Leistungsbewertung automatisch generierter Protokollimplementierungen mit Estelle — eine Bestandsaufnahme

J. Thees, R. Gotzhein

Nr. 290/97

Leistungsbewertung automatisch generierter Protokollimplementierungen mit Estelle — eine Bestandsaufnahme

J. Thees, R. Gotzhein

Nr. 290/97

Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
D-67653 Kaiserslautern

Email: {thees, gotzhein}@informatik.uni-kl.de

Leistungsbewertung automatisch generierter Protokollimplementierungen mit Estelle — eine Bestandsaufnahme [1]

J. Thees, R. Gotzhein

Universität Kaiserslautern, Postfach 3049, D-67653 Kaiserslautern

Email: {thees, gotzhein}@informatik.uni-kl.de

Zusammenfassung

Formale Beschreibungstechniken (FDTs) erlauben durch ihre formale Syntax und Semantik eine präzise Systembeschreibung und sind Grundlage für die formale Verifikation. Bei der Implementierung von Systemen wird jedoch nach wie vor von Hand implementiert, selbst wenn ausgereifte Werkzeuge zur automatischen Generierung von Kode direkt aus der formalen Spezifikation existieren. Die Ursache dafür liegt in dem Ruf dieser Werkzeuge, Kode mit extrem geringer Leistungsfähigkeit zu erzeugen. Es gibt jedoch kaum quantitative Leistungsvergleiche zwischen manuell und automatisch generierten Implementierungen, die dieses Vorurteil stützen oder widerlegen könnten.

In diesem Beitrag wird ein solcher Leistungsvergleich anhand des Hochleistungsprotokolls XTP und der FDT Estelle vorgestellt. Er liefert eine *Bestandsaufnahme* des momentanen Entwicklungsstandes bei der automatischen Generierung von Kode aus Estelle-Spezifikationen im direkten Vergleich zu gut optimierten Handimplementierungen. Es zeigt sich, daß in dem betrachteten Fall eines komplexen Protokolls die Handimplementierung zwar merklich leistungsstärker ist. Dieser Leistungsvorteil wird jedoch durch einen sehr hohen Implementierungsaufwand sowie die Schwierigkeit, die Korrektheit bzgl. der Spezifikation sicherzustellen, erkauft. Im einzelnen Anwendungsfall kann es daher trotz der Leistungseinbußen durchaus vorteilhaft sein, automatisch Kode zu erzeugen, zumal in der Bestandsaufnahme festgestellt wurde, daß automatisch generierte Implementierungen z.T. besser abschneiden als erwartet. Zudem besteht — anders als bei der bereits umfassend optimierten Handimplementierung — noch ein erhebliches ungenutztes Potential zur Leistungsverbesserung der automatisch generierten Implementierung.

Keywords

Leistungsbewertung, Monitorsysteme, Kommunikationssysteme, Formale Beschreibungstechniken (FDTs), Estelle, automatische Implementierung, Hochleistungsprotokolle, XTP, SandiaXTP

1 Einleitung

Die Hauptanforderungen an Protokollimplementierungen sind *Korrektheit* und *Effizienz*. Der Aspekt der Korrektheit legt den Einsatz von formalen Beschreibungstechniken (FDTs) nahe, wohingegen zur Erreichung hoher Effizienz nach wie vor handkodierte Implementierungen zum Einsatz kommen. Dieses Vorgehen stellt jedoch speziell bei großen Systemen die Übertragbarkeit der Eigenschaften der formalen Beschreibung auf die Implementierung in Frage.

Der Einsatz von Werkzeugen zur *automatischen Generierung von Protokollimplementierungen* direkt aus formalen Beschreibungen heraus könnte zur Lösung dieses Problems beitragen. Jedoch werden diese Werkzeuge in der Praxis wegen ihres Rufs, Kode mit extrem geringer Leistungsfähigkeit zu erzeugen, nur sehr selten eingesetzt. Dabei beruht dieser Ruf im

1. Diese Arbeit wurde von der Deutschen Forschungsgemeinschaft (DFG) im Rahmen des Projekts “Protokollimplementierung mit Estelle — Vom Prototypen zum effizientem Kode” (Go 503/4-1) gefördert.

wesentlichen auf dem Vorurteil, daß FDTs zu wenig implementierungsbezogen seien und die daraus automatisch generierten Implementierungen sich nicht mit handkodierten Implementierungen messen könnten. Jedoch gibt es kaum direkte quantitative Leistungsvergleiche zwischen handkodierten Protokollmaschinen und Implementierungen, die automatisch aus formalen Spezifikationen erstellt wurden.

In diesem Beitrag stellen wir einen solchen quantitativen Leistungsvergleich und die dazu entwickelten Werkzeuge anhand des Hochleistungsprotokolls "*Xpress Transport Protocol*" (XTP) und der FDT *Estelle* vor. Diese Untersuchung liefert erste quantitative Leistungsdaten über den aktuellen Stand der Entwicklung im Bereich der automatischen Protokollimplementierung im Vergleich mit einer gut optimierten Handimplementierung. Derartige Daten sind unerläßlich bei der Entscheidung, ob die automatisch generierte Implementierung eines Protokolls genügend Leistung für eine konkrete Anwendung erbringt, oder ob es notwendig und sinnvoll ist, eine Handimplementierung zu entwickeln. Diese Fragestellung hat besondere Bedeutung beim Einsatz von anwendungsspezifischen Protokollen, bei denen die Entwicklung einer gut optimierten Handimplementierung möglicherweise unrentabel ist.

In Abschnitt 2 geben wir zunächst eine kurze Einführung in die FDT Estelle und einen Überblick über die in der nachfolgenden Untersuchung eingesetzten Compiler. Abschnitt 3 behandelt XTP, die dem Vergleich zugrundeliegende Estelle-Spezifikation von XTP und die Handimplementierung "SandiaXTP". In Abschnitt 4 werden die Methoden, Werkzeuge und Ergebnisse des Leistungsvergleichs vorgestellt. In Abschnitt 5 wird schließlich ein Ausblick auf laufende Forschungsaktivitäten gegeben.

2 Die FDT Estelle

Estelle [ISO89, DeBu89] ist eine formale Beschreibungstechnik ("formal description technique", FDT), die seit 1989 international genormt ist. Sie basiert auf ISO-Pascal und wurde zur Beschreibung von *Kommunikationsprotokollen* entwickelt. Estelle-Spezifikationen sind formal in dem Sinne, daß sie eine *formale Syntax* und eine *formale Semantik* besitzen. Dies erlaubt eine präzise Beschreibung des spezifizierten Systems und ist eine Voraussetzung für den Nachweis funktionaler und nicht-funktionaler Eigenschaften.

2.1 Die Struktur von Estelle-Spezifikationen

Eine Estelle-Spezifikation beschreibt ein dynamisch modifizierbares hierarchisches System von miteinander *kommunizierenden erweiterten endlichen Automaten* (EFSMs), den *Modulinstanzen* (siehe Abbildung 1). Die Erzeugung und Freigabe von Modulinstanzen wird lokal durch die jeweilige Vatermodulinstanz kontrolliert. Die oberste Modulinstanz (die Instanz des Spezifikationsmoduls) wird dabei implizit zu Beginn der Ausführung der Spezifikation erzeugt.

