets statt. In Augenschein genommen wurde diese Implementation noch nicht. Nach Mannheimer Einschätzung sollte diese Implementation nur in der Form verwendet werden, in der sie zur Zeit vorliegt, da sich die schlechte interne Strukturierung als ein Hindernis für eventuelle Modifikationen erweisen dürfte.

[8] Es handelt sich um die boolesche Variable `protocol_error`. Hat sie zu Beginn den Wert `true`, so löst die Protokollmaschine sofort eine Protokollfehlerbehandlung aus.

2.2.4 Weitere, gänzlich fehlende Spezifikationen

Eine ganze Reihe von TP-Komponenten und Dienstelementen, die von TP benötigt werden, sind noch gar nicht vorhanden. Hierbei handelt es sich um:

CCR-ASE. Das Dienstelement für die Durchführung des Commit.
(Es existiert eine Estelle-Spezifikation für eine ältere Version von CCR aus der Diplomarbeit [And91]. Die Definition von CCR hat sich seitdem allerdings erheblich verändert.)

Routing-Komponente des TP-SAO. Die Komponente des SAO, die die Nachrichten, die am Darstellungs-Interaktionspunkt ankommen, an die richtige SAO-Komponente weiterverteilt.

Konkatenations-/Separationskomponente des TP-SAO. Die Komponente des SAO, die mehrere abgehende Nachrichten zu einer einzigen zusammenfaßt und ankommende wieder trennt. (Durch die Zusammenfassung wird viel Kommunikationsoverhead in den niederen Schichten eingespart.)

Initialisierungskomponente des TP-SAO. Die Komponente des SAO, die unmittelbar nach der Erzeugung des SAO die Assoziation aufbaut, indem sie das ACSE-Dienstelement entsprechend ansteuert. Ebenso muß unmittelbar von Beendigung des SAO wiederum ACSE angesteuert werden, um die Assoziation abzubauen.

U-ASE. Ein Dienstelement, das spezifisch für den jeweiligen TP-Dienst-Nutzer ist.

(TP-ASE) Ein Dienstelement, das innerhalb eines Single Association Objects (SAO) eine einfache Abbildung von Dienstprimitiven auf Protokolldateneinheiten (und zurück) durchführt. Die Funktionalität ist zwar bereits in der NIST-Version der TP-Spezifikation vorhanden, aber nur aufgelöst in den Transitionen der SACF-Komponente. (Vergleiche Kapitel 2.1.2)

3 Einbettung in das OSI-Basisreferenzmodell

Bei dem Entwurf der Architektur für ein parallelisiertes TP wurde auch untersucht, wie sich diese Architektur in das ISO-Basisreferenzmodell ([ISO81]) einordnen läßt. Der ISO-Standard für die Struktur der Anwendungsschicht [ISO89a] (ALS-Standard; siehe auch Kapitel 3.1) legt die Struktur für eine solche Einbettung fest. Eine Vergleich zeigte, daß sich im ISO-Standard für TP die im ALS-Standard definierten Begriffe genau wiederfinden.

Abbildung 1 zeigt die Details. Die gesamte *Welt* enthält eine (feste) Anzahl von *offenen Systemen*. Auf jedem von diesen läuft wiederum eine (veränderliche)

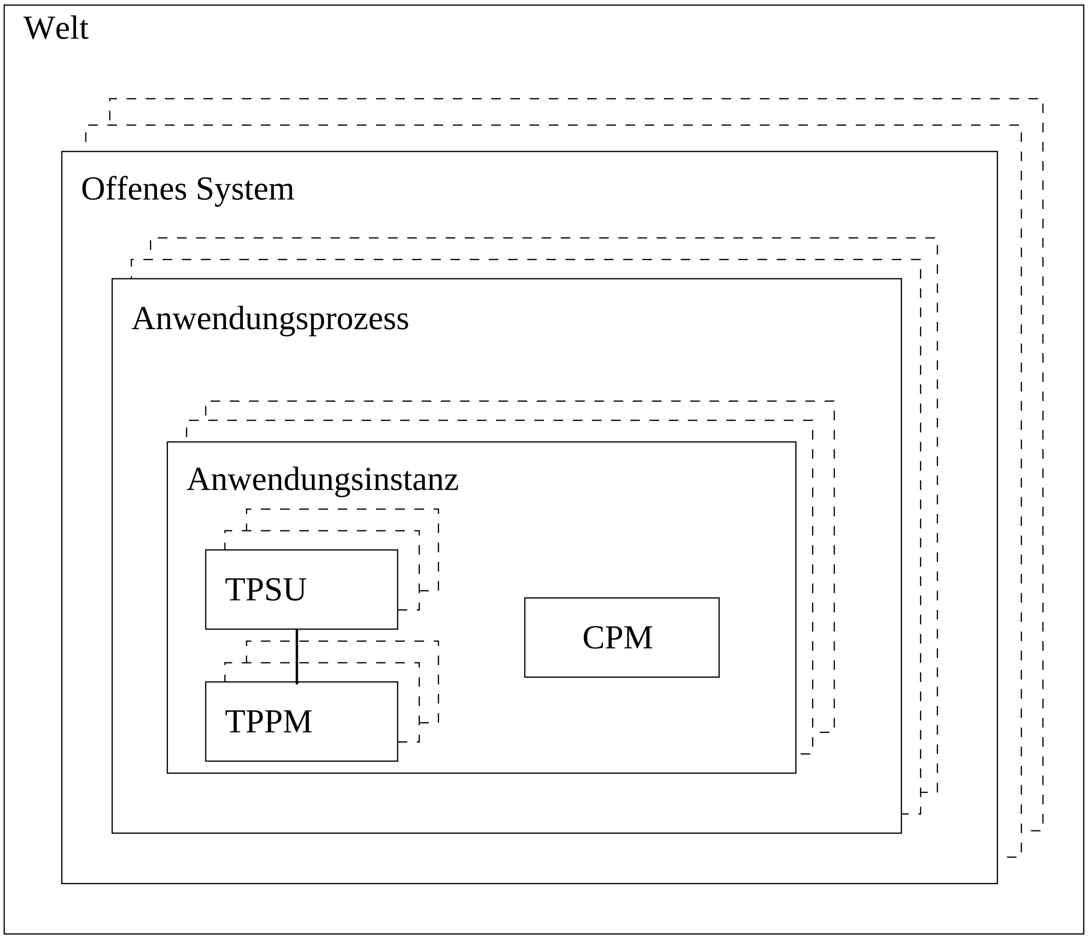


Abbildung 1: Die Einbettung von TP in das ISO-Basisreferenzmodell in der Anwendungsschichtstruktur-Sicht.

Anzahl von *Anwendungsprozessen* ab. Jeder *Anwendungsprozeß* kann für Zwecke der Kommunikation mit anderen *Anwendungsprozessen Anwendungsinstanzen* enthalten. Im Falle von TP kann eine solche eine beliebige Anzahl von Paaren aus TP-Dienstnutzerinstanzen *TPSUI* und TP-Protokollmaschineninstanzen *TPPM* enthalten, sowie genau eine Instanz einer Kanalprotokollmaschine *CPM*. (Die CPM ist für das Recovery von Transaktionen zuständig.)

3.1 ALS vs. XALS

Die Struktur der Anwendungsschicht im ISO-Basisreferenzmodell ([ISO81]) wird im ALS-Standard ([ISO89a], application layer structure) festgelegt. Entsprechend wurde dieser Standard mit seiner Begriffswelt bei der Entwicklung der (inneren) Struktur von TP zugrundegelegt. Während dieser Entwicklung stellte sich heraus, daß seine Begriffe für TP nicht ausreichend waren, so daß in der Folge mit einer Überarbeitung und Erweiterung begonnen wurde, die zum XALS-Standard ([ISO91a], extended application layer structure) führen soll. Während der Entwicklung von TP selbst stand diese Erweiterung aber noch nicht zur Verfügung, so daß, so gut es ging, der ALS-Standard verwendet wurde. Auf seiner Begriffswelt baut der Standard für TP auf.

Im XALS-Standard werden als Verallgemeinerung die zwei neuen Konzepte des Anwendungsdienstobjektes ASO (application service object) und der Steuerfunktion CF (control function) eingeführt, dafür entfallen die bisherigen, spezielleren Konzepte des Einzelassoziationsobjektes SAO (single association object), der Mehrassoziationssteuerfunktion MACF (multiple association control function) und der Einzelassoziationssteuerfunktion SACF (single association control function).

Ein ASO ist eine Verallgemeinerung eines SAO und eines Anwendungsdienstelementes ASE. Eine Anwendungsinstanz AE enthält ein oder mehrere ASOs. ASOs selbst sind rekursiv aufgebaut. Sie enthalten ein oder mehrere ASEs und/oder weitere ASOs, dazu enthalten sie eine Steuerfunktion CF.[9]

Eine Steuerfunktion CF ist eine Zusammenfassung der Konzepte der Mehrassoziationssteuerfunktion MACF und der Einzelassoziationssteuerfunktion SACF. Weiterhin steuert sie nun nicht nur ASEs, sondern auch ASOs.

Für die Entwicklung der Struktur des parallelisierten TP wurde entschieden, sich nicht am Entwurf für den XALS-Standard zu orientieren, sondern am ALS-Standard. Erstens basiert die gesamte Begriffswelt im TP-Standard auf dem ALS-Standard, so daß eine umfangreiche Umbenennung mit der Gefahr von Verwechslungen bei späteren Vergleichen notwendig gewesen wäre. Dies betrifft ebenso auch die Namensgebung in den bereits existierenden Estelle-Spezifikationen.

[9] Die einfachen Fälle, in denen bereits die Anwendungsinstanz AE eine Steuerfunktion CF und/oder Anwendungsdienstelemente ASE besitzt, werden dadurch beschrieben, daß das AE dann nur genau ein ASO enthält und man die AE mit ihrem ASO identifiziert.

Und zweitens stand XALS noch nicht in einer genügend ausgereiften Version zur Verfügung.

Die Architektur des bestehenden TP-Standards selbst läßt sich recht gut mit dem ALS-Standard zur Deckung bringen, sofern man davon absieht, daß der TP-Dienstnutzer TPSU auch ein Teil der Anwendungsinstanz AE sein muß.[10]

3.2 Spezifikation der TP-Einbettung in Estelle

Die Einbettung von TP in das OSI-Basisreferenzmodell aus Abbildung 1 läßt sich vollkommen direkt in Estelle übertragen. Abbildung 2 zeigt die gewählte Estelle-Modulstruktur. Man vergleiche die beiden Abbildungen. (Die innere Struktur von TP ist in Abbildung 2 noch nicht dargestellt, da wir in Kapitel 5 ausführlich darauf eingehen werden.)

4 Die Architektur von TP

Die konzeptuelle Architektur von TP läßt sich aus zwei orthogonalen Sichten beschreiben, aus der Dienst-Sicht und aus der Sicht der Anwendungsschichtstruktur des OSI-Basisreferenzmodells.

Die Dienst-Sicht ist in Abbildung 3 beschrieben. Es gibt mehrere TP-Dienstnutzerinstanzen TPSUI, die die Dienste des TP-Diensterbringers TPSP nutzen. Der TP-Diensterbringer selbst ist wiederum in mehrere Komponenten strukturiert, die auf dem Dienst der tieferen Schichten 1 bis 6 des ISO-Basisreferenzmodells aufbauen. Die räumliche Verteilung spielt in dieser Sicht nur insofern eine Rolle, als daß die Komponenten des TP-Diensterbringers, die einen einzelnen TP-Dienstnutzer bedienen, untereinander nur über die Kommunikationsdienste der tieferen Schichten verbunden sind, und nicht direkt kommunizieren können.

Die Sicht der Anwendungsschichtstruktur hatten wir bereits in Kapitel 3 (vergleiche Abbildung 1) eingenommen, als wir die Einbettung von TP in das OSI-Basisreferenzmodell beschrieben haben. In Abbildung 4 ist die Struktur einer *Anwendungsinstanz* nun noch einmal feiner aufgeschlüsselt. Man erkennt, daß die beiden Protokollmaschinen TPPM und CPM weiter untergliedert sind in jeweils genau eine Mehrassoziationssteuerfunktion *MACF* (multiple association control function) und eine veränderliche Anzahl von Einzelassoziationsobjekten *SAO* (single association object). Die SAOs enthalten wiederum eine Einzelassoziationssteuerfunktion *SACF* (single association control function) und mehrere Anwendungsdienstelemente (ASEs).

[10] Der TP-Dienst ist nur innerhalb der AE verfügbar, so daß der Nutzer auch Teil der AE sein muß. Dies läßt sich aber dadurch rechtfertigen, daß der TPSU derjenige *Teil* eines Anwendungsprozesses ist, der Transaktionsverarbeitung durchführt, und der deshalb aufgrund seines Kommunikationsbedarfs in die AE fällt.

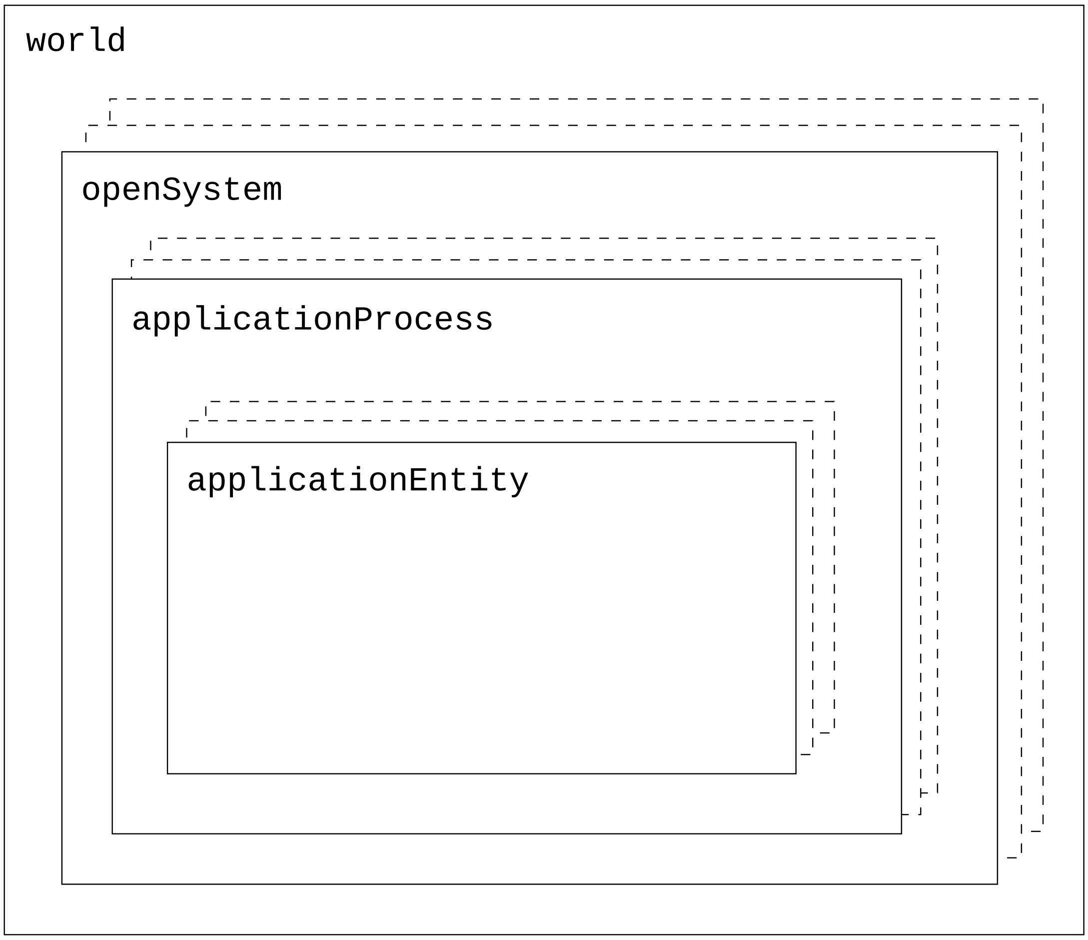


Abbildung 2: Die Estelle-Modulstruktur, die die Einbettung von TP in das OSI-Basisreferenzmodell abbildet.

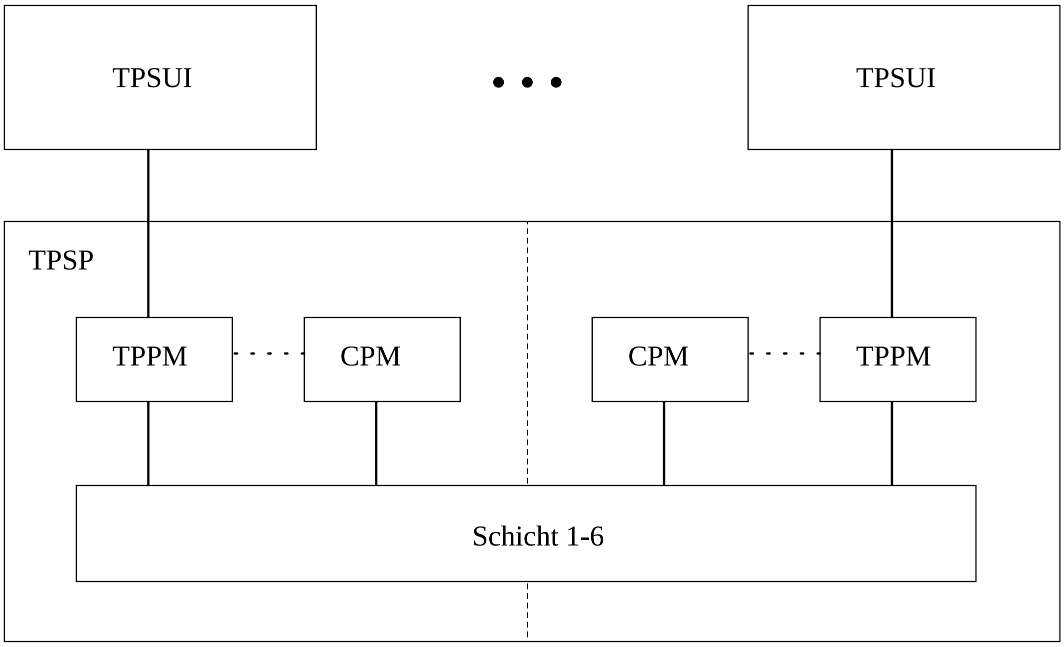


Abbildung 3: Die Dienst-Sicht auf die Architektur von TP.

5 Entwurf der parallelisierten TP-Architektur

In diesem Kapitel werden die Entwurfsentscheidungen beschrieben, die bei dem Entwurf für ein parallelisiertes TP getroffen wurden. In Kapitel 5.1 wird eine Teilmenge der Funktionalität von TP für eine erste Implementierungsphase ausgewählt, in Kapitel 5.2 werden die möglichen Kandidaten für eine Parallelisierung identifiziert, in Kapitel 5.3 wird auf einige Besonderheiten der Zustandsmaschinen eingegangen, die innerhalb von TP vorkommen, in Kapitel 5.4 wird die parallele Architektur auf Estelle abgebildet, in Kapitel 5.5 wird untersucht, wieviel Nebenläufigkeit in dieser abgebildeten Architektur möglich ist, in Kapitel 5.6 wird auf die Integration der Darstellungs- und der Sitzungsschicht in die Estelle-Spezifikation eingegangen, in Kapitel 5.7 werden die Anwendungsdienstelemente (ASEs) betrachtet, und in Kapitel 5.8 wird dargestellt, inwieweit bereits vorhandene Estelle-Spezifikationen verwendet werden konnten.

