Atomarität in parallel implementierten Estelle-Spezifikationen

Jan Bredereke
Universität Kaiserslautern, Postfach 3049, D-67653 Kaiserslautern
E-mail: bredereke@informatik.uni-kl.de

Zusammenfassung

In nebenläufigen Systemen erleichtert das Konzept der Atomarität von Operationen, konkurrierende Zugriffe in größere, leichter beherrschbare Abschnitte zu unterteilen. Wenn wir aber Spezifikationen in der formalen Beschreibungstechnik Estelle betrachten, erweist es sich, daß es unter bestimmten Umständen schwierig ist, die Atomarität der sogenannten Transitionen bei Implementationen exakt einzuhalten, obwohl diese Atomarität eine konzeptuelle Grundlage der Semantik von Estelle ist. Es wird aufgezeigt, wie trotzdem sowohl korrekte als auch effiziente nebenläufige Implementationen erreicht werden können. Schließlich wird darauf hingewiesen, daß die das Problem auslösenden Aktionen oft vom Spezifizierer leicht von vorneherein vermieden werden können; und dies gilt auch über den Kontext von Estelle hinaus.

1 Einleitung

Das Konzept der Atomarität von Operationen erleichtert es einem Systementwerfer, konkurrierende Zugriffe in größere, leichter beherrschbare Abschnitte zu unterteilen. In parallel implementierten und in verteilten Systemen mit ihrer hohen Nebenläufigkeit ist dieses Konzept daher wichtig. Eine Operation wird allgemein als atomar bezeichnet, wenn stets entweder noch keine oder bereits alle Auswirkungen dieser Operation beobachtbar sind. Bei einer atomaren Operation ist es nicht möglich, daß in einer Beobachtung erst einige ihrer Auswirkungen sichtbar sind, andere aber noch nicht.

Da in einer Implementation jede Berechnung des Ergebnisses einer Operation eine Zeit größer als Null braucht und da dabei gewöhnlich Zwischenzustände auftreten, müssen besondere Vorkehrungen getroffen werden, um die Implementation einer Operation atomar zu machen. Üblicherweise bestehen diese darin, alle anderen Operationen, die auf den betreffenden Teil des Zustandsraumes zuzugreifen versuchen, zu verzögern, bis die erste Operation abgeschlossen ist.

Die formale Beschreibungstechnik Estelle ([ISO89, Hog89]) ist als Mittel zur präzisen Spezifikation von nebenläufigen informationsverarbeitenden Systemen entwickelt worden, insbesondere für die Standardisierung der verschiedenen Protokolle innerhalb des OSI-Basisreferenzmodells ([ISO81]).

Im Rahmen von Untersuchungen über eine Erweiterung von Estelle zur Steigerung der spezifizierbaren Nebenläufigkeit ([BrGo93]) erkannten wir einige Probleme, die die sogenannten Estelle-Transitionen in Bezug auf die Atomarität aufwerfen, sofern sie eine nicht-lokale Wirkung besitzen. Diese Probleme werden relevant, sobald man eine parallele Implementation von Estelle-Spezifikationen durchführt.

Bisherige Untersuchungen (s.u.) waren offensichtlich ungenau, daraus resultierende parallele Implementationen von Estelle-Spezifikationen halten die Semantik von Estelle nicht immer genau ein. Unter bestimmten Bedingungen können dort *Races* auftreten, als deren Konsequenz schließlich die Atomarität von Estelle-Transitionen nicht mehr gegeben ist. Diese Atomarität aber ist ein grundlegendes Konzept von Estelle. Zusammen mit dem sogenannten Interleaving dient sie zur Beschreibung der kombinierten Wirkung von nebenläufigen Aktionen. Und da Estelle insbesondere zur Spezifikation von parallelen — und damit nebenläufigen — Systemen vorgesehen ist, sollte es hier keine Zweifel geben, inwieweit Implementationen ihren Spezifikationen entsprechen.

2 Estelle

Estelle wurde von der International Standardization Organization (ISO) entwickelt und besitzt seit 1989 den Status eines internationalen Standards ([ISO89]). Eine Estelle-Spezifikation beschreibt ein hierarchisch aufgebautes System von kommunizierenden erweiterten endlichen Automaten, den *Modulinstanzen*. Die Struktur eines solchen Systems kann sich durch die Erzeugung und Freigabe von Modulinstanzen dynamisch ändern. Für das Voranschreiten der Modulinstanzen lassen sich abgestufte Grade von Parallelität spezifizieren. Oben in der Modulhierarchie finden sich die sogenannten Subsysteme, auch Systemmodulinstanzen genannt. Diese arbeiten vollständig asynchron zueinander. Alle weiteren, ihnen untergeordneten Sohnmodulinstanzen arbeiten pro Unterbaum entweder synchron parallel oder synchron sequentiell.