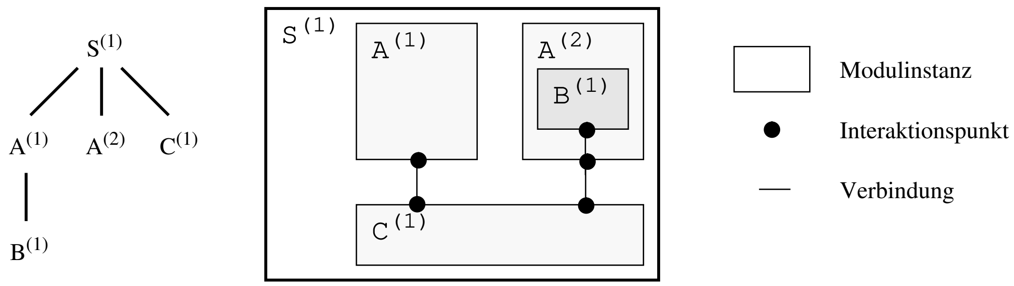

Abbildung 1: Dynamische Modulinstanz- und Verbindungshierarchie

Module haben eine wohldefinierte externe Schnittstelle (beschrieben durch den *Modulheader*) und ein privates Innenleben (beschrieben durch den *Modulrumpf*).[2] Die Kommunikation zwischen den Modulinstanzen basiert auf zwei orthogonalen Konzepten: *Asynchrone Nachrichtenübertragung* und *Gemeinsame Variablen*. Entsprechend besteht die äußere Schnittstelle eines Moduls aus einer Menge von *externen Interaktionspunkten* und *exportierten Variablen*. Die externen Interaktionspunkte dienen dabei als Schnittstelle für den asynchronen Austausch parametrisierter Nachrichten (*Interaktionen*), und die exportierten Variablen dienen als gemeinsame Variablen der Modulinstanz und ihrer Vatermodulinstanz.

Die dynamische Modulstruktur wird ergänzt durch eine *dynamische Verbindungsstruktur*, die den Zielpunkt der über einen Interaktionspunkt verschickten Nachrichten bestimmt. Zwischen zwei verbundenen Interaktionspunkten besteht ein bidirektionaler Kommunikationskanal. Nachrichten, die über einen verbundenen Interaktionspunkt geschickt werden, werden am anderen Endpunkt der Verbindung in einen FIFO-Puffer unbegrenzter Länge abgelegt, aus dem sie später (über den Empfängerinteraktionspunkt) in der Reihenfolge ihres Eintreffens entnommen werden können.

2.2 Zustand und Verhalten von Modulinstanzen

Jede Modulinstanz besitzt einen *lokalen Zustand*, der (unter anderem) den *Kontrollzustand* des Automaten und die Inhalte der lokalen Variablen umfaßt. Das *Verhalten* der Modulinstanz wird durch eine Menge von *Transitionen* beschrieben. Jede Transition beschreibt eine Schaltbedingung und eine Schaltwirkung. Die Schaltbedingung legt fest, unter welchen (im wesentlichen lokalen) Bedingungen die jeweilige Transition ausgeführt ("gefeuert") werden darf. Die Schaltwirkung beschreibt die (lokalen und globalen) Auswirkungen der Ausführung der Transition. Das Schalten einer Transition erfolgt dabei atomar.

Syntaktisch wird eine Transition durch eine Menge von Transitionsklauseln und einen Transitionsblock repräsentiert. *Transitionsklauseln* beschreiben Schaltbedingungen und/oder Schaltwirkungen. Der *Transitionsblock* besteht im wesentlichen aus einem Pascal-ähnlichen Anweisungsblock, der beim Feuern der Transition ausgeführt wird.

Die *Auswahl* der zu schaltenden Transitionen aus der Menge der schaltbaren geschieht zum einen durch eine modullokale Transitionsauswahl und zum anderen durch eine globale Auswahl zwischen den Modulinstanzen. Die *modullokale Transitionsauswahl* bestimmt aus der Menge der schaltbaren Transitionen eine Transition mit maximaler Priorität und bietet diese zur Ausführung an. Falls es mehrere mögliche Transitionen gibt, erfolgt die Auswahl indeterministisch. Die *globale Auswahl* zwischen den angebotenen Transitionen der Modulinstanzen wird durch die Attributierung der Modulinstanzen gesteuert und erfolgt z.T. ebenfalls indeterministisch.

Aufgrund der Möglichkeit zu indeterministischem Verhalten wird die Semantik einer Estelle-Spezifikation durch eine "next-state"-Relation beschrieben, die alle zulässigen Zustandsübergänge beschreibt. Daraus läßt sich (ausgehend vom initialen Zustand des Systems) die Menge der erlaubten Ereignis- und Zustandsfolgen ableiten.

2.3 Anbindung an eine Umgebung

Estelle-Spezifikationen beschreiben (topologisch) geschlossene Systeme, d.h. Systeme ohne Schnittstelle zu ihrer Umgebung. Es gibt zwar erste Ansätze zu einer Erweiterung von Estelle hin zu "Open Estelle" [GRT96], jedoch werden sie von den gängigen Compilern noch nicht unterstützt. Somit ist die Ankopplung von in Estelle spezifizierten Systemen an reale Umgebun-

2. Eine Ausnahme bildet das Spezifikationsmodul, das keinen explizit definierten Modulheader besitzt; Estelle-Spezifikationen beschreiben geschlossene Systeme und haben somit keine äußere Schnittstelle.

gen (z.B. an einen IP-Service unter UNIX) vorerst nur durch pragmatische Mittel auf Implementierungsebene möglich. Das in Estelle dazu bereits vorgesehene Mittel sind *primitive Funktionen* bzw. *Prozeduren*. Diese erlauben den Zugriff auf Dienste, die nicht Bestandteil der Estelle-Spezifikation sind, machen jedoch die Existenz einer formalen Semantik vom Vorhandensein einer formalen Beschreibung dieser Dienste abhängig.

2.4 Estelle Kodegeneratoren

Estelle eignet sich durch seine operationale Verhaltensbeschreibung und seine einfache Typstruktur besonders für eine *automatische Implementierung*. Im Gegensatz zu anderen FDTs, die eigenschaftsorientierte Verhaltensbeschreibungen oder abstrakte, rekursive und axiomatische Typdefinitionen benutzen, kann Estelle relativ direkt implementiert werden: Estelle-Typen sind im wesentlichen Pascal-Typen und haben somit eine feste Struktur. Sie können direkt in eine Implementierungssprache wie C abgebildet werden. Gleiches gilt für die Verhaltensbeschreibung von Estelle-Spezifikationen, die im wesentlichen durch Transitionen mit sequentiellen, imperativen, Pascal-ähnlichen Anweisungsblöcken gegeben werden. Auch sie können direkt nach C abgebildet werden.