5.1 Auswahl einer Teilmenge der Funktionalität

Um in der ersten Phase des Projektes bereits eine ablauffähige Version des parallelisierten TP zu erhalten, wurde für diesen ersten Schritt eine in sich geschlossene Teilmenge der Funktionalität von TP ausgewählt. TP ist ohnehin in mehrere Funktionseinheiten (FUs, functional units) aufgeteilt, die nicht alle in jeder Implementation vorhanden sein müssen, und deren Bereitstellung bei dem Aufbau einer Assoziation über den Anwendungskontext ausgehandelt wird. Die einzelnen

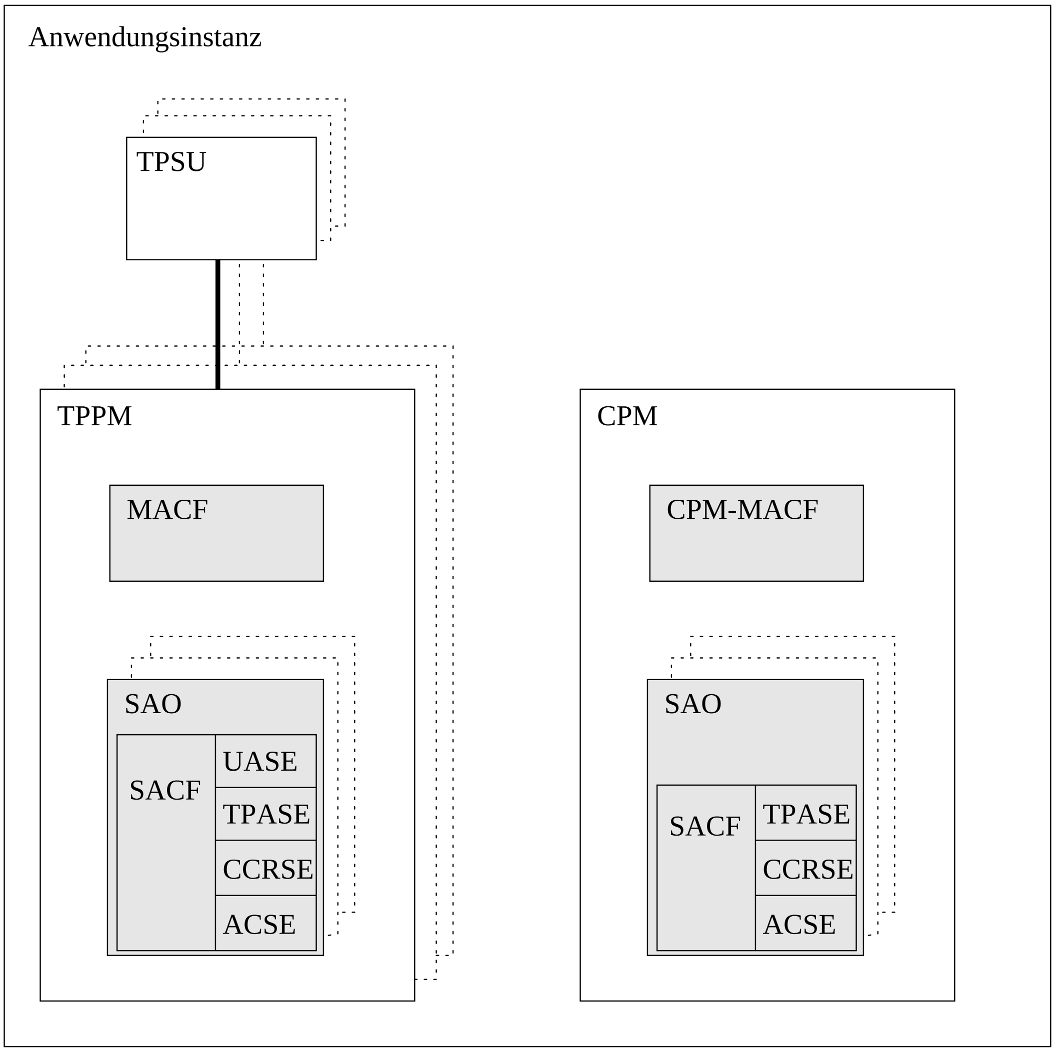


Abbildung 4: Die Anwendungsschichtstruktur-Sicht auf die Architektur von TP, Feinstruktur.

Funktionseinheiten sind in Tabelle 2 aufgeführt. Wie man dort erkennen kann, ist von den optionalen Funktionseinheiten die Commit-FU mit Abstand die umfangreichste. Daher wurde beschlossen, die Commit-FU in Phase 1 des Projektes nicht zu realisieren, sondern nur den Auf- und Abbau von Dialogen und die nicht transaktionsgesicherte Übertragung von Daten.

Funktionseinheit	notwendig	Anzahl der Dienstelemente
Dialogue	•	5
Shared Control	} •	0
Polarized Control		2
Handshake		2
Commit		10
Chained Transactions		0
Unchained Transactions		1

Tabelle 2: Die Funktionseinheiten von TP

5.2 Die Komponenten von TP

Um die Architektur von TP parallelisieren zu können, mußten zuerst einmal sinnvoll unterscheidbare Komponenten von TP identifiziert werden, damit Kandidaten für die Parallelisierung zur Verfügung standen. In Tabelle 3 sind noch einmal die Komponenten dargestellt, die sich aus der Einbettung von TP in das OSI-Basisreferenzmodell ergeben. Sie alle können offensichtlich völlig nebenläufig arbeiten. In Tabelle 4 sind dann die eigentlichen Komponenten von TP aufgeführt, mit einer kurzen Beschreibung ihrer Aufgabe.[11] Inwieweit zwischen ihnen Nebenläufigkeit möglich ist, wird dann in Kapitel 5.5 untersucht.

5.3 Die Zustandsmaschinen

Bei der Parallelisierung von TP waren einige Besonderheiten in den Zustandsmaschinen zu beachten, die der TP-Protokoll-Standard ([ISO92]) definiert, da diese die Architektur des parallelisierten TPs teilweise erheblich beeinflußten. Hier folgt nun eine Aufstellung der besonderen Eigenschaften aller im TP-Protokoll-Standard definierten Zustandsmaschinen; wir werden uns auf sie im weiteren immer wieder beziehen.

MACF Die Mehrassoziationssteuerfunktion MACF der TP-Protokollmaschine TPPM enthält pro aufgebautem Dialog je einen erweiterten endlichen Auto-

[11] In Tabelle 4 wird keine Unterscheidung zwischen einer Instanz und den Vorkommnissen dazu gemacht, außer bei TPSU/TPSUI.

Komponente	Ort/Teil von	Aufgabe
Offenes System	Welt	(ist ein System von Rechnern, Software, Peripherie und Übertragungsmedien, das einem Satz von Standards für den Informationsaustausch mit anderen solchen Systemen gehorcht. ([Ker92]))
Anwendungsprozeß	Offenes System	hängt von Anwendung ab
AEI (Anwendungsvorkommnis)	Anwendungsprozeß	erbringt OSI-Dienste für den Anwendungsprozeß

Tabelle 3: Die Komponenten in der Einbettung von TP

maten. Diese Automaten sind gekoppelt durch spezielle Zustandsübergänge, die sie nur alle gemeinsam und synchron ausführen können. Jeder der Automaten besitzt als erweiterten Zustand einen eigenen Satz von Variablen, dazu kommt noch ein Satz Variablen, der allen Automaten einer MACF gemeinsam ist. Schließlich gibt es auch noch die (dauerhaften) Log-Daten, die sogar allen MACF und der CPM-MACF eines Anwendungsvorkommnisses gemeinsam sind.

CPM-MACF Für die Mehrassoziationssteuerfunktion CPM-MACF der Kanalprotokollmaschine CPM gilt das gleiche wie für die MACF der TPPM, nur daß sie statt über Dialoge über Kanäle kommuniziert.

SACF Die Einzelassoziationssteuerfunktion SACF besitzt einen erweiterten endlichen Automaten. Die Zuordnung eines Einzelassoziationsobjektes SAO zu einer TP- oder Kanalprotokollmaschine kann sich im Laufe der Zeit ändern, ein SAO kann auch keiner der Protokollmaschinen zugeordnet sein. Ist das SAO aber einer Protokokollmaschine zugeordnet, so kann die (CPM-) MACF der aktuell verbundenen Protokollmaschine auf einige Variablen des erweiterten endlichen Automaten der SACF des SAO zugreifen.

TP-ASE Die Zustandsmaschine des TP-Anwendungsdienstelementes TP-ASE besitzt nur einen einzigen Zustand und keine Variablen. Sie besitzt also keine Zustandsinformation, es besteht ein funktionaler Zusammenhang zwischen Ein- und Ausgaben.

U-ASE Der Aufbau des TP-Benutzer-Dienstelementes U-ASE ist benutzerspezifisch.

ACSE Das Assoziationssteuerungsdienstelement ACSE enthält als Zustandsmaschine einen endlichen Automaten ohne erweiterten Zustand, also ohne Variablen. Der endliche Automat hat die Besonderheit, daß er terminiert wird,

Komponente	**Ort/Teil von**	**Aufgabe**
TPSU/TPSUI	AEI (Anwendungsvorkommnis)	hängt von Anwendung ab (nutzt den TP-Dienst)
TPPM	AEI	erbringt (in Zusammenarbeit mit anderen TPPMs) den TP-Dienst für eine TPSUI
CPM	AEI	führt das Recovery für das AEI durch: erzeugt und startet die TPSUIs und TPPMs nach einem Crash, unterstützt danach die TPPMs bei dem Wiederaufsetzen
MACF	TPPM	koordiniert die Kommunikation auf den verschiedenen Assoziationen einer TPPM
CPM-MACF	CPM	koordiniert die Kommunikation auf den verschiedenen Assoziationen einer CPM
SAO	TPPM oder CPM oder frei (in AEI)	beinhaltet alles, was eine einzelne Assoziation betrifft (für ein AEI)
ASE	SAO	liefert einen (Teil-)Dienst in der Anwendungsschicht
SACF	SAO	koordiniert die Kommunikation der ASEs des SAO (d.h. für eine Assoziation)
SACF-PM	SACF	steuert den Ablauf innerhalb des SAO
SACF-Router	SACF	leitet im SAO die ankommenden Nachrichten an die richtige Komponente
SACF-Konkatenation	SACF	faßt mehrere Nachrichten an die Darstellungsschicht zu einer zusammen und trennt entsprechend ankommende Nachrichten
TP-ASE	SAO	ASE, die das TP-Protokoll für eine Assoziation abwickelt
U-ASE	SAO	ASE, die einen benutzerspezifischen Dienst liefert, der in die Transaktionsverarbeitung eingebunden ist
CCR-ASE	SAO	ASE, die den CCR-Dienst liefert
ACSE	SAO	ASE, die den Auf- und Abbau einer Assoziation verwaltet
P-SAP/tiefere Schichten	—	ermöglicht TP die Nutzung des Darstellungsschicht-Kommunikationsdienstes

Tabelle 4: Die Komponenten von TP

sobald er in seinen Anfangszustand zurückkehrt. Eine weitere Aufforderung zum Aufbau einer Assoziation wird daher von einem dafür neu erzeugten Automaten bearbeitet. Eine Konsequenz daraus ist unter anderem, daß nach einem Assoziationsabbau noch im Kommunikationsmedium befindliche weitere Nachrichten nicht mehr zugestellt werden können, insbesondere nicht an einen eventuell neu erzeugten Automaten für eine neue Assoziation. Vergleiche auch [Lap94b].

CCR-ASE Das Dienstelement für Concurrency, Commitment und Recovery CCR-ASE wird in Phase 1 des Projektes nicht benötigt, da die Commit-Funktionseinheit von TP noch nicht realisiert werden soll. Die Zustandsmaschine von CCR-ASE dürfte aber keine Besonderheiten aufweisen.

5.4 Abbildung auf Estelle

5.4.1 Zugang zum Kommunikationsmedium

In Kapitel 4 haben wir bereits gesagt, daß man die Architektur von TP aus zwei orthogonalen Sichten sehen kann, der Dienst-Sicht und der Anwendungsschichtstruktur-Sicht (vergleiche die Abbildung 3 gegenüber den Abbildungen 4 und 1). Entsprechend muß man sich bei der Wahl einer Architektur für ein parallelisiertes TP für eine der beiden Sichten entscheiden. Da es in unserem Falle nicht das Ziel war, die Eigenschaften des TP-Dienstes zu untersuchen, sondern das TP-Protokoll mit seiner wichtigen Eigenschaft der Verteiltheit der Protokollinstanzen abzubilden und zu optimieren, fiel die Entscheidung eindeutig gegen die Dienst-Sicht und für die Anwendungsschichtstruktur-Sicht.

Diese Entscheidung hat eine wichtige Konsequenz: In der Dienst-Sicht ist das Kommunikationsmedium explizit vorhanden, es zieht sich als Basis durch das gesamte Bild. In der Anwendungsschichtstruktur-Sicht kommt es nicht explizit vor, da es nicht zur Anwendungsschicht gehört (und auch gar nicht einem einzelnen Anwendungsvorkommnis zuzuordnen wäre). Das einzige, was man in dieser Sicht noch betrachten könnte, wären die Dienstzugangspunkte P-SAP der tieferen Schichten.

Wollte man diese P-SAPs innerhalb der Estelle-Spezifikation verbinden, so müßte man dies mit Estelle-Kanälen tun. Dabei würde sich einerseits das Problem stellen, daß man diese zwischen zwei jeweils sehr tief eingeschachtelten Estelle-Modulen definieren müßte, was direkt nicht möglich ist, und andererseits hätten die Estelle-Kanäle auch ganz andere Eigenschaften als das zu spezifizierende Kommunikationsmedium. Dies wird insbesondere dann zum Problem, wenn man Leistungsmessungen durchführen will, weil man dann diese anderen Eigenschaften aufwendig und ineffektiv mit weiteren Estelle-Modulen simulieren müßte.

Somit haben wir in der parallelisierten Estelle-Spezifikation von TP das Kommunikationsmedium *nicht explizit modelliert*, sondern lediglich die Dienstzugangspunkte der tieferen Schichten spezifiziert. Hierfür bietet Estelle einerseits das Konzept der `primitive`-Funktion und andererseits das des `external`-Moduls an.

Bei der Auswahl eines der beiden Konzepte war zu beachten, daß in der endgültigen Estelle-Spezifikation nicht nur das TP-Protokoll in Schicht 7 spezifiziert sein sollte, sondern alle Protokolle der oberen Schichten 5–7. Also mußten auch die Estelle-Spezifikationen der Protokolle der Schichten 5 und 6 integriert werden.

Solange allerdings die Commit-FU von TP noch nicht realisiert wird, liefern die Schichten 5 und 6 lediglich einen Nulldienst, sie stellen jeweils nur den Dienst der unterliegenden Schicht unverändert zur Verfügung. Daher wurde beschlossen, in der Phase 1 des Projektes die Protokolle dieser Schichten vorläufig noch nicht zu integrieren.

Das Konzept der `primitive`-Funktion ist gut geeignet, um einen effizienten Anschluß einer ausführbaren Estelle-Spezifikation an Prozeduren in einer Programmiersprache zu ermöglichen, die zum Beispiel die Verbindung zum Betriebssystem herstellen. Das Konzept des `external`-Moduls dagegen bietet keine ganz so effiziente Anbindung, da es zusätzliche Kommunikation über Estelle-Kanäle erfordert. Dafür erlaubt es, zu einem späteren Zeitpunkt das `external`-Modul durch ein in Estelle ausspezifiziertes Modul zu ersetzen.

So wurde beschlossen, den Zugang zu Schicht 6 als einen Estelle-Kanal zu einem `external`-Modul zu modellieren und später, wenn dieses Modul in Estelle ausspezifiziert wird, den Zugang zu Schicht 4 als `primitive`-Funktion zu modellieren. Damit hat man am Zugang zu Schicht 4 die größtmögliche Effizienz und am Zugang zu Schicht 6 die notwendige Flexibilität. Denn eine spätere Umstellung von `primitive`-Funktionen auf `external`-Module wäre sehr aufwendig, da die Schnittstelle völlig umgestaltet werden müßte von Funktionsaufrufen zu Nachrichtenversand. Für die Schichten tiefer als 5 ist im Rahmen dieses Projektes nicht vorgesehen, sie in Estelle zu spezifizieren.

Die Spezifikation der Schichten 1–6 als separates Estelle-Modul (ob nun `external` oder ausspezifiziert) besitzt noch einen weiteren wichtigen Vorteil. Probleminhärent besteht ein großes Potential an Nebenläufigkeit zwischen der Implementierung der tieferen Schichten einerseits und dem SAO von TP, das auf die tieferen Schichten zugreift, andererseits. Durch diese Spezifikation in getrennten Modulen wird es grundsätzlich möglich, diese Nebenläufigkeit in der Estelle-Spezifikation zu erhalten und in einer parallelen Implementation zu nutzen.

5.4.2 Die Struktur der Protokollmaschinen TPPM und CPM

Eine weitere zu lösende Frage war die nach einer geeigneten Abbildung der Struktur der Protokollmaschinen TPPM und CPM in Estelle. Jede dieser Protokollmaschinen (hier gibt es keinen Unterschied zwischen TPPM und CPM) umfaßt eine Steuerfunktion MACF und eine variable Anzahl von Einzelassoziationsobjekten SAO. Dies ließe sich sehr gut durch ein Estelle-Modul abbilden, das zwei

Arten von Sohnmodulen hat, wobei von der zweiten Art dynamisch Instanzen erzeugt und freigegeben werden können. Allerdings hat ein SAO auch die Eigenschaft, daß es nach Beendigung seiner Aufgabe für die Protokollmaschine in einen Pool von freien SAOs wandern kann, von dem aus es anschließend einer neuen Protokollmaschine (TPPM oder CPM) zugeordnet werden kann. Diese Art von Modulwanderung läßt sich mit Estelle-Sohnmodulen nicht beschreiben. In Estelle können zwar dynamisch Kommunikationsverbindungen auf- und wieder abgebaut werden, aber die Zuordnung zu einem Vatermodul ist nach der Modulerzeugung bis zur Modulfreigabe endgültig.

Daher wurde entschieden, die Protokollmaschinen TPPM und CPM mit ihrer wechselnden Zuordnung der SAOs nicht als Estelle-Module, sondern als dynamisch veränderbare Struktur von Estelle-Kanälen zu modellieren. Dies ist deshalb problemlos möglich, da eine TPPM oder CPM selbst keine aktive Komponente ist, vielmehr werden alle ihre Aktionen durch ihre Steuerfunktion MACF veranlaßt. Damit werden alle MACFs und SAOs aus allen TPPMs und der CPM eines Anwendungsvorkommnisses als nebeneinanderstehende Sohnmodule desselben Vatermoduls spezifiziert. In dieser „flachen“ Modulstuktur ergibt sich die Gruppierung zu einer TPPM oder CPM aus der Struktur der Kommunikationsverbindungen. Abbildung 5 zeigt zwei solche Gruppen und ein freies SAO. (In diesem Beispiel gibt es genau eine Gruppe von jeder Art, aber im allgemeinen Falle kann ein Anwendungsvorkommnis mehrere TPPMs enthalten und muß lediglich immer genau eine CPM enthalten.) Auch das (`external`-)Modul, das den Zugang zu den tieferen Schichten modelliert, steht auf der gleichen Modulhierarchieebene, da es zwar eine feste eins-zu-eins-Beziehung zwischen diesen Modulen und den SAOs gibt, aber keine konzeptuelle Struktur existiert, die eine Zusammenfassung eines solchen Paares darstellen würde. (Darüber hinaus ist die jetzt gewählte Lösung effizienter, da sie das „Weiterreichen“ von Nachrichten an Sohnmodule vermeidet).

5.4.3 Die Struktur der Steuerfunktion MACF

In Kapitel 5.3 wurden bereits die Besonderheiten der Steuerfunktion MACF beschrieben, und in Kapitel 2.1 wurde diskutiert, auf welche Weise diese in der INRIA- und der NIST-Version der Estelle-Spezifikation von TP umgesetzt wurden.