Für Estelle wurden bereits mehrere Compiler mit unterschiedlichen Zielsetzungen entwickelt. Allen gemeinsam ist, daß sie aus einer Estelle-Spezifikation Pascal-, C- oder C++-Quellkode generieren, der dann zu einem ablauffähigen Programm übersetzt werden kann. Im folgenden werden die gebräuchlichsten Compiler kurz vorgestellt:

- Der **NIST-Estelle-Compiler** war der erste Estelle-Compiler und wurde vom amerikanischen National Institute for Standards and Technologies (NIST) entwickelt [SiSt90]. Er ist nicht mehr verfügbar und wurde von PET/DINGO (s.u.) abgelöst.
- Der **UHH-Estelle-Compiler** wurde an der Universität Hamburg entwickelt [KrGo93]. Er basiert auf dem oben erwähnten NIST-Compiler und erzeugt wie dieser C-Kode. Der UHH-Compiler erlaubt die Erzeugung einer auf mehrere Rechnerknoten verteilt ausführbaren Implementierung einer Spezifikation. Einer seiner Schwachpunkte ist die fehlende Unterstützung von exportierten Variablen. Er konnte deshalb in den XTP-Leistungsvergleich nicht miteinbezogen werden, ist jedoch beim Estelle-Benchmark (Abschnitt 4.2) vertreten.
- Das Kodegenerierungswerkzeug **PET/DINGO** besteht aus dem Frontend PET (Portable Estelle Translator) und dem Backend DINGO (Distributed Implementaion Generator). Er wurde vom NIST [SiSt93] als Nachfolger des o.g. NIST-Estelle-Compilers entwickelt und unterstützt ebenfalls die Erzeugung einer auf mehrere Rechnerknoten verteilt ausführbaren Implementierung einer Spezifikation. Da PET/DINGO objektorientiert (in C++) implementiert wurde und auch der generierte Kode auf einer C++-Klassenbibliothek aufsetzt, eignet sich PET/DINGO gut für die Einbettung von Werkzeugen zum Performance-Monitoring. Wir werden in Abschnitt 4.1.3 das Werkzeug PATO vorstellen, das zu diesem Zweck speziell für PET/DINGO entwickelt wurde.
- Bei **EC** (Estelle Compiler) handelt es sich im Gegensatz zu den übrigen Werkzeugen um einen *kommerziellen* Estelle-Compiler, der von Bull S.A. entwickelt [RiCl89] und später vom französischen Institut National des Télécommunications (INT) übernommen wurde. EC ist Teil des Estelle-Entwicklungssystems EDT (Estelle Development Toolset), das unter anderem auch einen Estelle-Debugger (EDB) enthält. EC wurde mit dem Ziel entwickelt, eine *optimierte Implementierung* einer Estelle-Spezifikation zu erzeugen. Es wird sich zeigen, daß EC diesen Anspruch im Vergleich zu den anderen Compilern auch erfüllen kann. Da es sich jedoch um ein kommerzielles Produkt handelt, sind die Quellen des Compilers und der Laufzeitbibliothek nicht zugänglich, was die Möglichkeiten zur detaillierten Leistungsanalyse stark einschränkt.

3 Das 'Xpress Transport Protocol'

Als Grundlage für den quantitativen Leistungsvergleich zwischen automatisch generierten Implementierungen und Handimplementierungen von Kommunikationsprotokollen haben wir das 'Xpress Transport Protocol' (XTP, [XTP95]) Version 4.0 gewählt: Zum einen sind für dieses Protokoll sowohl eine gut optimierte Handimplementierung (Abschnitt 3.1) als auch eine formale Estelle-Spezifikation (Abschnitt 3.2) verfügbar, zum anderen ist es als Vertreter der Klasse der Hochleistungsprotokolle für eine effiziente Implementierung besonders geeignet.

XTP 4.0 ist — im Gegensatz zu früheren XTP-Versionen — ein reines Transportschichtprotokoll.[3] Die Zielsetzung bei der Entwicklung war es, durch eine vollständige Neuentwicklung die Schwächen der etablierten Transportschichtprotokolle (wie TCP und TP4) zu überwinden und eine bessere Anpassung an die Eigenschaften moderner Kommunikationsmedien mit ihren hohen Bandbreiten und geringen Fehlerraten zu erreichen.

XTP ist ein sehr flexibles Protokoll, denn es erlaubt den jeweiligen Anwendungen eine umfassende Kontrolle über die genauen Eigenschaften der Datenübertragung. So können verschiedene Kommunikationsparadigmen (Datagramm, Transaktion, Verbindung) völlig unabhängig von der Fehlerkontrolle (völlig zuverlässig bis unkorrigiert) gewählt und sogar bei einer schon bestehenden Verbindung noch geändert werden. Sogar die Unterstützung zuverlässiger Multicasts, die ein integraler Bestandteil von XTP ist, kann zusammen mit (und unabhängig von) den übrigen Mechanismen eingesetzt werden. Ebenso erlaubt XTP eine getrennte Raten- und Flußkontrolle und macht so ineffiziente Heuristiken, wie etwa den "slow-start" von TCP, überflüssig. XTP 4.0 ist dabei nicht an einen speziellen Basisdienst gebunden, sondern kann neben typischen Vermittlungsschicht-Diensten (z.B. IP) auch direkt auf einfache LAN-Dienste (z.B. Ethernet IEEE 802.3) aufgesetzt werden.

Im Sinne eines Hochleistungsprotokolls war neben großer Flexibilität auch eine hohe Performance unter allen Betriebsbedingungen ein Ziel beim Design von XTP. Dazu wurden zum einen Protokollmechanismen eingesetzt, die es erlauben, die Anzahl der Synchronisationen zwischen Partnerinstanzen zu minimieren, um dadurch unnötige Wartezeiten zu vermeiden. So erlaubt z.B. der Einsatz eines *impliziten Verbindungsaufbaus* die Einsparung einer kompletten "round trip delay time". Dies führt speziell im Transaktionsbetrieb zu enormen Leistungssteigerungen gegenüber einem regulären Verbindungsaufbau.

Zum anderen wurde eine Leistungssteigerung bei der Verarbeitung von Paketen in der Protokollmaschine durch verschiedene Maßnahmen ermöglicht. Z.B. haben die Paketheader ein festes Format und erlauben daher eine schnelle Dekodierung. Die einzelnen Felder im Header liegen auf optimal ausgerichteten Offsets (8-Byte-Alignment für 8-Byte Strukturen usw.) und die einzelnen Komponenten eines Paktes (Segmente) haben eine auf 8-Byte-Vielfache aufgerundete Größe, so daß die in modernen Hardwarearchitekturen vorhandenen Optimierungen greifen können. Weiterhin erlaubt das Protokoll den Austausch von sogenannten "return keys", durch die der Empfänger eines Paket dieses ohne Suchoperationen einer Verbindung zuordnen kann. Diese und viele weitere Maßnahmen erlauben eine sehr effiziente Implementierung von XTP.