Eine MACF besitzt für jeden aufgebauten Dialog einen eigenen erweiterten endlichen Automaten. Und da jedes Estelle-Modul einen erweiterten endlichen Automaten darstellt, ist es zunächst sehr naheliegend, innerhalb der Estelle-Spezifikation einer MACF ein Estelle-Sohnmodul zu spezifizieren, das genau einen Dialog verwaltet. Ebendies wurde auch in der NIST-Version getan. Leider entfernt sich aber der Standard für das TP-Protokoll ein Stück weit von dem Konzept der (asynchron) kommunizierenden erweiterten endlichen Automaten, indem er spezielle Zustandsübergänge vorsieht, die die Automaten nur ausführen dürfen,

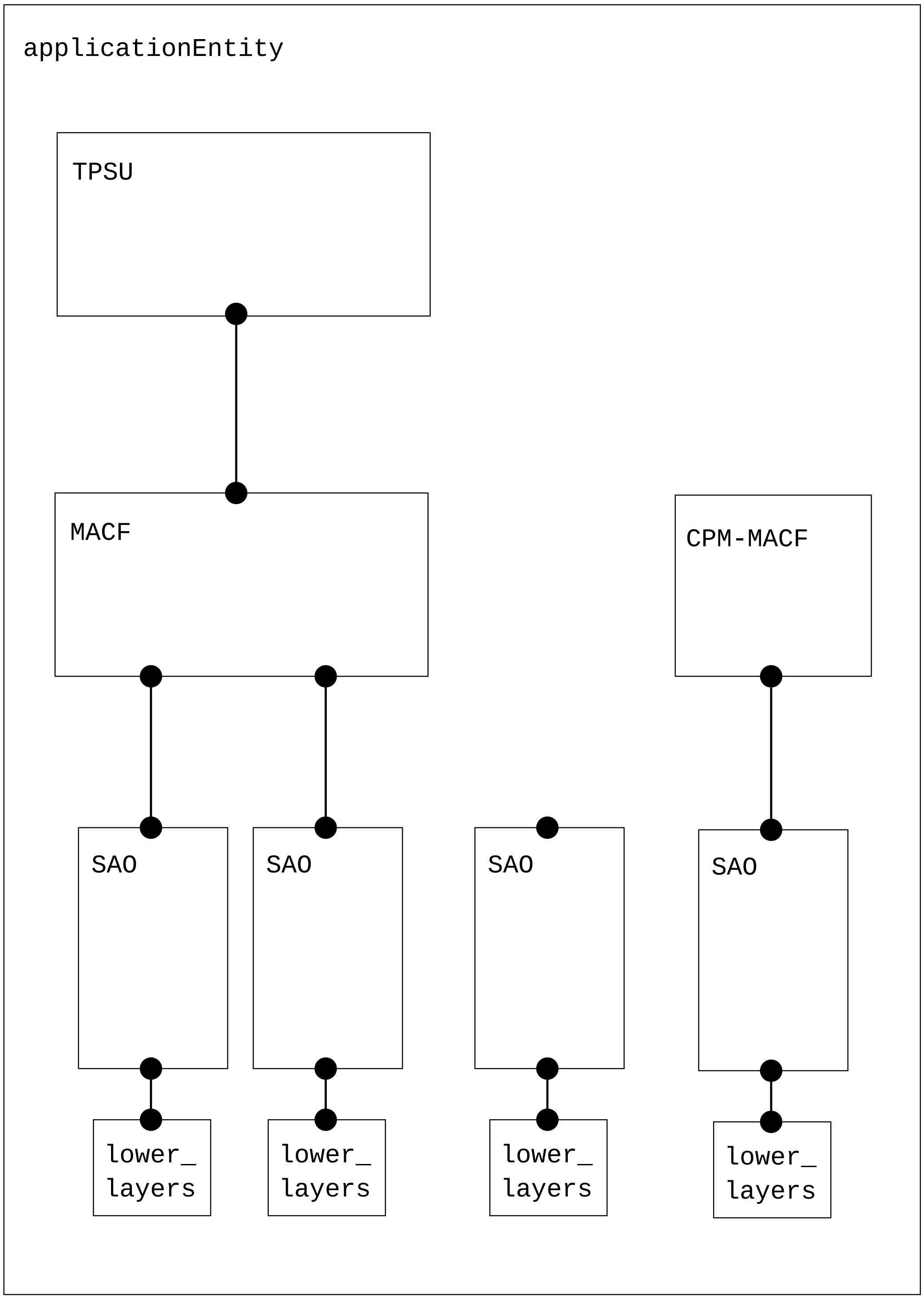


Abbildung 5: Eine „flache“ Modulstruktur aufgrund der wechselnden Zuordnung der SAOs.

wenn alle Automaten dazu bereit sind und es gemeinsam und gleichzeitig tun. Es handelt sich in diesem Falle also um synchrone Kommunikation. In der NIST-Version führte dies dazu, daß eine umständliche und ineffiziente Annäherung an diese Kommunikationsart spezifiziert wurde, bei der das MACF-Vatermodul per „busy waiting“ bestimmte exportierte Variablen seiner Sohnmodule überwacht und bei Eintreten des erwarteten Ereignisses auf gesonderten Kanälen eine Nachricht an alle Sohnmodule sendet, die diese anschließend zum eigentlichen Zustandsübergang veranlaßt. Diese Lösung ist für unsere Zwecke selbstverständlich völlig ungeeignet, da erstens das „busy waiting“ ineffizient ist und zweitens die Kommunikation über exportierte Variablen bei einer Realisierung mit asynchron parallelen Modulen nicht mehr möglich ist (vergleiche dazu Kapitel 5.5). (Weiterhin findet sich durch diese Aufteilung der Zustandsübergänge in diejenigen, die alle Dialogen gemeinsam durchführen, und diejenigen, die nur einen Dialog betreffen, die Struktur des TP-Standards nur relativ schlecht in der Estelle-Spezifikation des NIST wieder. Die Zustandsübergänge werden recht willkürlich auseinandergerissen und verteilt.)

Da also die Strukturierung in Estelle-Module in der NIST-Version für eine effiziente Parallelisierung nicht verwendbar ist, mußte eine neue und geeignete Modulstruktur gefunden werden. Hier bot sich der Ansatz an, der bei INRIA verwendet worden war. Wie in Kapitel 2.1.1 erläutert, konnte die INRIA-Version insgesamt allerdings nicht verwendet werden, weil sie (a) auf einer erheblich veralteten TP-Version basierte und (b) stark unvollständig war. Somit mußte der Teil der NIST-Spezifikation, der das Modulgerüst ausfüllt, in eine Modulstruktur eingepaßt werden, die der der INRIA-Version entspricht.

Wir spezifizierten folglich nur ein Estelle-Modul für die MACF, ohne Sohnmodule. Der Hauptzustand des Estelle-Moduls bildet den MACF-Gesamtzustand ab, der für alle Dialoge gilt. Für die Repräsentation der Dialoge besitzt das MACF-Modul eine Tabelle von erweiterten Zuständen, dargestellt als ein Feld von Verbundvariablen. Jeder dieser Verbunde enthält eine Variable, die den Hauptzustand des betreffenden Dialoges abbildet, sowie weitere Variablen, die wie gewohnt den erweiterten Zustand abbilden. Mithilfe der `any`-Klausel wurden diejenigen Estelle-Transitionen spezifiziert, die die (gleichartigen) gemeinsamen Zustandsübergänge aller Dialog-bezogenen Automaten beschreiben. Hierdurch kann die oben beschriebene synchrone Kommunikation der Automaten ganz natürlich ausgedrückt werden: Die Schaltbedingung einer solchen Transition muß lediglich den betreffenden Teil des erweiterten Zustandes (bzw. der Variablen des Verbundes) aller Automaten (bzw. aller Feldeinträge) prüfen, und die Schaltwirkung muß entsprechend alle setzen.

Als Konsequenz ergab sich damit allerdings eine umfangreiche Umstrukturierung der Spezifikation der MACF-Komponente. Aus der NIST-Version konnten lediglich die Typ- und Kanaldefinitionen unbesehen übernommen werden. Die Transitionen mußten einzelnd inspiziert, oft umformuliert und dann neu angeordnet werden. Hierbei wurde auch intensiv mit den Zustandstabellen des

TP-Standards verglichen, um Fehler in der NIST-Version aufzudecken. In Kapitel 2.1.2 hatten wir bereits erwähnt, daß diese Spezifikation offensichtlich über eine Syntaxprüfung hinaus niemals validiert worden war.

Insgesamt war damit ein recht hoher Aufwand notwendig, um die Spezifikation in eine Form zu bringen, die eine effiziente Implementation überhaupt zuläßt.

Als Vorteil der Wahl der MACF-Struktur kann man noch erwähnen, daß sie gegenüber der NIST-Version strukturell nunmehr wieder deutlich dichter am TP-Standard liegt, was sich bei Vergleichen zwischen dem Standard und der Estelle-Spezifikation positiv bemerkbar macht.

5.4.4 Die Struktur des Einzelassoziationsobjektes SAO

Ein Einzelassoziationsobjekt SAO enthält eine ganze Reihe von Komponenten, vergleiche Abbildung 4. Es bot sich an, ein Estelle-Modul zu spezifizieren, das mehrere Untermodule besitzt, zum Beispiel für jedes Anwendungsdienstelement eines. Im Gegensatz zu TPPM und CPM standen dem auch keine prinzipiellen Hindernisse entgegen.

Bei einer näheren Analyse des TP-Protokolls stellte sich allerdings heraus, daß innerhalb eines SAOs sehr wenig Nebenläufigkeit vorkommt. (Mit Ausnahme der Konkatenations- und Separations-Komponente, einem in Abbildung 4 nicht dargestellten Teil der Steuerfunktion SACF, die sogar fast völlig nebenläufig zur gesamten TP-Protokollmaschine läuft.) Darüberhinaus führt jede SAO-Komponente jeweils meist nur sehr einfache und kurze Aktionen aus. Daher stand es zu erwarten, daß eine Aufteilung der Funktionalität des SAO auf derart viele Estelle-Sohnmodule zu einem großen Aufwand an Kommunikation führen würde im Vergleich zu den eigentlichen Aktionen, und dies, ohne Geschwindigkeitsvorteile durch eine Parallelisierung gewinnen zu können. Dies war nicht wünschenswert im Hinblick auf die angestrebte Effizienzsteigerung der parallelisierten Spezifikation.

Und noch eine Schwierigkeit zeigte sich. Die Dienste einiger der Komponenten innerhalb eines SAO werden von nicht nur einer anderen Komponente, sondern oft von fast allen anderen Komponenten benutzt. So wird etwa der ACSE-Dienst von der Steuerfunktion SACF, von der TP-ASE und (potentiell) von der U-ASE benutzt. Damit kann der ACSE-Dienst den anderen Komponenten nicht mehr über einen einzigen Estelle-Kanal zur Verfügung gestellt werden, es werden drei Kanäle notwendig, und im ACSE-Modul müssen die betreffenden Empfangstransitionen verdreifacht werden.

Die oben erwähnte Konkatenations- und Separations-Komponente wurde aufgrund ihrer offensichtlichen und starken Nebenläufigkeit zum restlichen Teil des SAOs als asynchrones Sohnmodul des SAOs spezifiziert.

Ansonsten wurde aber beschlossen — insbesondere wegen des Argumentes der Ineffizienz von vielen kleinen, sequentiell arbeitenden Sohnmodulen — die Anwendungsdienstelemente innerhalb des SAO sowie die Steuerfunktion SACF selbst nicht als separate Estelle-Sohnmodule zu spezifizieren, sondern in einem für Estelle neuen Spezifikationsstil, einem *prozeduralen Stil*. Dies bedeutet, daß eine solche Komponente als ein zusammengehöriger Satz von Prozeduren spezifiziert wird, der von den anderen Komponenten aufgerufen werden kann.

Mußte also zum Beispiel bisher die Steuerfunktion SACF eine Nachricht an das TP-Anwendungsdienstelement TP-ASE senden, damit dieses die Nachricht umkodiert und das Ergebnis an die Darstellungsschicht weitersendet, so kann nun die SACF einfach eine TP-ASE-Prozedur aufrufen, die sowohl die Umkodierung als auch das Senden an die Darstellungsschicht übernimmt.

Die Verwendung eines solchen Stils wird dadurch erleichtert, daß die meisten Anwendungsdienstelemente des SAO im wesentlichen Aufgaben ausführen, bei denen immer genau eine Nachricht versandt wird, wenn eine Nachricht empfangen wird, also Umkodierungsaufgaben. Für TP-ASE trifft dies exakt zu, für die anderen ASEs nur teilweise, aber in ausreichendem Maße. (Es gibt bei diesen ASEs auch einen Zustand, und gelegentlich folgt auf den Empfang einer Nachricht keine Ausgabe einer Nachricht.)

Letzlich zeigt das oben angeführte Argument der intensiven gegenseitigen Benutzung der Dienste innerhalb eines SAO, daß das Konzept des Prozeduraufrufes die im TP-Standard beschriebene SAO-Struktur sogar besser wiedergibt als das Konzept von über Estelle-Kanäle kommunizierenden Sohnmodulen.

Erschwert wird die Anwendung des prozeduralen Stils, wenn bereits existierende Estelle-Spezifikationen von ASEs in die TP-Spezifikation integriert werden sollen. Hier ist dann ein Umschreiben der ASE-Spezifikation erforderlich. Dies betraf allerdings konkret nur das Assoziationssteuerungsdienstelement ACSE, da zu den anderen ASEs ohnehin noch keine Estelle-Spezifikationen vorhanden waren. Die ACSE-Spezifikation wurde außerhalb des Projektes in einer Studienarbeit in den prozeduralen Stil umgesetzt ([Lap94a]).

Auch die Transitionen der Steuerfunktion SACF, die in der NIST-Version vorlagen, mußten in Prozeduren umgesetzt werden. Diese Umsetzung konnte allerdings recht geradlinig erfolgen und erforderte hauptsächlich Editieraufwand, da jeweils meist genau eine Transition auf eine Prozedur abgebildet wurde. (Pro empfangener Interaktion wurde eine Prozedur vorgesehen. Weitere Fallunterscheidungen, die bisher gelegentlich mit `provided`-Klauseln getroffen wurden, werden nun innerhalb der Prozedur abgehandelt.)

Das größte Hindernis haben wir allerdings bis jetzt verschwiegen. Die Sprachdefinition von Estelle enthält die Einschränkung, daß in Prozeduren keinerlei Estelle-spezifische Anweisungen vorkommen dürfen. Damit wird es zum Beispiel einer TP-ASE-Prozedur unmöglich gemacht, nach der Umkodierung auch die Ausgabe (mittels der Estelle-spezifischen `output`-Anweisung) durchzuführen. Der Grund für diese Einschränkung in der Sprachdefinition liegt wohl darin, daß es ein ausdrückliches Entwurfsziel bei der Entwicklung von Estelle war, den Pascal-artigen Sprachanteil, der sich nicht mit endlichen Automaten und deren Kommunikation befaßt, sondern z.B. der Beschreibung von Daten dient, im Prinzip austauschbar zu halten. ([Dem90, S. 447]).

Die Einschränkung verhindert aber nicht nur den beschriebenen neuen Spezifikationsstil, sondern macht auch ganz allgemein große Estelle-Spezifikationen wie etwa die von TP deutlich schwerer lesbar. Denn bereits in den beiden existierenden TP-Spezifikationen fällt auf, daß sich in den vielen Transitionen bestimmte Anweisungsfolgen immer und immer wieder wiederholen. Dies hat unter anderem seine Ursache darin, daß sie die Anwendung von Prozeduren widerspiegeln, die im TP-Standard definiert sind. In Estelle dagegen konnten diese Anweisungsfolgen nicht als Prozedur spezifiziert werden. Estelle basiert zwar auf Pascal und kennt damit die üblichen Stukturierungsmöglichkeiten durch Prozeduren und Funktionen, aber durch die erwähnte bisher vorgeschriebene starke Trennung zwischen dem Pascal-Teil und dem Estelle-spezifischen Teil ist es nicht erlaubt, zum Beispiel die Estelle-spezifische `output`-Anweisung in einer Prozedur zu verwenden. Und da die sich wiederholenden Anweisungsfolgen insbesondere `output`-Anweisungen enthalten, können Prozeduren hier nicht zur Strukturierung der Spezifikation verwendet werden.

In [Bre93] haben wir die Definition der Semantik von Estelle näher untersucht und festgestellt, daß man die beschriebene Einschränkung ohne Schwierigkeiten aufheben kann, und wir haben beschrieben, wie dies in der Definition der Semantik von Estelle geschehen kann. Die notwendigen Änderungen der Syntax und Semantik in [ISO89b] sind im Anhang von [Bre93] im Detail beschrieben und stellen gleichzeitig Hinweise dar, wie vorhandene Estelle-Werkzeuge leicht angepaßt werden können.

Somit haben wir beschlossen, neben dem in Kapitel 5.5.1 diskutierten zusätzlichen Modulattribut eine weitere kleine Änderung an der Sprachdefinition von Estelle vorzunehmen, um dem Ziel einer effizient implementierbaren Spezifikation näher zu kommen.

5.4.5 Modul-Management

Das OSI-Basisreferenzmodell ([ISO81]) schreibt für jedes Anwendungsvorkommnis die Existenz eines Schichtmanagers vor, der bei Empfang eines Verbindungsaufbauwunsches die notwendigen Vorkommnisse instanziiert, also in unserem Falle die Paare aus TP-Dienstnutzer TPSUI und TP-Protokollmaschine TPPM. Da wir bereits ein Estelle-Modul (`applicationEntity`) zur Abbildung eines Anwendungsvorkommnisses vorgesehen haben, und da dieses Estelle-Modul sonst keine weitere Aufgabe besitzt, haben wir die Aufgabe des Managers diesem Modul übertragen. Auf diese Weise haben wir gleich noch ein weiteres Problem gelöst. In Estelle kann sich kein Modul selbst beenden oder die Kommunikationsstruktur seiner äußeren Interaktionspunkte verändern, weshalb ohnehin sein Vatermodul diese Managementaufgaben übernehmen muß. Da wir für die diversen Komponenten von TP im wesentlichen eine „flache“ Modulstruktur vorgesehen haben (vergleiche Kapitel 5.4.2), ist dieses Vatermodul ohnehin das Modul `applicationEntity`. Somit übernimmt dieses Modul sämtliche Modul- und Kommunikationsmanagementaufgaben für TP.

5.4.6 Die Gesamt-Modulstruktur

Abbildung 6 gibt einen Gesamt-Überblick über die Modulstruktur der Estelle-Spezifikation und faßt damit einige Aspekte der vorigen Kapitel zusammen.

Die gestrichelten Kästen bezeichnen Module, die erst dann in die Spezifikation integriert werden, wenn der Zugang zu den tieferen Schichten nicht mehr als `external`-Modul (`lowerLayers`) spezifiziert wird, sondern ausspezifiziert wird (vergleiche Kapitel 5.4.1). Hier sind auch die Namenskonventionen etwas uneinheitlich, da diese Teile direkt aus anderen Estelle-Spezifikationen übernommen wurden (vergleiche auch Kapitel 5.6).

5.4.7 Allgemeiner Spezifikationsstil

Anzahl der Transitionen Ziel der Erstellung der Estelle-Spezifikation ist es, eine effizient ausführbare Spezifikation zu erhalten. Neben dem Aspekt der Parallelisierung haben wir uns auch sonst bemüht, einen Spezifikationsstil zu wählen, der eine effiziente Ausführung erleichtert. Wo es möglich war, haben wir versucht, Overhead für verschiedene Arten von Verwaltung zu vermeiden. So benötigt zum Beispiel das Estelle-Laufzeitsystem einen deutlichen Anteil an der gesamten Ausführungszeit (vergleiche etwa [Hof93, Kap. 4]).

Um den Aufwand für die Auswahl einer schaltbereiten Transition zu verringern, haben wir uns bemüht, die Anzahl der Transitionen möglichst gering zu halten (und somit eher „große“ Transitionen zu spezifizieren). Natürlich ist weiterhin für jeden Eintrag in der Tabelle des Protokoll-Standards eine Estelle-Transition notwendig, aber zum Beispiel können die Untergliederungen innerhalb dieser Kästchen ebensogut auch durch Fallunterscheidungen (`if/then/else`) innerhalb einer einzelnen Transition in Estelle abgebildet werden.

Namenskonventionen Aus der TP-Spezifikation des NIST wurden die Typ- und Kanaldefinitionen übernommen, und damit wurden im wesentlichen auch die Namenskonventionen dieser Spezifikation übernommen. In Anhang B sind diese Namenskonventionen beschrieben, und ebenso auch die kleinen Modifikationen, die für die parallelisierte TP-Version an den Regeln vorgenommen wurden, um die Spezifikation noch übersichtlicher und besser lesbar zu machen.