3.1 SandiaXTP

SandiaXTP 1.4 [SNL95a, SNL95b] ist eine optimierte, handcodierte Implementierung von XTP 4.0, die von den Sandia National Laboratories entwickelt wurde. Die XTP Protokollma-

3. Die früheren XTP Versionen umfaßten auch Funktionalitäten der Vermittlungsschicht; entsprechend bedeutete XTP damals 'Xpress Trans<u>fer</u> Protocol'.

schine ist hier als UNIX-Dämon-Prozeß realisiert, der u.a. über UDP oder IP eine Kommunikation zwischen verschiedenen Hostrechnern erlaubt. Die Anwendungen von XTP arbeiten als getrennte Prozesse, die über Sockets (für Synchronisation und Austausch von Kontrollinformationen) und gemeinsamen Speicher ("shared memory" für die Übertragung der Nutzdaten) mit dem XTP-Dämon am selben Host kommunizieren (siehe Abbildung 2). Die Ankopplung an den Basisdienst erfolgt im Falle von IP und UDP vollständig über Sockets.

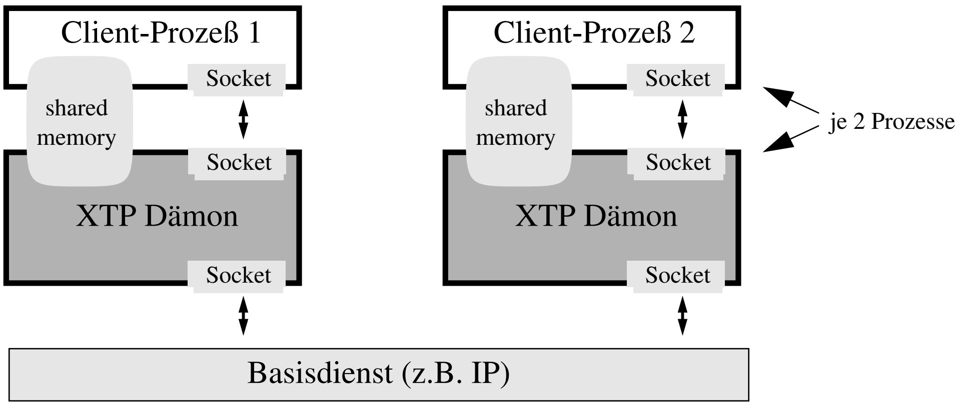

Abbildung 2: Die Kommunikationsstruktur von SandiaXTP

SandiaXTP hat eine durchgehend objektorientierte Struktur und ist in C++ implementiert. Durch effiziente Algorithmen und den gezielten Einsatz von Inline-Kode konnte eine hohe Effizienz erreicht werden. So wird z.B. der Nutzdatenanteil eines Sendeauftrags nur zweimal kopiert: Beim Sendeaufruf eines Clients werden die Daten in einen Shared-Memory Puffer kopiert, vom XTP Dämon ohne weiteres Kopieren bearbeitet und schließlich direkt aus diesem Puffer als IP-Sendeauftrag an das Betriebssystem übergeben.

3.2 Eine Estelle-Spezifikation von XTP

Die der Untersuchung zugrundeliegende Estelle-Spezifikation wurde ursprünglich am Institut National des Télécommunications für XTP 3.6 erstellt [Bud93] und später an der PUB Bukarest an XTP 4.0 angepaßt. Da Estelle nur geschlossene Systeme beschreiben kann, wurde der XTP Protokollautomat in ein vollständiges Umgebungsszenario eingebettet. Dieses enthält zusätzlich ein Netzwerkmodul und zwei Benutzermodule. Die Grobstruktur dies Kommunikationsszenarios ist in Abbildung 3 dargestellt.

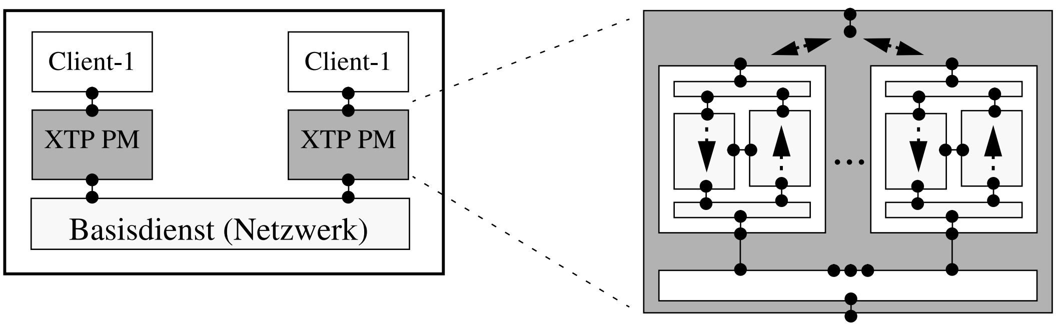

Abbildung 3: Die Modulinstanz- und Verbindungsstruktur der Estelle-Spezifikation

Ziel bei der Erstellung der Spezifikation waren im wesentlichen die Entwicklung einer geeigneten Dienstschnittstelle und die Erprobung der Protokollfunktionalität. Performanceaspekte wurden dabei nicht berücksichtigt. Dies macht sich besonders in der starken Strukturierung des Protokollautomaten in eine komplexe Modulhierarchie bemerkbar: Die sich daraus ergebende Notwendigkeit zur häufigen Übertragung der Nutzdaten zwischen Modulen führt zu hohem Kopieraufwand. So wird ein Datenpaket auf seinem Weg vom Benutzermodul zum Netzwerk insgesamt siebenmal (!) kopiert, ebenso auf dem Weg vom Netzwerk zum Benutzermodul. Außerdem macht die Modularisierung zusammen mit der eingeschränkten Möglichkeit zu gemeinsamem Speicher in Estelle zusätzliche interne nachrichtenbasierte Kommunikation zwischen den Komponenten einer Protokollmaschine nötig.

Ein weiterer Unterschied zur Handimplementierung besteht darin, daß die XTP Protokoll-Dateneinheiten (PDUs) in der Estelle-Spezifikation ohne Berücksichtigung der exakten Binärrepräsentation des XTP-Protokolls spezifiziert sind, da Estelle keine angemessenen Mittel zur Spezifikation von Aspekten wie *Byte-Ordering* oder *Prüfsummenberechnung* besitzt. Für den Leistungsvergleich mit SandiaXTP wurden diese Funktionalitäten durch entsprechende primitive Kodierungsfunktionen (außerhalb der Estelle-Spezifikation) ergänzt (siehe Abschnitt 4.1). Zur Prüfsummenberechnung wurden die Funktionen von SandiaXTP eingesetzt.