Strukturierung des Spezifikationstextes Aufgrund des extrem großen Umfanges der TP-Spezifikation war es aus Gründen der Handhabbarkeit auf jeden Fall notwendig, den Spezifikationstext auf mehrere Dateien zu verteilen. Der Estelle-Standard ([ISO89b]) sieht dafür zwar nichts vor, aber praktisch alle

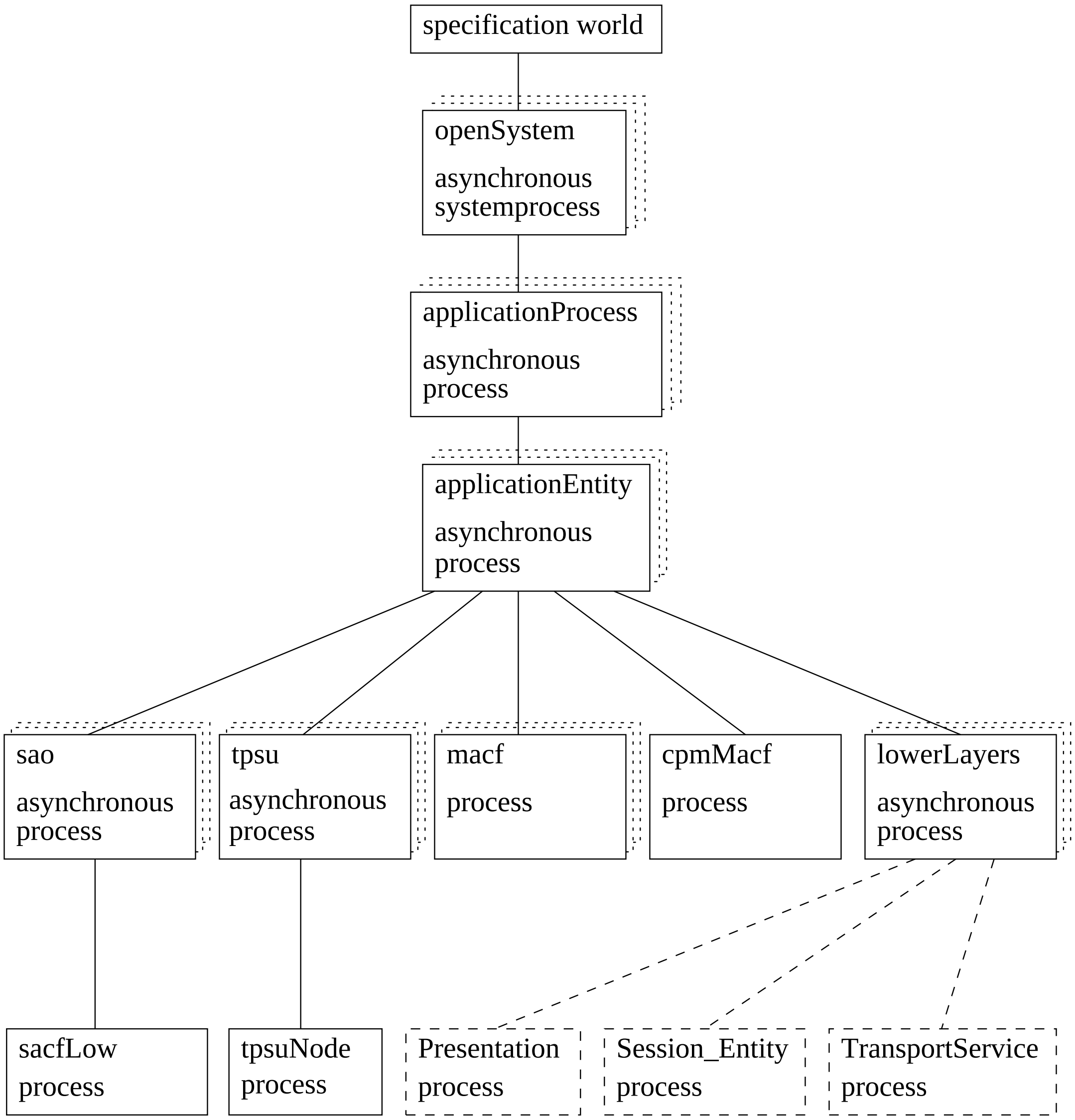


Abbildung 6: Die Gesamt-Modulstruktur der Estelle-Spezifikation von TP

Estelle-Werkzeuge unterstützen eine solche Aufteilung durch ein `#include`-Konstrukt, das den Inhalt einer Datei anstelle dieser Zeile in den Text einfügt. Bild 7 zeigt die gewählte Dateistruktur.

Die Datei `tpworld.e` enthält außer dem Spezifikationsrahmen `specification world;` auch alle Moduldefinitionen für die Einbettung von TP in die OSI-Anwendungsschichtstruktur, da diese nur sehr kurz ausfallen. Die Datei `applentity.e` enthält dann die Definition des Anwendungsvorkommnisses `applicationEntity` und besitzt `#include`-Konstrukte für fast alle weiteren Dateien, die ganze Moduldefinitionen enthalten, was die flache Modulstruktur (vergleiche Kapitel 5.4.2) wiederspiegelt. Die prozeduralen Definitionen der SAO-Komponenten (vergleiche Kapitel 5.4.4) wurden in gleicher Weise jeweils in eine eigene Datei strukturiert, die dann der Datei `sao.e` untergeordnet ist. Aufgrund ihres Umfangs wurden die Funktionsdefinitionen des MACF-Moduls in die Datei `macf.f.e` ausgelagert, es handelt sich hierbei jedoch nicht um eine separate Komponente des MACF-Moduls. Die gestrichelten Kästen bezeichnen Module, die erst dann in die Spezifikation integriert werden, wenn der Zugang zu den tieferen Schichten nicht mehr als `external`-Modul (`lowerLayers`) spezifiziert wird, sondern ausspezifiziert wird (vergleiche Kapitel 5.4.1). Die Typdefinitionen in den Dateien `lowerlayers1.t.e` und `lowerlayers2.t.e` sind allerdings von Anfang an notwendig, da sie die Definitionen für den Zugang zu den tieferen Schichten enthalten.

Zur Unterscheidung der verschiedenen Arten von Estelle-Quelltextteilen wurden einige einfache Konventionen für die Dateiendungen festgelegt, Tabelle 5 gibt sie wieder.

Datei-Endung	Bedeutung
`*.e`	Estelle-Spezifikation
`*.t.e`	Typ- und Konstanten-Definitionen
`*.ch.e`	Kanal-Definitionen
`*.f.e`	Funktions- und Prozedur-Definitionen

Tabelle 5: Konventionen für Dateiendungen

5.5 Nebenläufigkeit

5.5.1 Ausdruckskraft von Estelle

Der Software-Entwicklungsprozeß wird üblicherweise als eine Folge von Schritten betrachtet, die vom Problem zur Software-Lösung führen. Auf dem Gebiet der Kommunikationsprotokolle und verteilten Systeme ist es wichtig, Nebenläufigkeit (bzw. Parallelität) während jeder dieser Schritte zu betrachten. Hier tritt Nebenläufigkeit als ganz natürliches Phänomen auf. In unseren weiteren Ausführungen unterscheiden wir die folgenden Arten von Nebenläufigkeit:

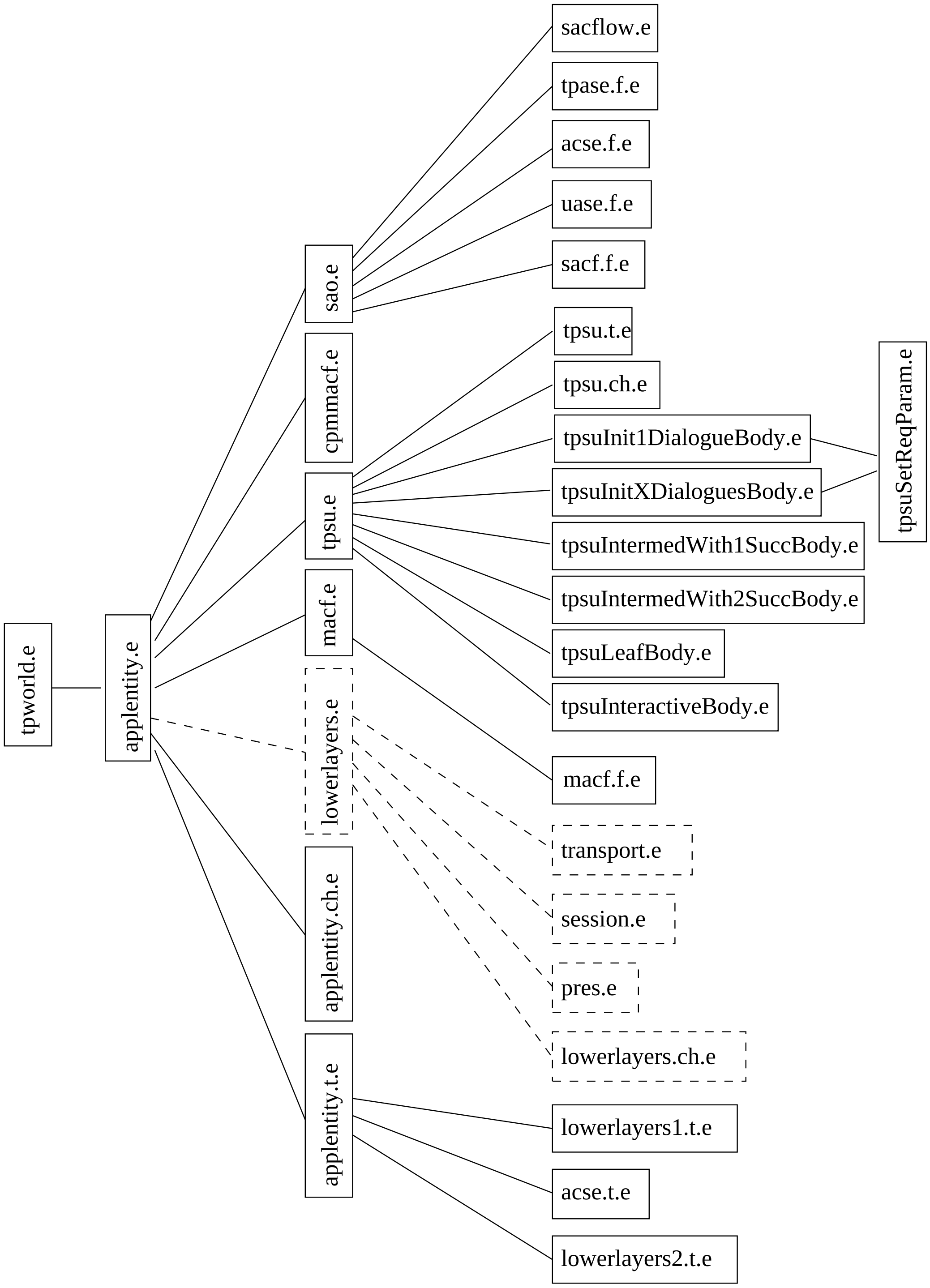


Abbildung 7: Strukturierung des Spezifikationstextes in Dateien

- Mit *expliziter Nebenläufigkeit* meinen wir offensichtliche Parallelität. Solche Parallelität gibt es zum Beispiel in Systemen mit interagierenden Komponenten, die parallel ausgeführt werden können. Explizite Nebenläufigkeit findet man typischerweise in konstruktiven Beschreibungen.

- Mit *impliziter Nebenläufigkeit* meinen wir das Parallelitätspotential, das nicht sofort offensichtlich ist, sondern erst aufgedeckt werden muß. Dies ist typischerweise der Fall in nicht-konstruktiven Beschreibungen. Aber man kann implizite Nebenläufigkeit ebenso auch in konstruktiven Beschreibungen finden, und sie kann zusammen mit expliziter Nebenläufigkeit auftreten.

Man kann auch noch weitere Formen von Nebenläufigkeit entlang den Stufen des Software-Entwicklungsprozesses unterscheiden:

- *Probleminhärente Nebenläufigkeit* bezeichnet die potentielle Parallelität innerhalb des Problems.

- *Spezifizierte Nebenläufigkeit* bezeichnet die potentielle Parallelität, die sich aus der Entwurfsphase ergibt.

- *Implementierte Nebenläufigkeit* bezeichnet die tatsächliche Parallelität, die im Endprodukt realisiert wurde.

In jeder dieser Stufen können wir zwischen expliziter und impliziter Nebenläufigkeit unterscheiden. Beispielsweise gibt es probleminhärente explizite Nebenläufigkeit in verteilten Anwendungen. Ein Kommunikationsdienst ist ein Beispiel für probleminhärente implizite Nebenläufigkeit. Spezifizierte explizite Nebenläufigkeit findet man in der Beschreibung einer Menge von Protokollinstanzen. Spezifizierte implizite Nebenläufigkeit ergibt sich etwa aus Eingabe/Ausgabe-Zusicherungen, die parallele Implementierungen zulassen. Implementierte explizite Nebenläufigkeit ergibt sich aus der parallelen Implementation von Protokollinstanzen. Ein Beispiel für implementierte implizite Nebenläufigkeit ist ein sequentielles Programm, für das paralleler Code für eine Spezial-Hardware (Pipelining) erzeugt wird. Folglich sind die obigen Klassifikationen der Nebenläufigkeit orthogonal.

Die probleminhärente Nebenläufigkeit beschreibt die maximale Parallelität innerhalb eines Problems. Diese Nebenläufigkeit kann ausgenutzt werden, um eine effiziente Lösung zu erhalten, und/oder sie kann in späteren Entwicklungsschritten reduziert werden. Beispielsweise kann sie reduziert werden, indem mehrere Systemkomponenten auf einem Einzelprozessor ausgeführt werden. Sobald die Nebenläufigkeit in einer Stufe reduziert worden ist, wird die resultierende spezifizierte Nebenläufigkeit zur oberen Grenze für die Parallelität in den folgenden Stufen, einschließlich der Implementation. Somit bestimmen Entwurfsentscheidungen, wie die für eine bestimmte Systemarchitektur oder sogar die Wahl einer bestimmten Spezifikationssprache, entscheidend das Potential für eine effiziente Implementierung. Wie bereits erwähnt, unterscheiden wir zwischen expliziter und impliziter Nebenläufigkeit in jedem Entwurfsschritt. Indem man implizite Nebenläufigkeit ausnutzt, kann man die explizite Nebenläufigkeit der folgenden Schritte erhöhen. Dies bedeutet, daß implizite Nebenläufigkeit explizit gemacht wird, die Gesamt-Nebenläufigkeit wird allerdings nicht erhöht.

An diesem Punkt kommen wir nun zur Formalen Beschreibungstechik Estelle. Damit die Estelle-Spezifikation von TP parallelisiert werden kann, ist es unerläßlich, daß Estelle überhaupt hinreichende Ausdrucksmittel für spezifizierte explizite Nebenläufigkeit zur Verfügung stellt. Leider stellte es sich aber schnell heraus, daß die Ausdrucksmittel bisher *nicht* ausreichen. Im folgenden werden wir beschreiben, wo die Schwierigkeiten liegen.

Eine Estelle-Spezifikation besteht aus einer Anzahl von Subsystemen. Jedes Subsystem ist, wie das Gesamtsystem, hierarchisch strukturiert, es besteht aus einer Modulinstanz, die das Attribut `systemprocess` oder `systemactivity` trägt. Im folgenden werden wir solche Modulinstanzen auch als *Systemmodulinstanzen* bezeichnen. Die Attributierungsregeln legen fest, daß die Nachfahren eines Systemmoduls nicht mehr selbst Systemmodul sein können. Gemäß der Definition von „Subsystem“ kann ein Subsystem keine weiteren Subsysteme enthalten. Dies hat erhebliche Auswirkungen auf die explizite Nebenläufigkeit, die in Estelle ausdrückbar ist, wie wir sehen werden.

Für die folgenden Überlegungen sind zwei Eigenschaften von Subsystemen relevant. Erstens definiert das Estelle-Ausführungsmodell keinerlei Synchronisationsbedingungen oder Prioritätsregelungen zwischen den Modulinstanzen verschiedener Subsysteme. Dies heißt, daß Subsysteme mit verschiedenen Geschwindigkeiten voranschreiten können, solange nicht die Spezifikation selbst für Synchronisation durch einen Nachrichtenaustausch zwischen verschiedenen Subsystemen sorgt. Die zweite Eigenschaft folgt indirekt aus den Attributierungsregeln. Da die Vorfahren einer Systemmodulinstanz passiv sind (d.h. sie besitzen nur eine Initialisierungstransition), kann die Menge der Subsysteme und der Verbindungen zwischen ihnen nach der Initialisierungsphase nicht mehr verändert werden. Dies bedeutet, daß die Grobstruktur eines Estelle-Systems statisch ist.

Innerhalb eines Subsystems gelten festgelegte Prioritätsregelungen sowie weitere Synchronisationsbedingungen, die über Modulattribute spezifiziert werden können. Diese Maßnahmen stellen sicher, daß das Schalten von Transitionen innerhalb eines Subsystems in „Runden“ (oder „Berechnungsschritten“, [DeBu89]) erfolgt. In jeder Runde werden ausführbare Transitionen gemäß bestimmter Regeln ausgewählt und geschaltet. Eine solche Regel besagt, daß jede Modulinstanz vor allen ihren Nachfahren Vorrang hat, unabhängig von ihrem Attribut. Das heißt, daß, wenn in einer gegebenen Runde eine Modulinstanz eine ausführbare Transition besitzt, alle ihre Nachfahren in dieser Runde keine einzige Transition schalten dürfen. Ob diese Modulinstanz selbst ihre Transition schalten darf, hängt von weiteren Bedingungen ab.

Aus den Prioritätsregeln und den Synchronisationsbedingungen folgt, daß jede Modulinstanz eines Subsystems höchstens eine Transition pro Runde schalten lassen kann. Um weitere Transitionen schalten zu lassen, muß sie bis zur nächsten Runde warten. Das heißt, daß, obwohl die Sohnmodulinstanzen einer Prozeß-Modulinstanz ihre Transitionen nebenläufig schalten lassen können, sie nicht mit unterschiedlichen Geschwindigkeiten laufen können, wie dies zum Beispiel zwischen Subsystemen der Fall sein kann. Dies schränkt die Verwendbarkeit eines Subsystems, das mehrere Modulinstanzen enthält, für die Beschreibung von expliziter Nebenläufigkeit und damit für parallele Implementationen erheblich ein.

Ein wichtiger Grund für diese sehr restriktiven Regeln innerhalb eines Subsystems besteht darin, daß eine Modulinstanz exportierte Variablen besitzen kann, auf die sie selbst und ihre Vatermodulinstanz (d.h. ihr direkter Vorfahre) zugreifen können. Die Prioritätsregel stellt sicher, daß niemals Transitionen dieser beiden Modulinstanzen nebenläufig schalten, weshalb die Vatermodulinstanz den Wert einer exportierten Variablen einer Sohnmodulinstanz prüfen kann (mithilfe einer `provided`-Klausel), um eine ausführbare Transition zu bestimmen, und sich darauf verlassen darf, daß die Sohnmodulinstanz den Wert nicht verändert, bevor die Transition geschaltet hat.