4 Leistungsmessung

In diesem Abschnitt werden die Methoden und Werkzeuge dargestellt, die beim quantitativen Leistungsvergleich zwischen den verschiedenen Protokollimplementierungen zum Einsatz gekommen sind. Der Vergleich der Estelle-Implementierungen mit der Handimplementierung (Abschnitt 4.1) machte eine Aufspaltung der Estelle-Spezifikation in zwei (pragmatisch realisierte) kommunizierende offene Systeme notwendig. Eine zweite Untersuchung (Abschnitt 4.2) vergleicht die verschiedenen Estelle-Compiler anhand von Benchmark-Spezifikationen. Weitere Details zum Meßverfahren und zu den gewonnenen Ergebnisse sind in [Dah96] zu finden.

4.1 Vergleich zwischen SandiaXTP und automatisch generierten Estelle-Implementierungen

Zu einem fairen Leistungsvergleich zwischen den XTP Implementierungen waren neben einigen Anpassungen am realisierten Protokollumfang (z.B. Prüfsummenberechnung) auch einige strukturelle Änderungen notwendig. So kann das geschlossene Kommunikationsszenario der Estelle-Spezifikation vom Estelle-Compiler EC nur in einen einzelnen UNIX-Prozeß umgesetzt werden, wohingegen SandiaXTP ein vergleichbares Szenario nur zwischen mehreren Dämon-Prozessen auf unterschiedlichen Hostrechnern darstellen kann. Aber auch die Möglichkeiten zur Erzeugung einer verteilten Implementierung der Estelle-Spezifikation durch PET/DINGO oder UHH führen zu keiner mit SandiaXTP vergleichbaren Implementierung, da die Kommunikation zwischen den verteilten Estelle-Komponenten nicht auf die Protokollkommunikation von XTP beschränkt werden kann, sondern ganz überwiegend aus Estelle-spezifischen Synchronisationsnachrichten besteht.

Um dennoch ein realistisches Kommunikationsszenario nachbilden zu können, wurde die Estelle-Spezifikation in zwei Spezifikationen zerteilt, die jeweils ein Benutzermodul, eine Protokollmaschine und ein Netzwerkmodul enthalten. Mit Hilfe von primitiven Funktionen [GRT96] wurden dann die Netzwerkmodule beider Spezifikationen so modifiziert, daß diese Module als Zugang zu realen Kommunikationsmedien (z.B. IP) dienen konnten (siehe Abbildung 4), wobei durch geeignete Kodierungs- und Dekodierungsfunktionen die implementierungsspezifischen Estelle-Datenstrukturen in binärkompatible XTP Pakete umgewandelt wurden. Dadurch konnten mit den zur Verfügung stehenden Estelle-Compilern jeweils zwei Prozesse erzeugt werden, die — genau wie SandiaXTP — über ein reines XTP-Protokoll kommunizierten. Es war dadurch sogar möglich, eine direkte Kommunikation zwischen SandiaXTP und den aus Estelle generierten Prozessen durchzuführen.

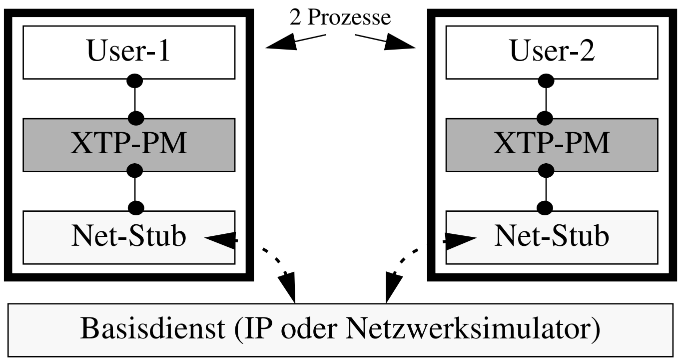

Abbildung 4: Kommunikation der modifizierten Estelle-Spezifikation

4.1.1 Netzwerksimulator

Als Basisdienst für den Vergleich zwischen SandiaXTP und den Estelle-Implementierungen wurde ein eigens dazu entwickelter *Netzwerksimulator* eingesetzt. Dieser erlaubt es, die Eigenschaften des Basisdienstes (Durchsatz, Verzögerung, Zuverlässigkeit) exakt zu kontrollieren, und gewährleistet somit die Reproduzierbarkeit der Meßergebnisse. Durch den Einsatz von Shared-Memory als Datentransportmedium kann der Aufwand für die Übertragung eines Pakets vom Sender- zum Empfängerprozeß auf eine Kopieroperation beschränkt werden. Dies erlaubt höchste Datendurchsatzraten und ermöglicht so die realistische Simulation eines Hochgeschwindigkeitsmediums. Da mit Hilfe des Netzwerksimulators alle Komponenten des Kommunikationssystems auf demselben Hostrechner arbeiten, existiert auch eine ausreichend genaue globale Zeitbasis, welche die zeitliche Verfolgung der Verarbeitung eines Pakets über die Grenzen einer Protokollmaschine hinweg erlaubt.

Die Ankopplung der XTP-Protokollautomaten erfolgte bei den automatisch generierten Implementierungen durch primitive Funktionen und Prozeduren und bei SandiaXTP durch eine entsprechende Anpassung an diesen neuen Basisdienst. Die Datenübertragung erfolgt bei allen Implementierungen über Shared Memory, die Signalisierung über UNIX-Signals. Alle Automaten erzeugen binäre XTP Protokollpakete. Am SandiaXTP Dämon waren zusätzliche Modifikationen notwendig, damit zwei seiner Instanzen gleichzeitig auf einem Hostrechner arbeiten können (virtuelle Host-IDs, parametrierbare Zuordnung der Clients an die Dämons).

4.1.2 Benutzerprofile

Aufgrund der große Flexibilität von XTP hat das Anwendungsprofil der Benutzermodule bzw. -prozesse großen Einfluß auf die Kommunikation zwischen den Protokollmaschinen und damit auch auf den Datendurchsatz des Gesamtsystems. Um trotz der unterschiedlichen Schnittstellen von SandiaXTP und der in Estelle spezifizierten Protokollautomaten mehrere vergleichbare Benutzerprofile dynamisch einsetzen zu können, wurden für beide Protokollrealisierungen dynamisch parametrierbare, generische Benutzermodule bzw. -prozesse erstellt. Diese erlauben die Simulation verschiedener Übertragungscharakteristika, wie z.B. “Nachrichten Ping-Pong” oder die Übertragung großer Datenmengen (“Bursts”). Die Einstellung des Benutzerprofils ist interaktiv oder über Parameterdateien möglich, so daß leicht mit verschiedenen Einstellungen experimentiert werden kann.

Eine Ausnahme bei der dynamischen Auswahl von Benutzerprofilen ist die maximale Datenpaketgröße bei der Estelle-Spezifikation: Da Estelle keine Datenpakete (Arrays) variabler Größe beschreiben kann, muß bei der Erstellung einer Implementierung eine konkrete Obergrenze für die Nutzdatenmenge eine Paketes (sowohl an der Benutzerschnittstelle als auch an der Netzwerkschnittstelle) angegeben werden. Bei einer Änderung dieser Größe muß die Implementierung neu erzeugt werden. Wie sich zeigen wird, hängt die Leistung der aus der Estelle-Spezifikation erzeugten Implementierungen neben der zu übertragenden Datenmenge auch ganz wesentlich von dieser maximalen Nutzdatenmenge der Pakete ab, da bei der Übertragung solcher Pakete durch Interaktionspunkte immer die gesamte Datenstruktur des Pakets kopiert wird. Deshalb wurde bei den Leistungsmessungen sowohl die *aktuelle* als auch die *maximale* Nutzdatenmenge (“Limit”) variiert.