Kommen wir nun zu den Grenzen der Ausdruckskraft von Estelle für Nebenläufigkeit. Dafür vereinfachen wir die komplexe Struktur der Architektur von TP aus Abbildung 4 auf Seite 15 zu der abstrakten Protokollarchitektur in Abbildung 8. Sie zeigt eine beliebige, aber feste Anzahl von Protokollkomponenten vom Typ A. Jede Protokollkomponente vom Typ A enthält eine variable Anzahl von Protokollkomponenten der Typen B und C. Wir nehmen an, daß sich die Zahl dieser Komponenten dynamisch ändern kann. Weiterhin enthält jede Protokollkomponente vom Typ C genau eine Protokollkomponente vom Typ D und eine veränderliche Anzahl von Protokollkomponenten vom Typ E.[12]

Nun stellen wir die Frage, ob eine solche Protokollarchitektur — unter Beachtung der genannten Randbedingungen — in Estelle durch eine geeignete Modulattributierung abgebildet werden kann. Eine Möglichkeit bestünde darin, die Protokollkomponenten vom Typ A als inaktive, nichtattributierte Estelle-Modulinstanzen zu spezifizieren. In jedem A-Modul könnten (aktive) B- und C-Module deklariert werden und dann die entsprechende Anzahl von Modulinstanzen erzeugt werden. Um die maximale explizite Nebenläufigkeit zu erhalten, die in Estelle ausdrückbar ist, könnten diese Module mit dem Attribut `systemprocess` versehen werden (siehe Abbildung 9). Der Nachteil dieser Lösung besteht darin, daß die resultierende Estelle-Architektur statisch wäre, während die Protokollarchitektur die dynamische Erzeugung und Beendigung von B- und C-Modulinstanzen verlangt.

[12] Eine Abbildung in die TP-Architektur erhält man, wenn man A = „Offenes System“ setzt, B = TPSU, C = TPPM, D = MACF und E = SAO.

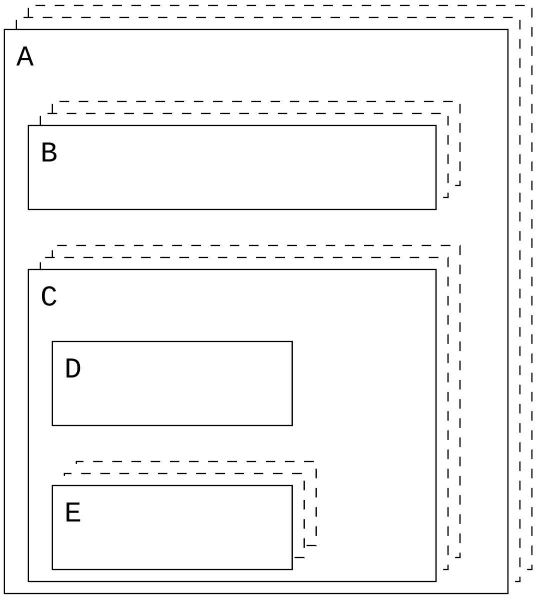


Abbildung 8: Eine abstrakte Protokollarchitektur

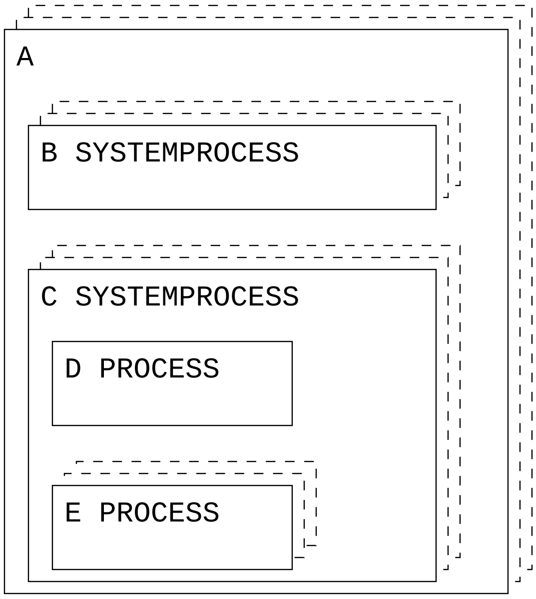


Abbildung 9: Modul A als inaktiv spezifiziert

Eine Alternative wäre, das A-Modul mit dem Attribut `systemprocess` zu attributieren (siehe Abbildung 10). Dann könnte das A-Modul die notwendigen Veränderungen der Estelle-Architektur durchführen. Aber dies hätte zur Folge, daß alle B- und C-Modulinstanzen einer A-Modulinstanz Teil desselben Subsystems wären. Sie müßten sich nach *jedem* Schalten einer Transition synchronisieren, selbst wenn dies im Hinblick auf die Anforderungen des Protokolls gar nicht notwendig wäre. Damit hat diese Lösung im allgemeinen (und speziell im Falle von TP) die nachteilige Konsequenz, daß sie die probleminhärente explizite Nebenläufigkeit in der Spezifikation drastisch reduziert.

Ähnliche Überlegungen gelten für die Abbildung der Architektur der C-Komponente in Estelle. Beide diskutierten Alternativen würden dazu führen, daß die D- und E-Modulinstanzen nicht als Systemmodulinstanzen attributierbar wären, womit eine asynchrone Ausführung unmöglich wäre. Selbst wenn die erste Alternative gewählt würde, wäre an dieser Stelle eine beträchtliche Reduktion der probleminhärenten expliziten Nebenläufigkeit unvermeidlich, und mit ihr ein unwiderruflicher Verlust an Effizienz in der Implementation.

Die Schwierigkeiten, die probleminhärente explizite Nebenläufigkeit in der Spezifikationsphase zu erhalten, sind genereller Natur und beschränken die Brauchbarkeit von Estelle nicht nur für TP, sondern allgemein als eine Spezifikationssprache für *verteilte* Systeme. Nebenläufigkeit, die in der Spezifikationsphase verloren gegangen ist, kann nicht einfach in der Implementationsphase wieder eingeführt werden, da dies die Semantik von Estelle verletzen würde. Weiterhin ist aus dem Spezifikationstext nicht ohne weiteres ersichtlich, welche Einschränkun-

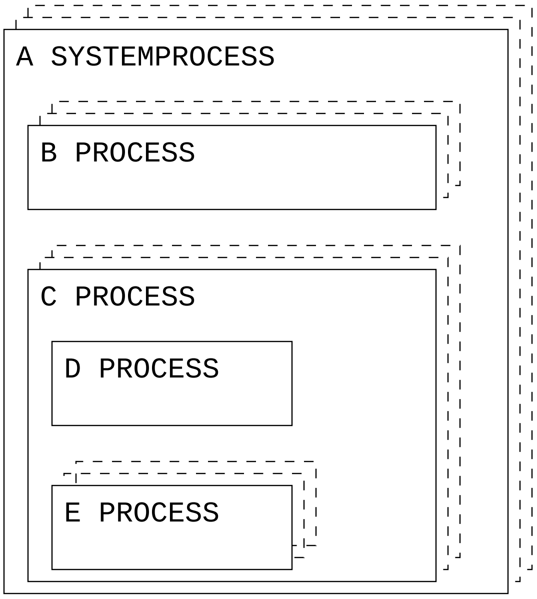


Abbildung 10: Modul A als Systemmodul spezifiziert

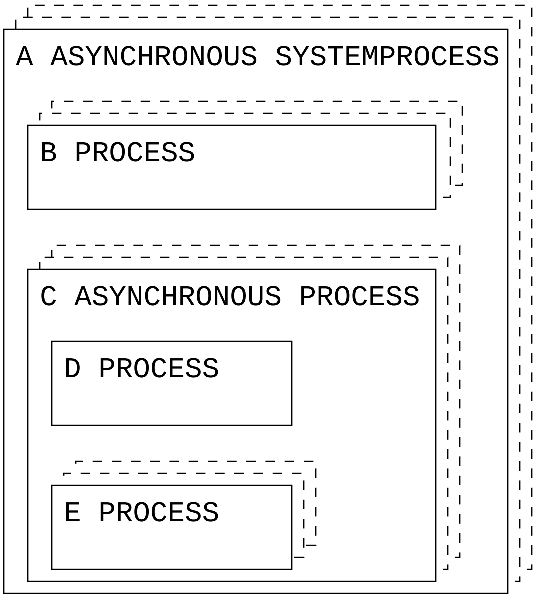


Abbildung 11: Lösung mit erweitertem Estelle

gen der Nebenläufigkeit von der Anwendung nicht gebraucht werden und fallen gelassen werden könnten. Folglich liefe man Gefahr, die Semantik der Anwendung zu verletzen, womit man die Vorteile des formalen Ansatzes verlieren würde. Somit sollte sich die Suche nach einer Abhilfe auf die Spezifikationssprache selbst konzentrieren.

Um die Nebenläufigkeit in Estelle zu erhöhen, haben wir in [BrGo93b] vorgeschlagen,

> die Modulattributierung eines beliebigen Moduls um die Möglichkeit zu erweitern, asynchron parallele Sohnmodulinstanzen zu spezifizieren,

selbst wenn diese Vatermodulinstanz aktiv ist und/oder dynamisch erzeugt wurde. Diese Sohnmodulinstanzen sind dann nicht mehr Teil der Transitionsrunden ihres Vaters. Semantisch gesehen führt die vorgeschlagene Erweiterung zu einer *vergrößerten Menge von möglichen Ausführungsfolgen*, weil asynchron parallele Sohnmodulinstanzen mit unterschiedlichen Geschwindigkeiten laufen können. Um auf unser obiges Beispiel zurückzukommen, findet sich die Lösung für das diskutierte Attributierungsproblem, die auf diese Weise möglich wird, in Abbildung 11.

In [BrGo93b] und [BrGo93a] haben wir die semantischen Konsequenzen der Erweiterung im Detail untersucht und die Einzelheiten der Anpassung der Definition von Estelle dargestellt. Dabei haben wir großen Wert darauf gelegt, die Erweiterung kompatiblel zur Estelle-Definition in [ISO89b] zu halten, die Bedeutung aller bisher erstellten Spezifikationen bleibt unverändert.

Im Detail sieht die Erweiterung der Syntax wie folgt aus. Die Definition von Estelle erlaubt fünf mögliche Attributierungen von (Vater-)Modulen:

- `process`
- `systemprocess`
- `activity`
- `systemactivity`
- — ohne Attribut —

Wie fügen ein Schlüsselwort und damit zwei weitere Möglichkeiten hinzu:

- `asynchronous process`
- `asynchronous systemprocess`

Um Probleme sowohl in der Semantik als auch bei der effizienten Implementierbarkeit derselben (siehe [BrGo93b, BrGo93a]) zu vermeiden, legen wir fest, daß die Sohnmodule von derart attributierten Modulen keine exportierten Variablen besitzen dürfen. Dies betrifft natürlich *nicht* die Sohnmodule von Modulen mit den bisher schon existierenden Attributen.

Die Semantik der Sohnmodulinstanzen von als asynchron deklarierten Modulinstanzen soll sein, daß sie, unabhängig von den Transitionsrunden ihres Vaters, vollständig asynchron arbeiten, genau wie Systemmodulinstanzen. Zu Details und der formalen Definition siehe [BrGo93b, BrGo93a].

Als Fazit dieses Kapitels halten wir fest, daß diese kleine, kompatible, aber entscheidende Erweiterung von Estelle im Hinblick auf unser Ziel, mit einer Parallelisierung der Estelle-Spezifikation von TP die Geschwindigkeit einer Implementation zu steigern, unbedingt notwendig ist. Anderenfalls hätten sich die Komponenten von TP ständig unnütz synchronisieren müssen, und die Effizienz jeder Implementation der Spezifikation wäre unvermeidlich und stark reduziert worden.

5.5.2 Atomare TP-Aktionen

Die Definition des TP-Protokolls in [ISO92] enthält das Konzept der atomaren TP-Aktionen. Wie bereits beschrieben, gibt es in TP verschiedene Protokollmaschinen, die zum Teil selbst wieder aus mehreren Protokollmaschinen zusammengesetzt sind. Ein Zustandsübergang einer dieser Teil-Protokollmaschinen (z.B. SACF als Teil eines SAO) kann bewirken, daß eine weitere Teil-Protokollmaschine einen Zustandsübergang ausführen muß (z.B. ACSE). Dies wird dadurch bewirkt, daß die erste Maschine der zweiten eine Nachricht sendet. Nun wäre es denkbar, daß bereits weitere Nachrichten von außerhalb der Gesamt-Protokollmaschine eintreffen, bevor die zweite Teil-Protokollmaschine geschaltet hat, und daß diese äußere Nachricht von einer anderen Komponente umgehend bearbeitet wird. Dies wäre eine Art von Parallelverarbeitung, die aber bisher *nicht* zulässig ist.

Um es etwas genauer zu formulieren: Die TP-Protokollmaschine TPPM, die Kanalprotokollmaschine CPM und alle freien Einzelassoziationsobjekte SAO (die also gerade keiner TPPM oder CPM zugeordnet sind) führen Aktionssequenzen *atomar* und *nebenläufig* aus. Eine *Aktionssequenz* beginnt, wenn eine dieser Maschinen eine Nachricht an einer ihrer äußeren Schnittstellen annimmt, und sie endet, wenn diese Nachricht verarbeitet ist und ebenso auch alle im Zuge der Verarbeitung innerhalb dieser Maschine versandten Nachrichten verarbeitet sind. (Die nach außerhalb versandten Nachrichten spielen hier keine Rolle.) Eine solche Aktionssequenz kann etliche Einzelaktionen umfassen. Die Atomarität einer Aktionssequenz bedeutet, daß eine TPPM, eine CPM oder ein freies SAO erst dann wieder eine Nachricht an einer ihrer (mehreren) äußeren Schnittstellen annehmen darf, wenn die vorige in ihrem Inneren vollständig verarbeitet ist.

Damit wird zwischen allen Komponenten einer dieser Maschinen bei jeder Annahme einer Nachricht eine *vollständige Synchronisation* notwendig, um sicherzustellen, daß nicht zwei Komponenten gleichzeitig eine Nachricht von außen annehmen und bearbeiten.

Selbstverständlich ist diese Regel hinreichend, um auszuschließen, daß irgendwelche unerwünschten Race-Situationen auftreten können, bei denen die parallele Bearbeitung zweier Nachrichten in einer Maschine zu Konflikten führt. Allerdings ist sie auch eine so starke Einschränkung, daß sie die interne Parallelisierung von TP-Knoten unmöglich macht.

Es ist anzunehmen, daß die Atomaritätsforderung für Aktionssequenzen in vielen Fällen fallengelassen werden kann, ohne den TP-Dienst zu beeinträchtigen. Nur für die Commit-Phase einer Transaktion ist es offensichtlich, daß die Atomarität zwingend notwendig ist.

Um also TP parallelisieren zu können, sind weitere, ausführliche Untersuchungen notwendig, an welchen Stellen des Protokolls die Atomarität zur korrekten Erbringung des TP-Dienstes notwendig ist und an welchen nicht. Voraussetzung für solche Untersuchungen ist, daß die Grundstruktur von TP in einer parallelisierten Form spezifiziert worden ist (vorläufig unter Einhaltung der Atomarität), so daß Experimente mit einem Simulator möglich werden, um Varianten von Abläufen veanschaulichen zu können.

Daher wurden weitere Arbeiten auf diesem Gebiet solange verschoben, bis eine erste Version der parallelisierten TP-Spezifikation mit einer dazugehörigen ablauffähigen Implementation vorliegt.

5.6 Die Darstellungs- und die Sitzungsschicht

In einer ersten Phase ist es das Ziel, eine eingeschränkte Version von TP möglichst bald fertigzustellen und ausführbar zu machen. Da weiterhin eine Implementation auf einer homogenen Hardware vorgesehen ist, wird die Umkodierung per ASN.1 in der Darstellungsschicht zunächst nicht benötigt. Von der Darstellungsschicht bleibt damit nur ein Nulldienst übrig, der den Dienst der unterliegenden Schicht unverändert zur Verfügung stellt. Ebenso haben wir in Kapitel 2.2.2 bereits festgestellt, daß in der ersten Phase, solange für TP auf die Commit-Funktionseinheit verzichtet wird, auch die Sitzungsschicht nur einen Nulldienst liefern muß.

Damit können wir zunächst gänzlich auf die Spezifikation der tieferen Schichten in Estelle verzichten und das Modul `lowerLayers` als `external` spezifizieren. Die gleichen externen (C++)-Prozeduren, die später den Dienst der Transportschicht zur Verfügung stellen sollen, können im wesentlichen auch den Dienst dieses externen Moduls zur Verfügung stellen.

Sobald allerdings die Commit-Funktionseinheit realisiert wird, wird auch die Rücksetz-Funktionalität der Sitzungsschicht benötigt (vergleiche Tabelle 1 auf Seite 8), so daß dann die Darstellungs- und die Sitzungsschicht in Estelle spezifiziert sein müssen. Hierfür sind dann die vorhandenen Spezifikationen (siehe Kapitel 2.2.2) an die TP-Spezifikation anzupassen, es müssen die entsprechenden zusätzlichen (Rücksetz)-Funktionseinheiten der Sitzungsschicht in Estelle spezifiziert werden, und es muß die ASN.1-Umkodierung für TP in C++ kodiert werden, zumindest als Dummy-Umkodierung.

Möglichkeiten, inwieweit die Darstellungs- und die Sitzungsschicht in sich parallelisiert werden können, um einen Effizienzgewinn zu erhalten, werden in Hamburg nicht untersucht. (Vergleiche auch [Her91, Hof93].)

5.7 Die Anwendungsdienstelemente

5.7.1 CCR-ASE

Wie bereits erwähnt, wird das CCR-Anwendungsdienstelement CCR-ASE von TP nicht benötigt, solange die Commit-Funktionseinheit nicht ausgewählt wird. Und da diese in der ersten Stufe zunächst nicht realisiert werden soll, wurde beschlossen, ebenso auch CCR-ASE zunächst nicht zu realisieren.

Sobald aber die Commit-Funktionseinheit in TP integriert werden soll, muß CCR-ASE neu in Estelle spezifiziert werden (vergleiche Kapitel 2.2.4).

5.7.2 ACSE

Für ACSE existierte bereits eine Estelle-Spezifikation, allerdings war sie noch fehlerhaft (siehe Kapitel 2.2.1). Daher wurden die Fehler behoben und dann die Spezifikation ausführlich getestet (siehe [Lap94b]). Nunmehr besteht ein gutes Vertrauen darin, daß die Estelle-Spezifikation von ACSE korrekt bezüglich des ISO-Standards für ACSE ([ISO88a, ISO88b]) ist.

In Kapitel 5.4.4 hatten wir beschlossen, alle Komponenten, mit Ausnahme der Konkantenations- und Separations-Komponente, innerhalb eines Einzelassoziationsobjektes SAO nicht als separate Estelle-Sohnmodule zu spezifizieren, sondern in einem prozeduralen Spezifikationsstil. Daher befaßt sich die Studienarbeit [Lap94a] damit, die Spezifikation von ACSE in diesen Stil zu konvertieren.

5.7.3 TP-ASE

In Kapitel 5.3 hatten wir bereits festgestellt, daß ein rein funktionaler Zusammenhang zwischen den Ein- und Ausgaben des TP-Anwendungsdienstelementes besteht, weshalb wir in Kapitel 5.4.4 beschlossen hatten, TP-ASE nicht als Estelle-Modul, sondern prozedural zu spezifizieren. So kann der im Verhältnis zum Berechnnungsaufwand innerhalb von TP-ASE sehr große Kommunikations- und Verwaltungsaufwand erheblich verringert werden. Nebenläufigkeit zur Steuerfunktion SACF war ohnehin nicht vorhanden, so daß hier auch kein Gewinn durch eine eventuelle Parallelisierung von Estelle-Modulen hätte erzielt werden können.

5.7.4 U-ASE

Der TP-Standard macht für den Aufbau und die Funktionalität des benutzerspezifischen Dienstelementes U-ASE konsequenterweise keine Vorschriften. Nichtsdestotrotz ist die Existenz eines U-ASE erforderlich, damit TP arbeiten kann. In Kapitel 6 werden wir die Auswahl einer Last für TP beschreiben, die für Meßzwecke einen TP-Dienstnutzer simuliert. Entsprechend benötigt auch eine solche simulierte Last ein U-ASE. Dies kann allerdings sehr einfach ausfallen.

Hier bietet sich die Verwendung eines bereits existierenden Protkolls an, das im Rahmen von TP mitentwickelt wurde, des Protokolls für unstrukturierte Datenübertragung UDT (unstructured data transfer, [ISO91b]).