4.1.3 Das Performance-Monitoring-Werkzeug PATO

Zur detaillierten Leistungsmessung in Protokollimplementierungen wurde das Performance Monitoring- und Analysewerkzeug PATO (“Performance Analysing Tool”, [Pen96]) eingesetzt, das im Rahmen einer Diplomarbeit u.a. zur Untersuchung von automatisch generierten Implementierungen von Estelle-Spezifikationen entwickelt wurde. PATO besteht aus einer Bibliothek von Funktionen, die in dem zu überwachenden Programm das Setzen von Zeitstempeln erlauben, und einem Auswertungswerkzeug. Die Aufrufe zur Erzeugung der Zeitstempel werden entweder manuell (bei SandiaXTP, UHH und EC) oder automatisch (durch eine entsprechend modifizierte Version von DINGO) eingefügt.

Speziell zusammen mit DINGO erlaubt PATO eine sehr selektive und spezialisierte Untersuchung von wiederkehrenden oder auch einmaligen Protokollabläufen. So kann z.B. der zeitliche Ablauf der Verarbeitung eines einzelnen Pakets durch die Modulhierarchie genauso verfolgt werden wie die Untersuchung der Auswahl- und Ausführungszeit einzelner Transitionen oder Modulinstanzen. Durch die starke Selektivität der Meßpunkte ist dabei eine Minimierung der durch die Messung verursachten Fehler möglich.

Die Kompensation von Meßfehlern ist auch eine wesentliche Aufgabe des grafischen Auswertungswerkzeugs von PATO. So erlaubt die grafische Darstellung der gewonnenen Daten das benutzergesteuerte Ausfiltern von Störungen, wie sie z.B. durch die Unterbrechung des zu untersuchenden Prozesses durch das Betriebssystem verursacht werden können.

4.1.4 Meßergebnisse

Am Beispiel von SandiaXTP und den von den verschiedenen Estelle-Compilern erzeugten Implementierungen der Estelle-XTP-Spezifikation wurde eine systematische vergleichende Leistungsmessung durchgeführt. Dabei wurden unter Variation verschiedener Parameter, wie Paketgröße, Eigenschaften des Basisdienstes und Benutzerprofil, quantitative Messreihen[4] durchgeführt. In diesem Abschnitt werden beispielhaft die Ergebnisse des Leistungsvergleichs anhand des Benutzerprofils “*Nachrichten-Ping-Pong*” unter Variation der Paketgröße zusammenfassend dargestellt. Dabei überträgt ein XTP Anwender über eine bestehende Verbindung ein Datenpaket zu einem zweiten Anwender, und dieser überträgt es anschließend zurück (Quittungs- und Kontrollpakete werden dabei nicht übertragen). Als Basisdienst wird der in Abschnitt 4.1.1 vorgestellte Netzwerksimulator bei einer Verzögerung von 0.9 ms (keine Ratenbegrenzung, kein Verlust oder Verfälschung) eingesetzt. Die im folgenden genannten Übertragungszeiten beziehen sich dabei auf eine Ende-zu-Ende Übertragung. Dieses Benutzerprofil erlaubt eine quantitative Bestimmung der reinen Paketbearbeitungszeiten durch die XTP Protokollautomaten, da zu jedem Zeitpunkt nur in einer Komponente des Systems Aktivität besteht und so der Ablauf der Übertragung nicht-konkurrierend (und somit deterministisch) verläuft.

Die in Tabelle 1 dargestellten Übertragungszeiten gelten für die Kommunikation zwischen zwei Client-Prozessen über SandiaXTP Protokolldämons (siehe Abbildung 2) und den Netzwerksimulator. Dies schließt die Zeiten für die Kontextwechsel zwischen Clientprozessen und den

4. Die Meßbedingungen sind in Anhang A zu finden.

	übertragene Nutzdaten [Byte]					
	10		1,000		10,000	
	absolut	relativ†	absolut	relativ	absolut	relativ
SandiaXTP	2.64 ms	1	3.36 ms	1.3	4.52 ms	1.7

Table 1: Ende-zu-Ende Übertragungszeit bei *separaten* Clientprozessen

†. Die Relativwerte in dieser und den folgenden Tabellen beziehen sich jeweils auf die Zeiten von SandiaXTP bei 10 Byte Nutzdaten.

Protokollprozessen (2 * 0.47 ms), die Übertragungszeit des Netzwerksimulators (0.9 ms) und die Prüfsummenberechnung ein.

Diese Werte sind zunächst nicht für einen direkten Vergleich mit den Implementierungen der Estelle-Spezifikation geeignet, da die Zeiten für den Kontextwechsel zwischen Client und Protokollmaschine bei den Estelle-Implementierungen aufgrund ihrer Integration in einen Prozeß nicht anfallen. Eliminiert man nun diese Verzögerung (2 * 0.47 ms) aus den Übertragungszeiten von SandiaXTP, so ergeben sich die in Tabelle 2 angegebenen Zeiten. Diese beinhalten weiterhin die Übertragungszeit des Netzwerksimulators (0.9 ms) und die Prüfsummenberechnung.

In der Tabelle sind ebenfalls die entsprechenden Übertragungszeiten der mit EC und PET/DINGO erzeugten Implementierungen dargestellt, wobei jeweils drei verschiedene (bei der Übersetzung festgelegte) Obergrenzen für die Nutzdatenmenge einer Übertragung ("Limit" 10, 1000 und 10000 Bytes) unterschieden werden. Da in der Estelle-Spezifikation bei der Übertragung eines Pakets über einen Interaktionspunkt nicht nur die *genutzte* sondern diese *maximale Datenmenge* übertragen wird (siehe Abschnitt 4.1.2), hat die Obergrenze einen erhebliche Einfluß auf die Übertragungszeiten.

		übertragene Nutzdaten [Byte]					
		10		1,000		10,000	
Implementation	Limit [Byte]	absolut	relativ	absolut	relativ	absolut	relativ
SandiaXTP	—	1.70 ms	1	2.42 ms	1.4	3.58 ms	2.1
EC	10	3.96 ms	2.3	—	—	—	—
	1,000	4.56 ms	2.7	7.64 ms	4.5	—	—
	10,000	11.1 ms	6.5	14.2 ms	8.4	38.5 ms	22.6
PET/DINGO	10	34.8 ms	20.5	—	—	—	—
	1,000	37.3 ms	22.0	40.3 ms	23.7	—	—
	10,000	82.6 ms	48.6	85.6 ms	50.4	109.2 ms	64.3

Table 2: Ende-zu-Ende Übertragungszeit bei *integrierten* Clients

Ein Vergleich der reinen Verarbeitungszeiten vom Senden des Pakets im Client bis zur Übergabe an das Medium (bzw. umgekehrt) spiegelt das Verhältnis der Leistungsfähigkeit der Protokollautomaten ohne Beeinflussung durch das Medium wider. Dazu muß die Verzögerung durch das Medium (0.9 ms) von den ermittelten Übertragungszeiten abgezogen werden (siehe Tabelle 3 und Abbildung 5). Da dies alle Protokollimplementierungen gleichermaßen betrifft, ergeben sich die relativen Unterschiede wesentlich ausgeprägter.