UDT definiert einen Datenübertragungsdienst, der es TP-Anwendungen, die nicht den OSI-Standards entsprechen, erlaubt, in der OSI-TP-Umgebung zu arbeiten. UDT ermöglicht damit die rasche Konversion von Netzwerk-Transaktionsprogrammen, die ursprünglich für Nicht-OSI-Netzwerke entworfen wurden, hin zu OSI für Programme, welche nicht die Aushandlungs- und anderen Features der OSI-Anwendungs- und Darstellungsschicht unterstützen, die die Zusammenarbeit zwischen heterogenen Systemen ermöglichen. UDT ist nicht außerhalb der TP-Umgebung nutzbar, und es ist auch nicht zur Unterstützung anderer standardisierter OSI-Anwendungen gedacht. Die abstrakte und Transfer-Syntax des nicht von OSI standardisierten Protokolls muß bei einer OSI-Behörde registriert werden, damit sie von der Darstellungschicht korrekt ausgehandelt werden kann. Alternativ können die nicht von der OSI standardisierten Anwendungs-Protokolldateneinheiten auf Oktett-Ketten abgebildet werden, welche für diesen Zweck bereits registriert sind. Die kooperierenden TP-Dienstnutzer müssen dann selbstverständlich ein a-priori-Wissen über Syntax und Semantik dieser Oktett-Ketten besitzen. [13]

Für unsere Zwecke fiel die Wahl für ein U-ASE somit auf UDT mit Oktett-Ketten, da ohnehin kein Austausch von „echten“ Daten geplant war.

[13]Dieser Absatz ist ein Auszug aus dem UDT-Standard-Entwurf [ISO91b].

Bisher ist noch keine Estelle-Spezifikation für ein solches U-ASE vorhanden, es muß noch erstellt werden. Dies wird im Rahmen der Entwicklung einer simulierten Last für TP (Kapitel 6) geschehen. Da es sich um ein sehr einfaches Protokoll handelt (nur ein einziges Dienstelement), wird der Aufwand nicht sehr groß sein.

5.8 Verwendungsgrad vorhandener Spezifikationen

In diesem Kapitel geben wir einen Überblick, welche der bereits existierenden Estelle-Spezifikationen inwieweit genutzt werden können.

Von der Estelle-Spezifikation von TP existierten zwei Versionen, die vom NIST und die von INRIA. Es wurde in Kapitel 5.4.3 entschieden, auf der NIST-Version aufzubauen, da nur sie auf der aktuellen (und endgültig standardisierten) Version von TP aufbaut. Allerdings konnten trotzdem auch von dieser Spezifikation im wesentlichen nur die Typ- und Kanaldefinitionen direkt verwendet werden, da die Modulstruktur sowohl für die Parallelisierung ungeeignet war (sehr viel Kommunikation über exportierte Variablen, die für asynchron parallel arbeitende Module nicht mehr sinnvoll sind), als auch für die Generierung einer effizienten Implementierung (Aufteilung der MACF-Komponente in je ein Sohnmodul pro Dialog, trotz synchroner Kommunikation). Weiterhin war sie noch niemals ausgeführt oder gar getestet worden und enthielt offensichtlich etliche Fehler, darunter auch schwerwiegende strukturelle Fehler (keine Atomarität von Aktionssequenzen). Eine Vereinigung mit der Version von INRIA war leider nicht möglich, obwohl dort die Modulstruktur wesentlich geeigneter war. Unter anderem waren die Namenskonventionen zu verschieden. So wurden nur Ideen aus der Architektur der INRIA-Version übernommen.

Auch von der Darstellungs- und Sitzungsschicht waren bereits Estelle-Spezifikationen vorhanden. Für eine Integration in die TP-Spezifikation muß die ASN.1-Kodierung für TP (in C++) neu geschrieben werden, zumindest als funktionsloser Platzhalter, um die Spezifikation der Darstellungsschicht ablauffähig zu machen. Sobald die Commit-Funktionseinheit realisiert wird, müssen in der Sitzungsschicht weitere Funktionseinheiten als die bisher vorhandenen Kern-Funktionseinheiten implementiert werden (siehe Kapitel 5.6 und 2.2.2).

Zum Assoziationssteuerungsdienstelement ACSE existierte ebenfalls bereits eine Estelle-Spezifikation, die nach einer Fehlerkorrektur und anschließendem Test verwendet werden konnte (vergleiche auch den Tätigkeitsbericht [Lap94b]). Um allerdings das Ziel zu erreichen, die Komponenten eines Einzelassoziationsobjektes SAO prozedural zu spezifizieren, und so die Effizienz zu steigern, wurde es notwendig, ACSE ebenfalls stark umzuschreiben (vergleiche auch die Studienarbeit [Lap94a]).

Als Fazit kann man festhalten, daß selbst die ausgewählte kleine Teilmenge der Funktionalität von TP viel Arbeit erfordert, insbesondere auch, weil die vorhandene TP-Spezifikation entgegen den Erwartungen in wesentlichen Teilen unbrauchbar war. Die Hinzunahme der Commit-Funktionseinheit kann damit erst in der zweiten Phase des Projektes erfolgen.

6 Auswahl einer Last für TP

Um mit einem parallelisierten TP Messungen ausführen zu können, ist ein TP-Dienstnutzer notwendig. Hierfür mußte eine simulierte Last entworfen werden, die in einer geeigneten und reproduzierbaren Weise den TP-Dienst nutzt.

Eine vollständige TP-Anwendung wäre für diese Zwecke in der Entwicklung viel zu aufwendig gewesen. Stattdessen wurde ein generisches Beispiel entworfen, das von seiner TP-Dienstnutzung her möglichst viele typische Fälle abdeckt. Auf diese Weise können sogar die typischen Belastungen durch mehrere, gänzlich verschiedene TP-Anwendungen untersucht werden.

In der ersten Stufe, in der die Commit-Funktionseinheit, also die Transaktionssicherung, noch nicht unterstützt wird, handelt es sich bei der generischen Last im wesentlichen um den Aufbau eines Dialogbaumes, die Übertragung von Nutzerdaten und den Abbau des Dialogbaumes.

Um dabei verschiedene Arten von Nutzern nachzubilden, werden entweder kurze, mittlere oder sehr große Nutzerdaten übertragen. Außerdem wurde zum Vergleich auch ein Fall betrachtet, in dem gar keine Daten übertragen werden, sondern nur der Dialogbaum auf- und abgebaut wird. Mit diesem Fall können die Auswirkungen der Parallelisierung auf den reinen TP-Kommunikationsaufwand separat studiert werden.

In dem Tätigkeitsbericht [Lan94] wird detailliert beschrieben, welche typischen Szenarien von Dialogbäumen ausgewählt wurden, was sie jeweils für Eigenschaften besitzen, und wie sie in Estelle spezifiziert wurden.

7 Prototypische Implementation von TP

7.1 Auswahl eines Werkzeugs

Um aus einer Estelle-Spezifikation eine ablauffähige Implementation generieren zu können, ist ein geeignetes Werkzeug notwendig. In Hamburg standen bei Projektbeginn vier Estelle-Compiler zur Verfügung.

ec von Bull Dieser Compiler ist ein kommerzielles Produkt der Firma Bull in Frankreich, das insbesondere zur Simulation gedacht ist. Er erzeugt sequentiellen Code für eine einzige Maschine, und er ist Teil eines recht umfangreichen Paketes zur Unterstützung der Entwicklung von Spezifikationen. Unter anderem enthält dieses Paket auch einen leistungsfähigen Debugger, edb. Der Quelltext des Compilers und der anderen Werkzeuge des Paketes ist nicht erhältlich.

NIST-Compiler Entwickelt wurde dieser Compiler vom National Institute of Standards und Technology der Vereinigten Staaten von Amerika ([FHSW89]). Er erzeugt ebenfalls sequentiellen Code für eine einzige Maschine. Es existieren keine unterstützenden Werkzeuge, allerdings ist der Quelltext des Compilers frei erhältlich.

UHH-Compiler Der UHH-Compiler wurde im Rahmen der Diplomarbeit [Pet91] (siehe auch [KrGo93]) an der Universität Hamburg auf der Basis des NIST-Compilers entwickelt. Er kann Code für die verteilte Ausführung von Spezifikationen auf mehreren Rechnern erzeugen. Sein Quelltext ist frei erhältlich. Allerdings enthält er keine Unterstützung für das Debugging, und seine Steuerung ist, wie die aller bisher erwähnten Compiler, ausschließlich textorientiert.

Pet/Dingo Pet und Dingo wurden ebenfalls vom NIST entwickelt ([SiSt90]). Der portable Estelle-Übersetzer Pet ([SiSt91a]) analysiert die Syntax einer Estelle-Spezifikation und erzeugt einen objektorientierten Zwischencode, der für weitere Werkzeuge zur Verfügung steht. Das einzige Werkzeug, daß bisher diesen Zwischencode weiterverarbeitet, ist der Generator für verteilte Implementationen Dingo ([SiSt91b]). Dingo kann Code für die verteilte Ausführung der Spezifikation auf mehreren Rechnern erzeugen. Dabei besteht die Option, das Fenstersystem X-Windows zu unterstützen. Mit dieser Option ist es möglich, zu jeder Modulinstanz ein Fenster zu öffnen, über das der Zustand der Modulinstanz inspiziert werden kann und in (sehr) begrenztem Maße in den Ablauf eingegriffen werden kann. Weiterhin ist der Quelltext von Pet und Dingo frei erhältlich, und es ist auch einfach möglich, in der Estelle-Spezifikation offengelassene Implementationsdetails (z.B. `primitive`-Funktionen) mit in der Programmiersprache C++ geschriebenen komfortablen X-Windows-Fensterobjekten auszufüllen. Schließlich sind Pet/Dingo und X-Windows auf eine inzwischen recht große Anzahl von Hardware-Plattformen portiert worden, unter anderem auch auf den Hochleistungs-Parallelrechner KSR/1 der Universität Mannheim ([FiHo93]).

Zur Erzeugung von verteilt und parallel ablaufenden Implementationen sind all diejenigen Compiler ungeeignet, die keinen Code zur verteilten Ausführung auf verschiedenen Rechnern erzeugen können, also ec von Bull und der NIST-Compiler. Leider hatte das zur Folge, daß damit auch bereits der einzige Compiler ausgeschieden war, der Debugging unterstützte. Trotzdem wird dieses Werkzeug im Projekt eingesetzt, um einzelne Estelle-Spezifikationen zu testen (siehe Kapitel 5.7.2).

Beide verbleibenden Compiler sind für die Integration von Estelle-Erweiterungen geeignet (vergleiche Kapitel 5.5.1 und 5.4.4), da sie als Quellcode vorliegen, für Pet/Dingo spricht aber ein zusätzliches Argument, das der graphischen

Schnittstelle. Gerade bei so extrem großen Spezifikationen wie der von TP ist es sehr schwer, mit einem textorientierten Debugger den Überblick zu behalten. Eine graphische Fensteroberfläche leistet bei der komplexen und dynamischen Modulstruktur von TP unschätzbare Dienste. Zwar bot auch Pet/Dingo nur marginale Debuggingmöglichkeiten, aber dank einer recht komfortablen Schnittstelle zur X-Windows-Programmierung war es mit vertretbarem Aufwand möglich, die wichtigsten Operationen selbst zu implementieren.

Es wäre selbstverständlich wesentlich günstiger gewesen, die Entwicklung des parallelisierten TP mit dem Debugger edb von Bull durchzuführen, und nur für die Messungen eine verteilte Implementation mit Pet/Dingo zu erzeugen. Leider stellte es sich schnell heraus, daß zur Spezifikation von TP, um sie ablauffähig zu machen, noch einige hundert `primitive`-Funktionen und noch mehr nicht ausspezifizierte Konstanten und Datentypen in einer Programmiersprache ergänzt werden mußten. Und da die Aufrufschnittstellen der beiden Compiler-Systeme hier sehr unterschiedlich sind, hätten diese alle noch einmal neu entwickelt und getestet werden müssen.

So fiel die Entscheidung für einen Estelle-Compiler zugunsten des Pet/Dingo/X-Windows-Systems.

7.2 Notwendige Arbeiten für die Implementation

Parallel zum Entwurf der Spezifikation der Architektur des parallelisierten TP wurde bereits auch eine Implementation generiert. Einerseits bot die Syntaxprüfung des Pet-Werkzeugs hilfreiche Unterstützung. Aber darüber hinaus konnte mit dem Dingo-Werkzeug die aktuelle Arbeit sofort in ihrer dynamischen Struktur visualisiert werden, wie in Kapitel 7.1 beschrieben.

Auf dieser Implementation kann später eine Meßreihe über die Auswirkungen der Parallelisierung auf die Laufzeit basieren, wenn die X-Windows-Schnittstelle wieder entfernt wird.

Für die automatische Implementation der Estelle-Spezifikation mußten trotz alledem noch von Hand einige Dateien hinzugefügt werden, da die in ihnen enthaltenen Informationen nicht in Estelle ausdrückbar sind oder nicht in Estelle ausgedrückt werden sollen.

Hauptsächlich betrifft dies die sogenannten „lokalen Entscheidungen". Der TP-Standard sieht vor, daß einzelne Komponenten bestimmte Entscheidungen fällen dürfen bzw. müssen, die nicht (direkt) die Interoperation verteilter Protokollinstanzen betreffen und damit nicht unter den Standard fallen. Beispiele wären die Entscheidung, eine ganz neue Transaktion zu beginnen und einen Wurzelknoten zu erzeugen, oder auch, die Erzeugung eines weiteren (untergeordneten) Knotens zu verweigern, weil etwa kein Speicherplatz mehr verfügbar ist. All diese Entscheidungen werden daher nicht in Estelle spezifiziert, sondern es wird lediglich ein Aufruf einer `primitive`-Funktion spezifiziert, der als externe Schnittstelle der Spezifikation zu lokalen Entscheidungsfunktionen außerhalb des

Standards dient.

Diese `primitive`-Funktionen waren alle in C++ zu schreiben, damit der von dem Pet/Dingo-Werkzeug erzeugte Kode ausführbar wurde.

Unter die lokalen Entscheidungen fielen auch die zusätzlichen interaktiven Beeinflussungsmöglichkeiten, die zum Test der Spezifikation vorgesehen wurden. So wurde die vorhandene X-Windows-Schnittstelle erweitert, um in die X-Windows-Fenster Knöpfe für einige der lokalen Entscheidungen zu integrieren. Gerade für die Entscheidung, ob etwa ein neuer TP-Wurzelknoten erzeugt werden sollte, war dies wichtig, ebenso auch für einige weitere Entscheidungen, die den Protokollablauf maßgeblich beeinflussen.

Außer den `primitive`-Funktionen waren natürlich auch noch die Typdefinitionen zu ergänzen, die in der Estelle-Spezifikation bewußt offengelassen worden waren, indem das „...“-Konstrukt verwendet wurde. (Dies betraf Datentypen, die in anderen Standards oder in der Anwendung definiert werden.)

Schließlich mußte auch noch die Verteilung der Estelle-Modulinstanzen auf verschiedene Rechner angegeben werden. Hierfür sieht das Pet/Dingo-Werkzeug bereits spezielle Pseudo-Kommentare im Estelle-Quelltext vor. Diese mußten integriert werden, und in einem Falle wurde auch noch eine C++-Funktion geschrieben, da zu einem Modul mehrere Modulinstanzen auf verschiedenen Rechnern erzeugt werden sollten, deren Namen diese Funktion zu liefern hatte.

7.3 Implementation mit Pet/Dingo

Einzelheiten der Implementation mit den Pet- und Dingo-Werkzeugen sind im Anhang A beschrieben.

7.4 Sequentielles TP für vergleichende Messungen

Ursprünglich war vorgesehen, die vorhandene Version von TP einfach nur zu parallelisieren, so daß damit zwei Versionen vorhanden gewesen wären, deren Effizienz in Messungen direkt hätte verglichen werden können. Da sich aber nun herausstellte, daß die sequentielle Version nicht nur fehlerhaft und nicht ablauffähig war, sondern von ihrer Struktur her nicht zu parallelisieren war, stand damit erst einmal nur noch die parallelisierte Version der Estelle-Spezifikation von TP in einer ablauffähigen Form zur Verfügung. Es hätte erheblichen Aufwand bedeutet, die sequentielle Version in einen sowohl korrekten als auch ablauffähigen Zustand zu bringen. Daher wurde beschlossen, als sequentielle Referenzversion eine modifizierte Version der parallelen Version zu verwenden, in der alle Parallelität entfernt wurde. Eine solche sequentielle Version läßt sich mit dem verwendeten Estelle-Compiler am einfachsten dadurch gewinnen, daß die Verteilung der Estelle-Modulinstanzen auf verschiedene Rechner dahingehend geändert wird, daß alle Modulinstanzen eines „offenen Systems“ auf nur noch einem einzigen Rechner ablaufen. Dann sorgt der vorhandene Scheduling-Mechanismus automatisch dafür, daß die gesamte Implementation sowohl korrekt als auch sequentiell abläuft.

Literatur

[AGS93] Andrae, C., Gotzhein, R. und Sédillot, S. *An evolutionary approach to the development of complex protocol standards*. In Danthine, A., Leduc, G. und Wolper, P. (Hrsg.), „13. IFIP Symposium on Protocol Specification, Testing and Verification — PSTV '93", S. ???–???, Lüttich, Belgien (25.–28. Mai 1993). North-Holland.

[And91] Andrae, C. *Spezifikation der Commit-Funktion des OSI-Transaktionsverarbeitungsprotokolls (OSI-TP) in Estelle*. Diplomarbeit, Universität Hamburg, FB Informatik (Juli 1991).

[Bre93] Bredereke, J. *Eine Estelle-Erweiterung für strukturiertere Spezifikationen und für einen neuen Spezifikationsstil*. In Vorbereitung (1993).

[BrGo93a] Bredereke, J. und Gotzhein, R. *Eine Estelle-Erweiterung zur Steigerung der Nebenläufigkeit*. Mittlg. FBI-HH-M-219/93, Universität Hamburg, FB Informatik (Feb. 1993).

[BrGo93b] Bredereke, J. und Gotzhein, R. *Increasing the concurrency in Estelle*. In Tenney, R. L., Amer, P. D. und Uyar, M. Ü. (Hrsg.), „Sixth International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols — FORTE '93", S. ???–???, Boston, Mass. (26.–29. Okt. 1993). North-Holland.

[CCI87] CCITT SG X, Contribution Com X-R15-E. *Recommendation Z.100: Specification and Description Language SDL* (1987).

[DeBu89] Dembinski, P. und Budkowski, S. *Specification language Estelle*. In Diaz, M. u. a. (Hrsg.), „The Formal Description Technique Estelle", S. 35–75. North-Holland (1989).

[Dem90] Dembinski, P. *Interfacing ASN.1 and Estelle: A practical approach*. In Quemada, J., Maños, J. und Vazquez, E. (Hrsg.), „Third International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols — FORTE '90", S. 445–468, Madrid, Spanien (5.–8. Nov. 1990). North-Holland.

[Des86] Desantis, R. *Parallel processing, the new computing, is promising super-economical telecom systems*. Comm. News **23**(12), 36–37 (1986).

[FaMe81] Faro, A. und Messina, G. *Parallel processing in computer communications*. In „International Conference on Parallel Processing", S. 294–296, Columbus, OH, USA (1981).

[FHSW89] Favreau, J.-P., Hobbs, M., Strausser, B. und Weinstein, A. *User guide for the NIST prototype compiler for Estelle*. Interner Bericht ICST/SNA–87/3, Institute for Computer Science and Technology, National Institute for Standards and Technology (Sep. 1989).

[FiHo93] Fischer, S. und Hofmann, B. *An Estelle compiler for multiprocessor platforms*. In Tenney, R. L., Amer, P. D. und Uyar, M. Ü. (Hrsg.), „Sixth International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols — FORTE '93", S. ???–???, Boston, Mass. (26.–29. Okt. 1993). North-Holland.

[Her91] Herbert, S. *Entwurf und Implementierung eines parallelen ASN.1 Encoders und Decoders*. Diplomarbeit, Universität Mannheim, FB Informatik (Dez. 1991).

[Hof93] Hofmann, B. *Integration von Darstellungs- und Kommunikationssteuerungsschicht in Estelle*. In Gerner, N., Hegering, H.-G. und Swoboda, J. (Hrsg.), „Kommunikation in verteilten Systemen", S. 560–573, München (1993). GI/ITG-Fachtagung, Springer.