Die Werte für verschiedene Paketgrößen erlauben Rückschlüsse auf die Ursachen des Zeitverbrauchs bei der Paketverarbeitung. So dominiert bei Paketen mit 10 Byte Nutzdaten (Obergrenze ebenfalls 10 Byte) deutlich der Aufwand für die Verarbeitung der Paketheader gegenüber dem Kopieraufwand für die Nutzdaten. In diesem Bereich kann die von EC erzeugte Implementierung mit einem Zeitfaktor von 3.8 fast in den Leistungsbereich von SandiaXTP vorstoßen.

		übertragene Nutzdaten [Byte]					
		10		1,000		10,000	
Implementation	Limit [Byte]	absolut	relativ	absolut	relativ	absolut	relativ
SandiaXTP	—	0.80 ms	1	1.52 ms	1.9	2.68 ms	3.4
EC	10	3.06 ms	3.8	—	—	—	—
	1,000	3.66 ms	4.6	6.74 ms	8.4	—	—
	10,000	10.2 ms	12.8	13.3 ms	16.6	37.6 ms	47.0
PET/DINGO	10	33.9 ms	42.4	—	—	—	—
	1,000	36.4 ms	45.0	39.4 ms	49.2	—	—
	10,000	81.7 ms	102.1	84.7 ms	105.9	108.3 ms	135.4

Table 3: Ende-zu-Ende *Verarbeitungszeit* bei integrierten Clients

Die von DINGO generierte Implementierung ist dagegen mehr als eine Zehnerpotenz langsamer. Betrachtet man nun die Vergrößerung des Zeitverbrauchs beim Übergang zu 10,000 Byte Paketen, so erhält man ein Maß für den Kopieraufwand in den Protokollmaschinen. Hier zeigt sich bei SandiaXTP mit 1.88 ms Differenz zum 10 Byte Paket der Vorteil der zentralen Paketpufferung, durch die mit lediglich vier Kopieroperationen ein Paket von Ende zu Ende transportiert werden kann (siehe Abschnitt 3.1). Die Estelle-Spezifikation erfordert dagegen eine weitaus größere Anzahl von Kopieroperationen (siehe Abschnitt 3.2), die sich im Verhältnis zu SandiaXTP in der 18-fachen (34.5 ms bei EC) bzw. 40-fachen (74.4 ms bei DINGO) Steigerung der Übertragungsdauer bei großen Paketen widerspiegelt.

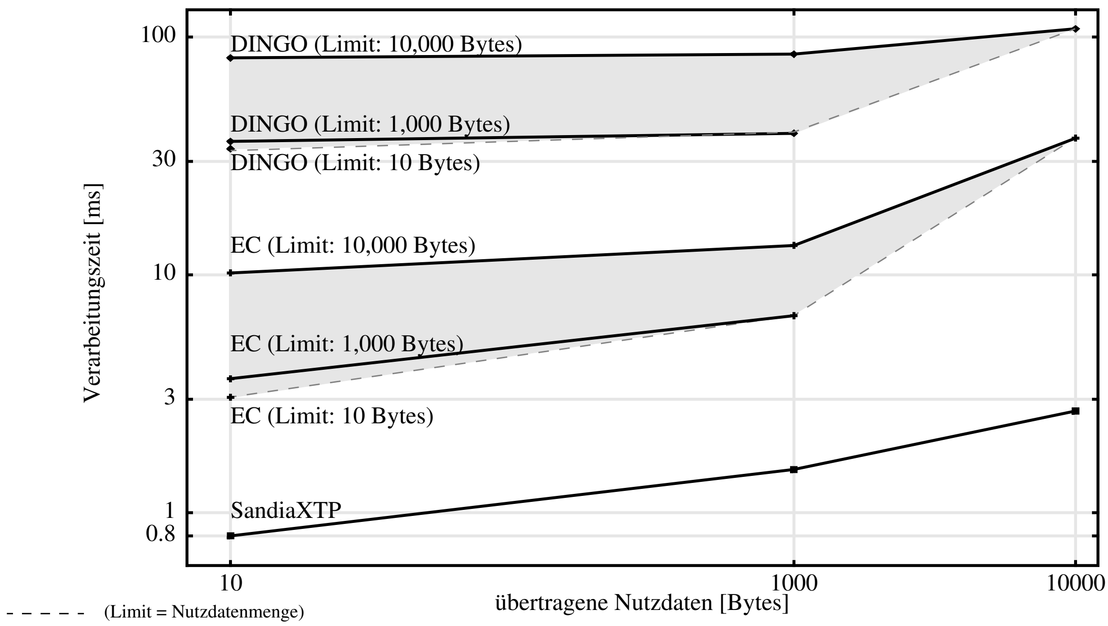

Abbildung 5: Doppelt-logarithmische Darstellung der Ende-zu-Ende *Verarbeitungszeit* bei integrierten Clients (siehe Tabelle 3)

Die Ergebnisse machen deutlich, daß aus Estelle-Spezifikationen automatisch generierte Implementierungen bei kleinen Datenpaketen durchaus in die Größenordnung von gut optimierten Handimplementierungen vorstoßen können. Dies gilt besonders vor dem Hintergrund, daß die *zugrundegelegte Estelle-Spezifikation ohne Berücksichtigung von Effizienzaspekten erstellt wurde*. Gleichzeitig werden aber auch einige Schwächen von Estelle und den daraus abgeleiteten Implementierungen deutlich: Da Module das wesentliche Strukturierungsmittel von Estelle sind, werden gerade große Spezifikation sinnvollerweise in eine Vielzahl von Modulen zerteilt. Dies führt jedoch wegen der beschränkten Möglichkeiten für gemeinsamen Speicher zu vielen

Nachrichtenübertragungen und damit zu vielen Kopieroperationen, was speziell bei großen Paketen die Effizienz merklich verschlechtert. Dies wird zusätzlich dadurch verschärft, daß Estelle keine Datenarrays variabler Größe kennt und daher immer Pakete maximaler Größe übertragen werden müssen (s.o.). Wir werden in Abschnitt 5 einige Entwicklungen aufzeigen, durch die diese Schwächen überwunden werden können.

4.2 Ein Estelle-Benchmark

Aufbauend auf die Untersuchungen der XTP Spezifikation wurde ein Estelle-Benchmark [Dah96] entwickelt, der es gestattet, die automatische Implementierung derjenigen Estelle-Mechanismen isoliert und quantitativ zu bewerten, die sich als besonders laufzeitintensiv erwiesen haben. So können z.B. Modulinstanziierungen, Nachrichtenübertragungen mit unterschiedlich großen Argumenttypen, verschiedenen Auswahlsituationen usw. auch ohne Quellen für Compiler oder Laufzeitbibliothek (wie z.B. bei EC) quantitativ bewertet werden.

Bisher liefert dieser Benchmark nur Einzelwerte für die verschiedenen Estelle-Mechanismen. Wir erarbeiten zur Zeit jedoch ein Gewichtungsschema, das es erlauben soll, aus diesen Einzelwerten Prognosen für die Leistung der Compiler für verschiedene Anwendungsgebiete abzuleiten. Erste Ergebnisse für den Vergleich zwischen EC, DINGO und dem UHH-Estelle-Compiler bestätigen die relativen Ergebnisse aus dem XTP-Vergleich und positionieren den UHH-Compiler zwischen EC und DINGO. So benötigt z.B. der UHH-Estelle-Compiler beim Austausch von Interaktionen mit einem Nettodatenanteil von 1000 Byte die 3-fache, PET/DINGO die 7.5-fache Zeit im Vergleich zu EC.

5 Ausblick

Anhand des Hochleistungsprotokolls XTP und der FDT Estelle wurde untersucht, inwieweit automatisch aus einer formalen Spezifikation generierte Implementierungen in ihrer Leistungsfähigkeit mit Handimplementierungen konkurrieren können. Ausgangspunkt für die Untersuchungen waren eine Estelle-Spezifikation von XTP 4.0 sowie die SandiaXTP-Handimplementierung dieses Protokolls. Es konnte gezeigt werden, daß automatisch generierter Kode fast in den Leistungsbereich effizienter Handimplementierungen vorstoßen kann. Dies ist insbesondere auch deshalb bemerkenswert, weil bei der automatischen Implementierung — anders als bei der bereits umfassend optimierten Handimplementierung — noch ein erhebliches ungenutztes Potential zur Leistungsverbesserung besteht.

Um dieses Potential zu quantifizieren und zu nutzen, soll in zukünftigen Arbeiten die Optimierung der automatischen Implementierung gezielt vorangetrieben werden. Dabei gibt es verschiedene Ansatzpunkte, die zusammen verfolgt werden sollen. Ein erster Ansatzpunkt ist die Optimierung der Estelle-Spezifikation von XTP. Wie bereits diskutiert, wurden Performanceaspekte bei der Entwicklung der für die Messungen verwendeten Estelle-Spezifikation bislang nicht berücksichtigt. So bedingt z.B. die komplexe Modulhierarchie des Protokollautomaten einen hohen internen Kopier- und Synchronisationsaufwand in der generierten Implementierung. Ferner soll der Einfluß einzelner Spezifikations-Stilelemente auf das Leistungsverhalten durch Estelle-Benchmarks und den Einsatz des Werkzeugs PATO quantifiziert werden, um so eine gezielte Optimierung zu ermöglichen. Ein zweiter Ansatzpunkt zur Optimierung besteht in der Verbesserung des generierten Kodes. Ein experimenteller Kodegenerator, der die Untersuchung der Auswirkungen verschiedener Implementierungsverfahren auf das Leistungsverhalten erlaubt, befindet sich in der Entwicklung. Erste Leistungsvergleiche mit den in dieser Arbeit genannten Estelle-Compilern belegen diesen Ansatz. Ein weiterer Ansatzpunkt ist die

Weiterentwicklung der FDT Estelle. Beispielsweise kann der Kopier- und Synchronisationsaufwand durch gezielte Spracherweiterungen [BrGo94] erheblich reduziert werden.

Literatur

[BrGo94] Bredereke, J., Gotzhein, R.: *Increasing the Concurrency in Estelle*, in: R. L. Tenney, P. D. Amer, M. U. Uyar (Hrsg.), *Formal Description Techniques VI*, North-Holland, 1994

[Bud93] Budkowski, S. (Hrsg.): *Formal Specification, Validation and Performance Evaluation of the Xpress Transfer Protocol (XTP)*, Research Report No. 931004, Institute Nationale des Télécommunications, Evry, Frankreich, 1993

[Dah96] Dahl, S.: *Performancevergleich von Protokollimplementierungen*, Diplomarbeit, Universität Kaiserslautern, Oktober 1996

[DeBu89] Dembinski, P., Budkowski, S.: *Specification Language Estelle*, in: M. Diaz et al. (eds.), *The Formal Description Technique, Estelle*, North-Holland, 1989, pp. 35-75

[GRT96] Gotzhein, R., Rößler, F., Thees, J.: *Towards "Open Estelle"*, in: 6. *GI/ITG Fachgespräch "Formale Beschreibungstechniken für verteilte Systeme"*, Erlangen-Nürnberg, May 1996

[ISO89] ISO/TC97/SC21: *Estelle - A Formal Description Technique Based on an Extended State Transition Model*, ISO/TC97/SC21, IS 9074, 1989

[KrGo93] Kreuz, D., Gotzhein R.: *A Compiler for the Parallel Execution of Estelle Specifications*, in: H. König (Hrsg.), *Formale Methoden für verteilte Systeme*, Fokus-Band 8, Saur-Verlag, München, 1993

[Pen96] Penner, H.: *Erstellung eines Software-Monitors zur Analyse automatisch generierter Protokollimplementierungen*, Diplomarbeit, Universität Kaiserslautern, Januar 1996

[RiCl89] Richard, J. L., Claes, T.: *A Generator for C-Code for Estelle*, in: M. Diaz et al (eds.), *The Formal Description Technique Estelle*, North-Holland, 1989, pp. 397-420

[SNL95a] Sandia National Laborities: *SandiaXTP Reference Manual*, Release 1.4, Livermore, California, USA, September 1995

[SNL95b] Sandia National Laborities: *SandiaXTP — An Object-Oriented Implementation of XTP 4.0 Derived from the Meta-Transport Library: User Manual*, Release 1.4, Livermore, California, USA, October 1995

[SiSt90] Sijelmassi, R., Strausser, B.: *NIST Integrated Tool Set for Estelle*, in: Quemada, J., Manos, J., Vazquez, E.: *Third International Conference on Formal Description Techniques (FORTE'90)*, Madrid, Spanien, 1990

[SiSt93] Sijelmassi, R., Strausser, B.: *The PET and DINGO tools for deriving distributed implementations from Estelle*, Computer Networks and ISDN Systems, Vol. 25, No. 7, 1993, pp. 1115-1130

[XTP95] XTP Forum, *Xpress Transport Protocol Specification*, XTP Rev. 4.0, XTP Forum, Santa Barbara, USA, 1995

Anhang A: Randbedingungen der Leistungsmessungen

- *Hardware:* SUN SPARCstation-20, 192 MB, 2 Prozessoren (75 MHz)
- *Betriebssystem:* SUN Solaris 2.5
- *C/C++ Compiler:* GNU gcc 2.7.2 mit Option "-O2 -msupersparc"
- *Estelle-Compiler:* EC 3.1, PET/DINGO 1.0, UHH 1.4