[ISO81] ISO/TC 97/SC 16, ISO 7498. *Data Processing — Open Systems Interconnection — Basic Reference Model* (1981).

[ISO87] ISO/TC 97, ISO 8326. *Information processing systems — Open Systems Interconnection — Basic connection oriented session service definition* (1987).

[ISO88a] ISO/TC 97, ISO 8649. *Information processing systems — Open Systems Interconnection — Service definition for the Association Control Service Element* (1988).

[ISO88b] ISO/TC 97, ISO 8650. *Information processing systems — Open Systems Interconnection — Protocol specification for the Association Control Service Element* (1988).

[ISO88c] ISO/TC 97/SC 21, ISO 8807. *Information Processing Systems — Open Systems Interconnection — Lotos: A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour* (1988).

[ISO89a] ISO/IEC JTC 1, IS 9545. *Information technology — Open Systems Interconnection — Application Layer Structure* (1989).

[ISO89b] ISO/TC 97/SC 21, ISO 9074. *Information Processing Systems — Open Systems Interconnection — Estelle: A Formal Description Technique Based on an Extended State Transition Model* (1989).

[ISO91a] ISO/IEC JTC 1/SC 21, IS 9545/PDAM 1. *Information technology — Open Systems Interconnection — Application Layer Structure — Proposed Draft Amendment 1: Extended Application Layer Structure* (1991).

[ISO91b] ISO/IEC JTC 1/SC 21, Commitee Draft CD 10026-6. *Information technology — Open Systems Interconnection — Distributed Transaction Processing — Part 6: Unstructured Data Transfer* (1991).

[ISO92] ISO/IEC JTC 1/SC 21, IS 10026. *Information technology — Open Systems Interconnection — Distributed Transaction Processing* (1992).

[Ker92] Kerner, H. (Hrsg.). *Rechnernetze nach OSI*. Addison-Wesley (1992).

[KrGo93] Kreuz, D. und Gotzhein, R. *A compiler for the parallel execution of Estelle specifications*. In König, H. (Hrsg.), „Formale Methoden für Verteilte Systeme“, Bd. 8 aus „FOKUS-Band“, GI/ITG-Fachgespräch Juni 1992 (1993). Saur-Verlag.

[Lan94] Lange, T. *Entwurf einer Last für OSI-TP und Spezifikation in Estelle*. In Vorbereitung (1994).

[Lap94a] Lapok, M. *Anpassung und Korrektur einer Estelle-Spezifikation zur Assoziationssteuerung*. Studienarbeit Nr. xxx, Universität Hamburg, FB Informatik (1994). In Vorbereitung.

[Lap94b] Lapok, M. *Validation der Estelle-Spezifikation des ACSE-Anwendungsdienstelementes*. In Vorbereitung (1994).

[Pet91] Peter, D. *Entwurf, Realisierung und Integration eines Protokolls zur verteilten Ausführung von Estelle-Spezifikationen*. Diplomarbeit, Universität Hamburg, FB Informatik (Feb. 1991).

[SiSt90] Sijelmassi, R. und Strausser, B. *NIST integrated tool set for Estelle*. In Quemada, J., Maños, J. und Vazquez, E. (Hrsg.), „Third International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols — FORTE ’90“, Madrid, Spanien (5.–8. Nov. 1990). North-Holland.

[SiSt91a] Sijelmassi, R. und Strausser, B. *The Portable Estelle Translator: an overview and user guide*. Tech. Rep. NCSL/SNA—91/2, U. S. Dept. of Commerce, National Institute of Standards and Technology, Gaithersburg, MD (Jan. 1991).

[SiSt91b] Sijelmassi, R. und Strausser, B. *The Distributed Implementation Generator: an overview and user guide*. Tech. Rep. NCSL/SNA—91/3,

U. S. Dept. of Commerce, National Institute of Standards and Technology, Gaithersburg, MD (Jan. 1991).

[Zit89a] Zitterbart, M. *High-speed protocol implementation based on a multiprocessor-architecture.* In Rudin, H. und Williamson, R. (Hrsg.), „Protocols for High-Speed Networks“, Amsterdam, Niederlande (1989). North-Holland.

[Zit89b] Zitterbart, M. *A multiprocessor architecture for high speed network interconnections.* In „IEEE Infocom ’89“, Bd. 1, S. 212–218, Washington, DC, USA (1989).

[Zit89c] Zitterbart, M. *A parallel architecture for transport systems and gateways.* In Kühn, P. J. (Hrsg.), „Kommunikation in verteilten Systemen“, S. 744–457, Berlin (1989). Springer.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abk.	Englisch	Deutsch
...I	invocation of ...	Vorkommnis von ...
ACSE	association control service element	Assoziationssteuerungs-Dienstelement
AE	application entity	Anwendungsinstanz
AEI	AE invocation	Anwendungsvorkommnis
ALS	application layer structure	Struktur der Anwendungschicht
ASE	application service element	Anwendungsdienstelement
ASN.1	abstract syntax notation one	abstrakte Notation für Syntax, eins
ASO	application service object	Anwendungsdienstobjekt
CCR	concurrency, commitment and recovery	Nebenläufigkeit, Commitment und Wiederherstellung
CCR-ASE	ASE for CCR	ASE für CCR
CF	control function	Steuerfunktion
CPM	channel protocol machine	Kanal-Protokollmaschine
CPM-MACF	MACF of the CPM	MACF der CPM
DFG	a German research funding board	Deutsche Forschungsgemeinschaft
Dingo	Distributed Implementation Generator	Generator für verteilte Implementationen
FTAM	file transfer, access and management	Dateiübertragung, -zugriff und -management
FU	functional unit	Funktionseinheit
INRIA	a French research institute	ein französisches Forschungsinstitut: Institut National de la Recherche de l'Informatique et de l'Automatisation
ISO	International Standardization Organization	eine internationale Standardisierungsorganisation
ISODE	ISO development environment	ISO-Entwicklungs-umgebung
MACF	multiple association control function	Mehrassoziationssteuer-funktion
NIST	National Institute for Standards and Technology	ein amerikanisches Forschungsinstitut

OSI	open systems interconnection	Verbindung offener Systeme
P-SAP	presentation SAP	Darstellungs-SAP
Pet	Portable Estelle Translator	Portierbarer Estelle-Übersetzer
PM	protocol machine	Protokollmaschine
ROSE	remote operations service element	Dienstelement zur entfernten Ausführung von Programmen
SACF	single association control function	Einzelassoziationssteuer-funktion
SAO	single association object	Objekt zur Verwaltung einer Assoziation
SAP	service access point	Dienstzugangspunkt
TP	(distributed) transaction processing	(verteilte) Transaktionsverarbeitung
TP-ASE	ASE for TP	ASE für TP
TPPM	TP protocol machine	TP-Protokollmaschine
TPSP	TP service provider	TP-Diensterbringer
TPSU	TP service user	TP-Dienstnutzer
TPSUI	TPSU invocation	TPSU-Vorkommnis
U-ASE	user ASE	Dienstnutzer-ASE
UDT	unstructured data transfer	unstukturierte Datenübertragung
XALS	extended application layer structure	Erweiterte Struktur der Anwendungschicht

A Details der Implementation mit Pet/Dingo

A.1 Übersetzungsverwaltung

Allein die Estelle-Spezifikation ist in weit über zwanzig Dateien aufgeteilt, weitere handgeschriebene Dateien kommen dazu, und die Anzahl der automatisch generierten Dateien beträgt ein vielfaches. Daher war es unumgänglich, eine geeignete Übersetzungsverwaltung zu verwenden, um nach Änderungen sicherzustellen, daß alle notwendigen Teile neu übersetzt werden (aber auch nicht mehr, der bereits deutlichen Übersetzungszeit wegen). Daher wurde auf das Standard-Unix-Werkzeug `make` zurückgegriffen, welches für diese Aufgabe sehr gut geeignet ist. `make` entnimmt die notwendige Information über die Abhängigkeiten und die auszuführenden Übersetzungsanweisungen einer speziellen Datei, dem *Makefile*. Anhand der Zeitmarken aller Dateien kann es dann die aktuell notwendigen Übersetzungsanweisungen bestimmen.

Das Dingo-Werkzeug generiert bereits automatisch ein Makefile, mit dessen Hilfe die von ihm erzeugten Dateien von einem C++-Compiler übersetzt werden können. Leider enthält dieses Makefile keine Informationen, wie die vom Pet-Werkzeug erzeugte Zwischendatei vom Estelle-Quelltext abhängt, und wie die Abhängigkeiten von den handgeschriebenen C++-Dateien sind.

Daher wurde ein zweites Makefile geschrieben, welches ausschließlich diese Informationen enthält und für seine letzte Bearbeitungsstufe schließlich das erste Makefile aufruft. Dies war notwendig, da die Abhängigkeiten bei der TP-Spezifikation so komplex sind, daß man sie nicht mehr leicht im Kopf überschauen kann. Darüberhinaus wurden u.a. auch Regeln integriert, um die generierte Implementation automatisch auf einer Menge von Rechnern zu starten, einschließlich des Starts der notwendigen lokalen Site-Server. Da dieses zweite Makefile im Prinzip für jede Estelle-Spezifikation nützlich ist, wurde es in einer allgemeinen Form geschrieben. Es müssen am Anfang der Datei nur einige Definitionen ausgetauscht werden, damit sie für beliebige andere Spezifikationen verwendbar ist. Als Name wurde darum `gmakefile` für *generisches* Makefile gewählt.

Die Aufgabe der Datei `gmakefile` ist recht komplex, und so benötigt sie noch drei weitere Hilfsdateien. Normalerweise ist es nach dem Lauf des Dingo-Werkzeugs notwendig, drei Dateien von Hand zu editieren:

- Das erzeugte Makefile namens `_*.make` selbst, um die Abhängigkeiten von weiteren, handgeschriebenen Dateien einzutragen,
- die Datei `_*.bindirs`, die die Namen und Adressen aller beteiligten Rechner enthalten soll, und schließlich
- die datei `_*.impl.hxx`, in die die noch undefinierten Datentypen aus den `...`-Definitionen eingetragen werden müssen.

(„*“ steht hier für den Namen der jeweiligen Spezifikation.) Diese Editier-Aufgaben werden nunmehr alle von dem generischen Makefile erledigt, indem es den Unix-Streameditor `sed` benutzt. Die Editieranweisungen finden sich genau in den erwähnten drei Zusatzdateien. Ihre Namen sind entsprechend `gmake.make.sed`, `gmake.bdir.sed` und `gmake.impl.sed`.

Es wäre auch möglich gewesen, die Editieranweisungen einmal von Hand auszuführen, die Datei an einen sicheren Ort zu kopieren und nach jedem Dingo-Lauf zurückzukopieren. Allerdings führt diese Methode bei Spezifikationen dieser Größe zu Problemen. Denn es kommt in der Entwicklungsphase sehr häufig zu Änderungen an den betroffenen Dateien, so daß man jedes Mal von Hand diese Änderungen suchen, finden und in die gesicherte Datei übertragen müßte. Dies wäre sehr fehlerträchtig. Die Methode mit dem Streameditor hat dagegen den Vorteil, daß in die zu editierenden Dateien weitere Zeilen eingefügt werden dürfen, ohne daß die Editieranweisungen verändert werden müssen. Die Praxis hat bereits gezeigt, daß dies richtig ist und Handarbeit überhaupt nur noch notwendig ist, wenn in die drei `*.sed`-Dateien explizit neue Editieranweisungen eingetragen werden müssen. Dieses wiederum gestaltet sich recht einfach, da jede Anweisung mit einem Suchbefehl beginnt, der den richtigen Kontext lokalisiert. So muß man üblicherweise nur eine vorhandene Zeile nehmen, kopieren und für einen neuen Bezeichner modifizieren.

Weitere Einzelheiten von `gmakefile` und die Aufrufsyntax von `make` sind in der Datei `gmakefile.README` dokumentiert.

A.2 Dateien der Implementation

Auf Seite 27 hatten wir bereits die Aufteilung des Spezifikationstextes in mehrere Dateien diskutiert und in Abbildung 7 auf Seite 30 auch den Baum der `#include`-Beziehungen dieser Dateien angegeben. Die Konventionen für die Dateiendungen der Estelle-Spezifikation hatten wir bereits in Tabelle 5 auf Seite 29 vorgestellt. Hier wollen wir nun auf die restlichen Dateien eingehen.

Vorweggestellt sei vielleicht noch, daß sämtliche Dateien sowohl der Spezifikation als auch die von Hand ergänzten als auch die generierten in dem selben Verzeichnis abgelegt wurden. Als Name für das Verzeichnis wurde `tp` gewählt.

A.2.1 Handgeschriebene Dateien

Außer den bereits in Kapitel A.1 beschriebenen Dateien um das `gmakefile` herum gibt es zu einigen Modulrumpfdefinitionen `*Body` jeweils eine Datei `_*Body.prims.cxx`, die den C++-Kode für die außerhalb von Estelle zu spezifizierenden `primitive`-Funktionen und -Prozeduren enthält.

Die Datei `_World.impl.cxx` enthält von Hand geschriebene Implementationen für einige Datentypen, deren Definitionen in der Estelle-Spezifikation offengelassen wurden, und für die aufgrund ihrer Länge auch in den Dateien

`_*.impl.hxx` nur die C++-Zugriffsschnittstelle definiert wurde (vergleiche auch Kapitel A.1). Außerdem enthält sie die Definition der am Ende von Kapitel 7.2 erwähnten C++-Funktion zur Erzeugung der Rechnernamen für die Verteilung der Modulinstanzen auf verschiedene Rechner.

Für die generische Last für den TP-Dienst wurde die Datei `tpworld.config` geschrieben. Damit Messungen automatisch ablaufen können, müssen die Parameter wie etwa Testfallnummer und Anzahl der Wiederholungen an die Last übergeben werden. Dies geschieht zur Zeit dadurch, daß die Last eine `primitive`-Funktion aufruft, die diese Datei einliest. (Diese Lösung kann evtl. noch verändert werden, wenn die detaillierte Vorbereitung der Messungen beginnt. Aber so wurde vorläufig das Problem gelöst, Parameter von außen zu übergeben, ohne eine interaktive Schnittstelle zu benutzen und ohne die Spezifikation neu zu übersetzen.) Die `primitive`-Funktion wertet nur die erste Zeile der Datei aus, die weiteren Zeilen enthalten daher eine Dokumentation der Parameter und ihrer Formate.

A.2.2 Automatisch erzeugte Dateien

Das Pet-Werkzeug erzeugt aus der Datei `tpworld.e` eine Datei `tpworld.obj` und mehrere Dateien `*.eLIS`. Erstere ist der objektorientierte Zwischenkode, der als Eingabe für das Dingo-Werkzeug dienen wird, letztere enthalten den Quelltext mit eingestreuten Fehlermeldungen, sofern vorhanden. Außer für die Wurzeldatei `tpworld.e` gibt es auch für jede mit dem `#include`-Konstrukt eingeschlossene `*.e`-Datei eine `*.eLIS`-Datei. Zusätzlich fängt das `gmakefile` sämtliche auf den Schirm ausgegebenen Fehlermeldungen auf und leitet sie in die Datei `tpworld.err` um. Dort kann man später in Ruhe nur die Fehlermeldungen ansehen, ohne erst lange suchen zu müssen, und ohne Gefahr zu laufen, daß Meldungen ungesehen vom Schirm rollen.

Das Dingo-Werkzeug erzeugt zu jeder Modulrumpfdefinition `*Body` (sofern sie als eigener Betriebssystemprozeß ablaufen soll, was aber hier immer der Fall ist) die folgenden Dateien: `_*Body.cxx`, `_*Body.hxx`, `_*Body.ch.cxx`, `_*Body.ch.hxx`, `_*Body.tc.cxx`, `_*Body.tc.hxx` und `_*Body.main.cxx`. Die Datei `_*Body.dummy` ist nur für das vom Dingo-Werkzeug generierte Makefile relevant. Außerdem erzeugt Dingo für das Spezifikationsmodul `world` die Datei `_World.impl.hxx` (vergleiche für diese Datei auch Kapitel A.1). Die genaue Bedeutung dieser Dateien kann der Dingo-Dokumentation [SiSt91b] entnommen werden.

Aus den von Dingo generierten Dateien erzeugt der C++-Compiler (zur Zeit Gnu-C++ in der *alten Version* 1.40[14]) Dateien mit den Namen `_*Body.o`, die in einer Library `_World.a` zusammengefaßt werden. Hieraus erzeugt schließlich der Linker die ausführbaren Dateien `_*Body`.

[14]Die Benutzung neuerer Versionen ist *nicht* möglich, da die Behandlung von Streams verändert wurde.

A.3 Ausführen der Implementation

Wie in Kapitel A.1 bereits kurz erwähnt, ermöglicht es das generische Makefile, die Implementation vollautomatisch zu starten (sobald die Dateipfade und Rechnernamen in der Datei `gmakefile` entsprechend gesetzt worden sind). Es werden auf allen beteiligten Rechnern die notwendigen Site-Server erzeugt, ihre Adressen werden eingesammelt und den jeweils anderen Site-Servern zur Verfügung gestellt, und schließlich wird die ausführbare Datei der Spezifikation selbst auf dem vorher gewählten Rechner gestartet. Danach kann man durch Öffnen der entsprechenden X-Windows-Fenster die aktuelle Modulstruktur betrachten, über einige zusätzlich eingefügte Knöpfe an diesen Fenstern die Modulstruktur verändern und, sofern mittels der Datei `tpworld.config` (vergleiche Kapitel A.2.1) die interaktive Last für TP gewählt wurde, auch TP-Dienstprimitive von Hand durch Anklicken der entsprechenden (hinzugefügten) Knöpfe versenden. Das Ansehen der Inhalte von Nachrichtenwarteschlangen unterstützt die X-Windows-Schnittstelle ohnehin.

Die Aufrufsyntax für das Starten ist (vergleiche auch die weitere Dokumentation in der Datei `gmakefile.README`):

```
make -f gmakefile start
```

B Namenskonventionen

B.1 Namenskonventionen des TP-Standards

Dieses Kapitel ist ein übersetzter und überarbeiteter Auszug aus dem TP-Standard [ISO92].

(Fast) jede Zustandsmaschine verwendet Variablen, um bestimmte Informationen zu verfolgen, und sie verwendet Variablen und Prädikate als Schaltbedingungen und Prädikatenausdrücke. Die Variablen sind vom Typ Boolean, Integer, Oktett-Kette und Verbund. (Die Namen letzterer beginnen mit „T“.)

Es gibt sechs Kategorien von Variablen:

Dialogvariablen (deren Namen mit „D“ beginnen) sind einer der TPPM-MACF-Zustandsmaschinen zugeordnet, die einen einzelnen Dialog verwaltet.

Kanalvariablen (deren Namen mit „C“ beginnen) sind einer der CPM-MACF-Zustandsmaschinen zugeordnet, die einen enzelnen Kanal verwaltet.

Knotenvariablen (deren Namen mit „N“ beginnen) sind der gesamten TPPM-MACF-Zustandsmaschine zugeordnet und werden von allen Dialogen geteilt.

Systemvariablen (deren Namen mit „S“ beginnen) sind allen TPPM- und CPM-MACF-Zustandsmaschinen eines Systems zugeordnet und werden von diesen geteilt, sie behalten ihren Wert auch über einen Knoten-Crash hinaus.

Lokale-Entscheidungs-Variablen (deren Namen mit „Ld“ beginnen) repräsentieren lokale Entscheidungen und Optionen. Aufeinanderfolgende Auswertungen Lokaler-Entscheidungs-Variablen können verschiedene Ergebnisse zeitigen. Dieses nichtdeterministische Verhalten modelliert potentielle Veränderungen der Systemresourcen und der lokalen Strategien.[15]

Assoziationsvariablen (deren Namen mit „A“ beginnen) sind einer bestimmten Assoziation zugeordnet. Diese Variablen werden von der Einzelassoziationssteuerfunktion SACF benutzt, allerdings werden einige dieser Variablen auch mit der Mehrassoziationssteuerfunktion MACF geteilt, solange eine Verbindung zu dieser MACF besteht.

Prädikate (deren Namen mit „P“ beginnen) können von jeder der Zustandsmaschinen inspiziert werden und repräsentieren Bedingungen außerhalb der TPPM.

[15] Im TP-Standard merkt an dieser Stelle ein Kommentar an, daß die beiden vorhergehenden Sätze das Beste des ganzen Standards seien. Dem ist nichts hinzuzufügen.

Auf den beschriebenen ersten Buchstaben eines Variablen- (oder Prädikat-) Namens folgen ein bis etwa fünf weitere, die eine Abkürzung der Bedeutung darstellen. Es wurden recht kurze Buchstabenkombinationen gewählt, damit die Namen in die (kleinen) Kästchen der (umfangreichen) Zustandstabellen passen.

Auch für Aktionen der Zustandsmaschinen gibt es Namen (die dann als Abkürzungen in den Tabellen zu finden sind). Wenn sich die Aktion auf einen Dienst bezieht, ergibt sich der erste Buchstabe aus Tabelle 6.

Erster Buchstabe	Dienst
A	AF-Dienst
C	CAF-Dienst
P	P-Dienst
S	SAF-Dienst
T	TP-Dienst
U	U-Dienst

Tabelle 6: Namenskonvention von TP: Der erste Buchstabe einer Aktion, die sich auf einen Dienst bezieht.

Diese Konvention und auch die Konventionen für die weiteren Buchstaben sind in Anhang A.2.4.2 des Teils 3 des TP-Standards [ISO92] beschrieben.

Wenn sich die Aktion auf eine Variable bezieht, beginnt der Name der Aktion mit „V“. Außerdem gibt es noch einige Aktionen mit frei gebildeten Namen, siehe den Anhang A.2.4.4 des Teils 3 des TP-Standards [ISO92].

Die Umkodierungsaktionen der TP-ASE sind im Anhang A.3.4 des Teils 3 des TP-Standards [ISO92] beschrieben, ihre Namen beginnen entweder mit „Dec“ oder mit „Map“.

B.2 Namenskonventionen in der TP-Spezifikation des NIST

Die im TP-Standard beschriebenen Namen (siehe Anhang B.1) sind im wesentlichen direkt in die Estelle-Spezifikation von TP des NIST übernommen worden. Allerdings gibt es hier gelegentlich etwas willkürliche Ausnahmen, bei denen einige dieser Namen durch lange, sprechende Namen ersetzt wurden.

Beispiele sind etwa die Variablen Naaid und Nbrid, die in der Spezifikation als Parameter der Dialogmoduldefinition realisiert sind und dort die Namen `actionId` und `branchId` tragen.

Wenn keine Namen von TP übernommen werden (mußten), wurde eine eigene Strategie zur Namensbildung verfolgt. Aus der Bedeutung des bezeichneten Objektes wurden einige Worte bzw. Wortfragmente ausgewählt und ein langer, sprechender Name gebildet. Dabei wurde Kleinschrift verwendet, und die Stellen, an denen eine Zusammensetzung zweier Worte stattfand, wurden durch die

Großschreibung des ersten Buchstabens des folgenden Wortes gekennzeichnet. Unterstriche wurden vermieden (und damit Zeichen eingespart). Beispiel:

```
diesIstEinBeispiel
```

Die Großschreibung des folgenden Wortes fand nicht statt, wenn das vorige Wort mit einer Ziffer endete.

Nur Konstanten wurden (meistens) im ganzen Namen großgeschrieben, in diesem Falle wurden die Grenzen der Namensteile ausnahmsweise durch Unterstriche abgetrennt. Beispiel:

```
const MAX_TPPMS = any integer;
```

Estelle-Schlüsselworte wurden dagegen klein geschrieben.

Bezeichner von Interaktionen wurden ebenso wie Konstanten gebildet, wobei allerdings der Zusatz `req/ind/rsp/cnf` in Kleinbuchstaben nach einem Unterstrich angehängt wurde. Beispiel:

```
TP_BEGIN_TRANSACTION_req
```

Bezeichner von Typen wurden dadurch gekennzeichnet, daß sie auf ...`Type` enden.

Wurde in einer Transition eine Aktion spezifiziert, die im TP-Standard einen Namen trägt, so wurde dieser Name als Kommentar in die Estelle-Spezifikation eingefügt, eingerahmt von +-Zeichen. Beispiel:

```
{+ ADDBRSB +}
```

B.3 Namenskonventionen in der parallelisierten TP-Spezifikation

Es wurden die Namenskonventionen aus der NIST-Spezifikation übernommen, um die Typ- und Kanaldefinitionen dieser Spezifikation direkt weiterverwenden zu können, und da sie sinnvoll erschienen. Dabei wurden lediglich die folgenden Änderungen vorgenommen:

Alle Namen aus dem TP-Standard wurden konsequent übernommen, ohne die beschriebenen Ausnahmen.

Anschließend wurden diese Namen um einen Unterstrich und eine sprechende Langform erweitert, die sich direkt aus der Kurzbeschreibung im Anhang A.2.2.2 und A.4.2.2 des Teils 3 des TP-Standards [ISO92] ergab. (Hinter jedem Namen waren in Klammern einige Stichworte angegeben.) Diese Erweiterung ließ sich sehr einfach mit einer globalen Textersetzung durchführen und erhöhte die Lesbarkeit der Estelle-Spezifikation wesentlich, ohne den engen Bezug zum TP-Standard zu kompromittieren. Beispiel:

```
var
  Nrn_rejectNotAllowed: boolean;
        { reject of superior dialogue not allowed }
```

Bei der Bildung von Langform-Namen wurde die zusätzliche Konvention eingeführt, `for` durch `4` und `to` durch `2` abzukürzen.

Für die Bennenung von Moduldefinitionen und -rümpfen wurde eine konsequentere Notation gewählt. Moduldefinitionsnamen enden immer auf ...`Type` (genau wie Typdefinitionen) und Modulrumpfnamen enden immer auf ...`Body`.

Estelle erlaubt es, zu jeder Transition einen Namen mit dem `name`-Konstrukt anzugeben. Da dies die Bearbeitung mit Simulationswerkzeugen wie etwa Pet/Dingo wesentlich erleichtert, wurden grundsätzlich für alle Transitionen Namen vergeben.

Diese Namen wurden aus der Position in der Zustandsübergangstabelle des TP-Standards abgeleitet. Der erste Teil besteht aus dem Namen der empfangenen Nachricht, in der üblichen Langform-Konstruktion. Nach einem Unterstrich folgt dann der Name des Zustandes. Sofern ein Kästchen der Tabelle in mehrere Unterkästchen unterteilt ist, wird an den ersten Teil des Namens noch ein Unterstrich und eine Zahl angehängt. Beispiel:

```
name tpBeginDialogueReq_1_reject:
```

Sofern ein Kästchen in zwei Estelle-Transitionen aufgeteilt werden mußte, weil ein Modulstrukturmanager-Aufruf dazwischen notwendig wurde, wurde an den gesamten Namen noch ein Unterstrich und eine Zahl angehängt. Beispiel:

```
name tpBeginDialogueReq_1_accept_1:
name tpBeginDialogueReq_1_accept_2:
```

Die Namen von Initialisierungstransitionen sind, da diese nicht den Tabellen entnommen sind, anders strukturiert und beginnen mit `init`... Sie beschreiben dann entweder das Modul, oder, falls es mehrere solche Transitionen in diesem Modul gibt, das Modul und die Bedingung, unter der die Transition schalten kann. Beispiel:

```
name initMacfAfterCrash:
```

Als Sprache für Estelle-Kommentare wurde ebenso wie in der Spezifikation des NIST Englisch gewählt, um die parallelisierte Spezifikation auch international austauschen zu können.

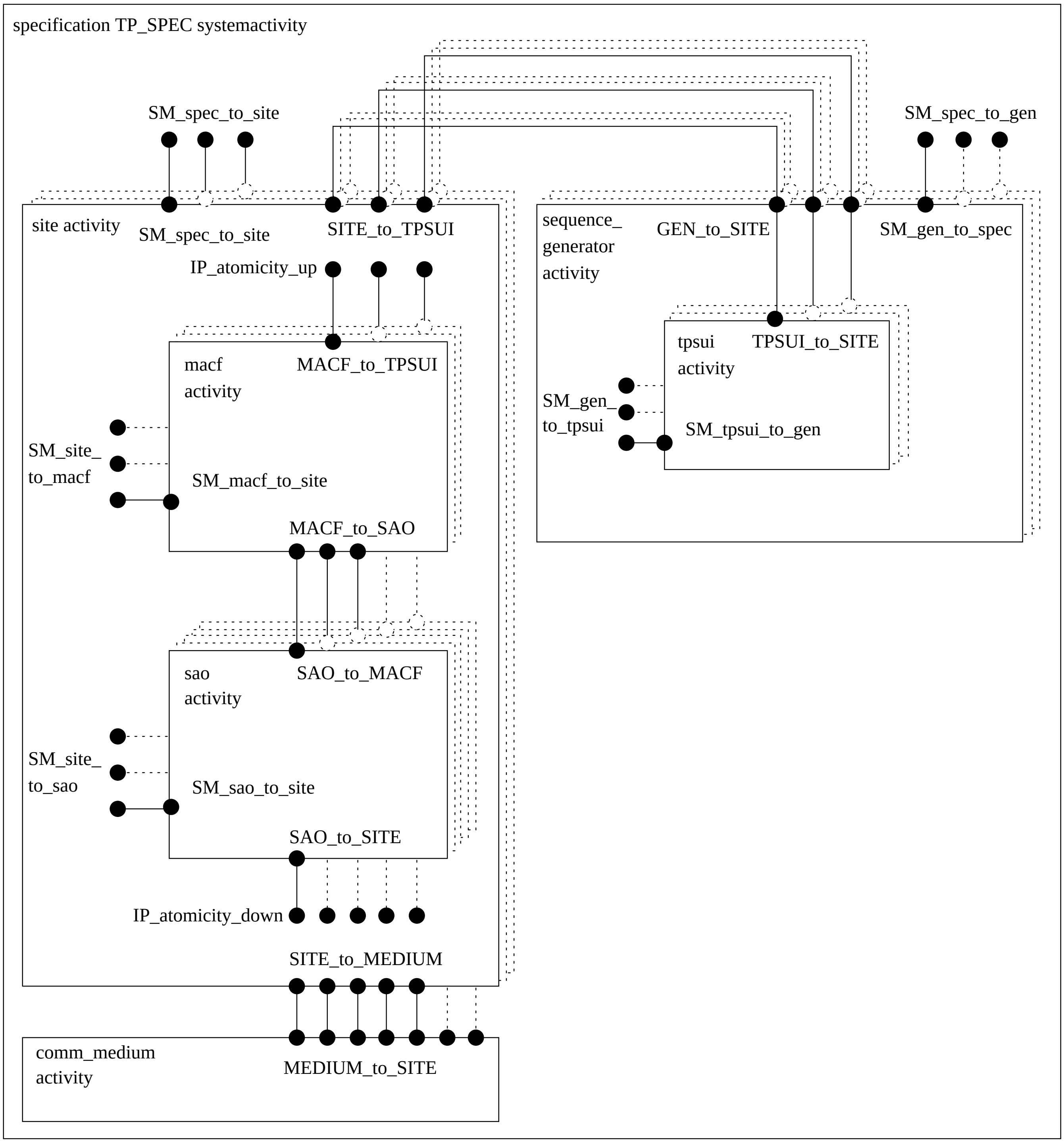
B.4 Namenskonventionen des X-Windows-Systems

Die Namenskonventionen des X-Windows-Systems entsprechen grob den bisher vorgestellten Regeln, insbesondere werden Namen auf die gleiche Weise aus mehreren Teilworten zusammengesetzt. Dies ist recht vorteilhaft für die Anbindung einer mit Pet/Dingo generierten Implementation an eine X-Windows-Fensteroberfläche.

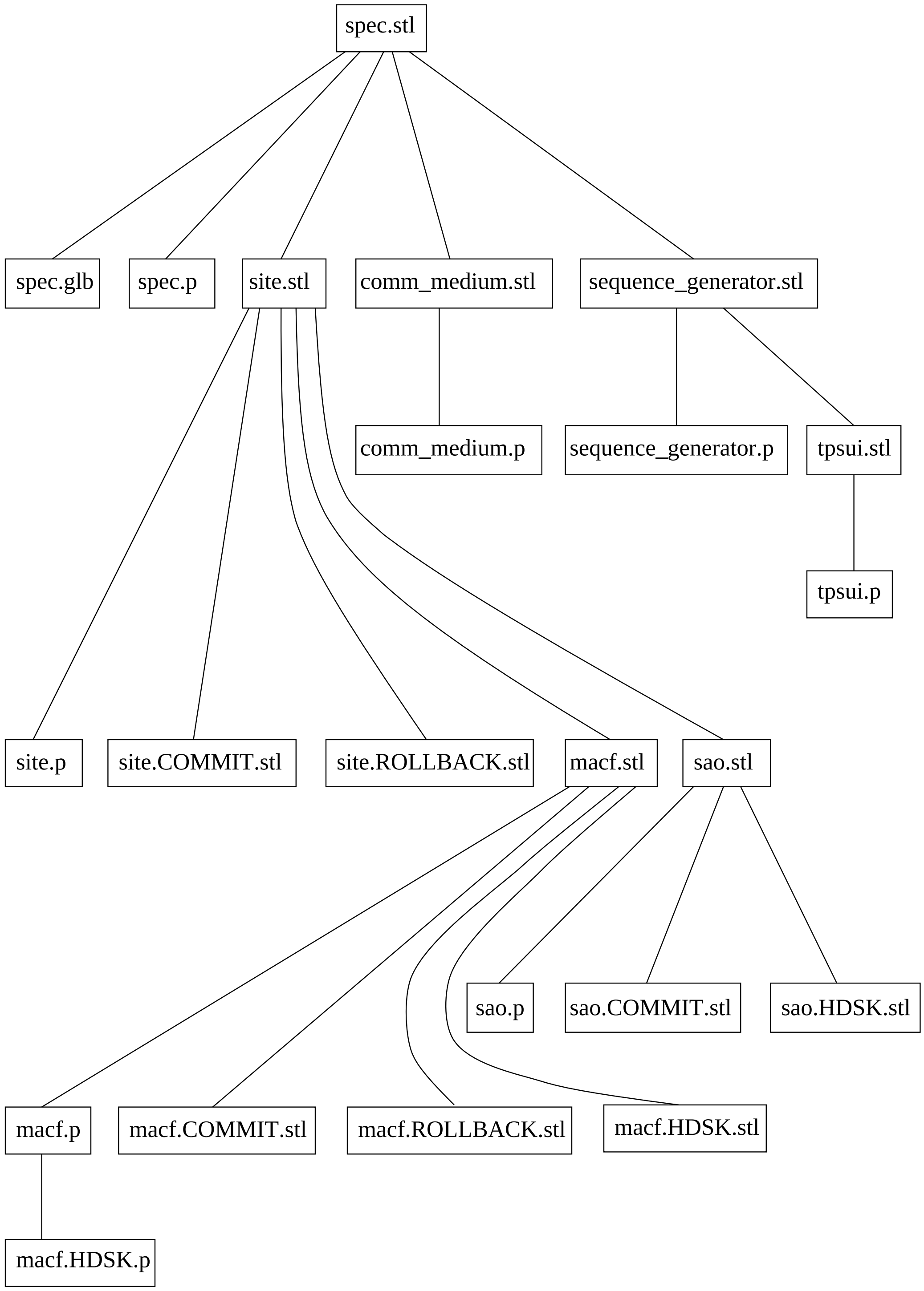
Allerdings werden Konstantennamen auf die gleiche Weise gebildet wie andere Namen, lediglich der erste Buchstabe wird groß geschrieben.

C Die Struktur der TP-Spezifikation von INRIA

C.1 Modulstruktur



C.2 Dateistruktur

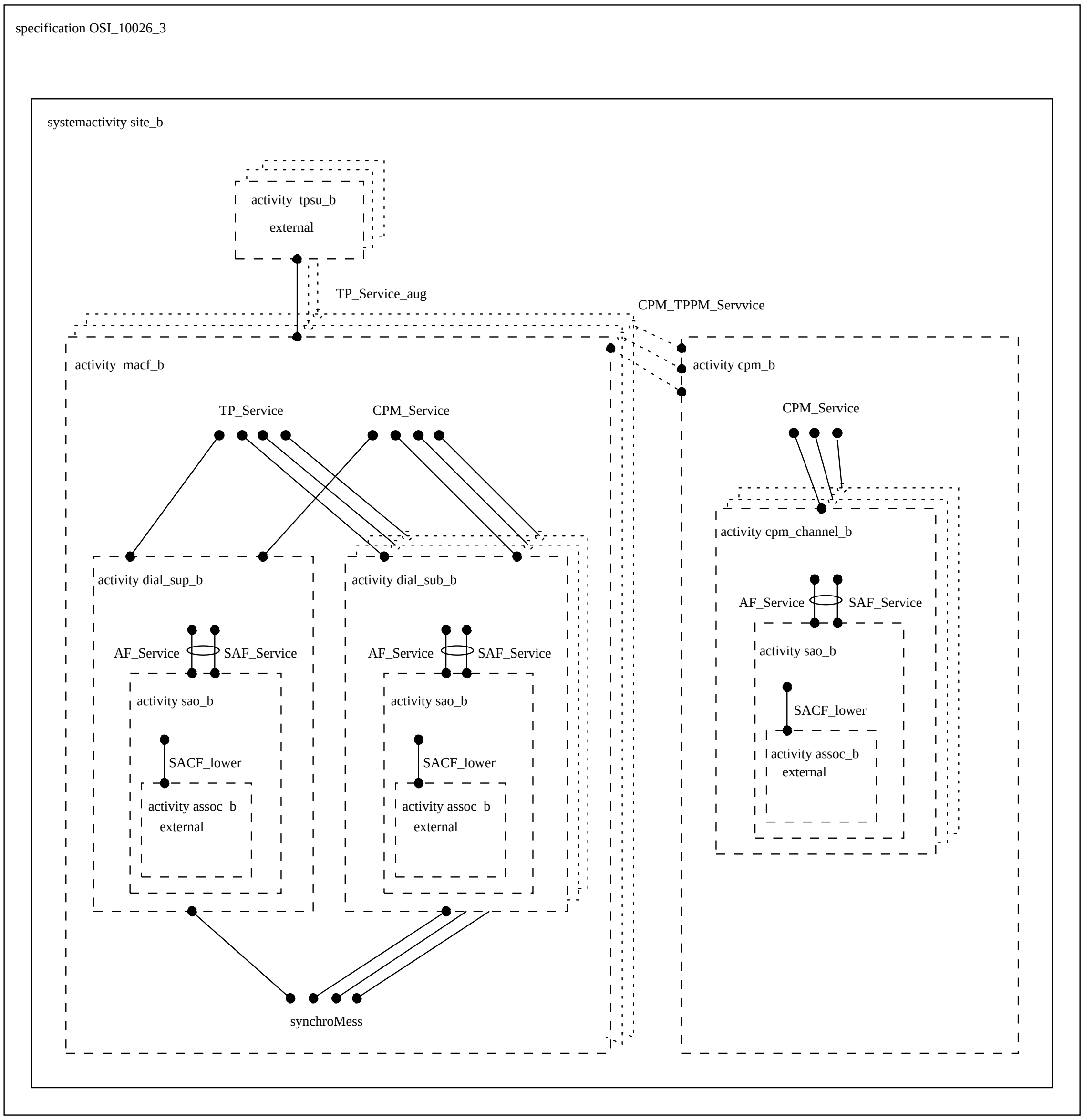


(Erläuterung der Dateiendungen: Siehe nächste Seite)

Datei-Endung	**Bedeutung**
`*.stl`	(Teil einer) Estelle-Spezifikation
`*.p`	Variablen- und Funktionsdefinitionen für ein Modul
`*.glb`	globale Konstanten- Typen- und Kanaldefinitionen
`*.HDSK.*`	auf die Handshake-Funktionseinheit bezogen

D Die Struktur der TP-Spezifikation des NIST

D.1 Modulstruktur



D.2 Dateistruktur