Die möglichen Zustandsübergänge einer Modulinstanz werden durch Stücke sequentieller, Pascal-ähnlicher Anweisungsfolgen spezifiziert, die *Transitionen*. Das Schalten dieser Transitionen kann unter anderem durch den Empfang von Nachrichten ausgelöst werden, und als Wirkung können wiederum auch Nachrichten an andere Modulinstanzen versandt werden. Das Schalten einer Transition geschieht dabei stets *atomar*. In der Definition der Semantik von Estelle ([ISO89]) ist festgelegt, daß alle Auswirkungen dieses Schaltens von einem Zustand des Ge-

samtsystems auf den nächsten eintreten müssen[1].

In Estelle gibt es Operationen, deren Wirkung ausschließlich *lokal* zu der Modulinstanz ist, die sie ausführt (Zuweisung auf Variablen, ...). Es gibt aber auch Operationen, deren Wirkung *nicht lokal* ist. Hierbei handelt es sich insbesondere um die Übertragung von Nachrichten zu anderen Modulinstanzen.

3 Parallele Implementationen

Wir können zwei Möglichkeiten unterscheiden, um eine Implementation zu einer Spezifikation in Estelle zu erhalten. Einerseits kann das gesamte System auf einer rein sequentiellen Hardware-Architektur ausgeführt werden. (Diesen Weg gehen die meisten Simulatoren.) Andererseits kann das System auch auf verschiedene Knoten einer parallelen Architektur verteilt werden.

Bei einer sequentiellen Implementation ist es recht einfach, die Atomarität der Transitionen zu realisieren, indem einfach ein globaler Scheduler entworfen wird, der jeweils den zu einer Transition erzeugten Code vollständig ausführt.

Bei einer parallelen Implementation ist dies schon schwieriger. Als Beispiel sei hier das Werkzeug Pet/Dingo ([SiSt93]) genannt, welches Estelle in verteilten C++-Code übersetzt. Jedes der ausführbaren Programme hat nun einen eigenen Scheduler. Die Synchronisation erfolgt durch Nachrichtenaustausch, wobei Pet/Dingo zugleich mit der Weitergabe der Kontrolle von Scheduler zu Scheduler auch die von den Transitionen ausgegebenen und zunächst gesammelten Nachrichten weiterreicht (und so geschickt den Kommunikationsaufwand verkleinert).

In Estelle können Kommunikationsverbindungen zwischen zwei Partnermodulinstanzen nur durch ihre gemeinsame Vatermodulinstanz errichtet werden, und um bestimmte Synchronisationsbedingungen einzuhalten, muß auch alle Kommunikation über den Vater erfolgen. Da sich Systemmodulinstanzen (also diejenigen, die in der Modulhierarchie oben stehen) nicht synchronisieren müssen, und da ihre Kommunikationsstruktur darüberhinaus statisch ist, enthält Pet/Dingo eine Optimierung: In Pet/Dingo kommunizieren Systemmodulinstanzen nicht über ihre gemeinsame, ansonsten inaktive Vatermodulinstanz, sondern die Vatermodulinstanz teilt ihren Söhnen einmalig die Adressen ihrer jeweiligen Kommunikationspartner mit und terminiert dann. Damit können die Söhne dann direkt kommunizieren, womit auch der Kommunikationsflaschenhals des Weges über die Wurzel der Modulstruktur etwas entschärft wird.

In Abschnitt 3.1 werden wir eine Estelle-Spezifikation vorstellen, die ein auf das wesentliche abgemagertes Beispiel für eine Situation mit *Races* darstellt. Diese „Wettläufe“ zwischen Signalen, und die daraus resultierenden Synchronisationsprobleme, stellen eine sehr grundlegende Schwierigkeit bei der korrekten Er-

[1] Die Definition der Semantik von Estelle basiert zwar der Einfachheit halber auf einem einzigen globalen Zustandsvektor, dieser ist aber derart strukturiert, daß er für eine parallele Implementierung zerteilt werden kann.

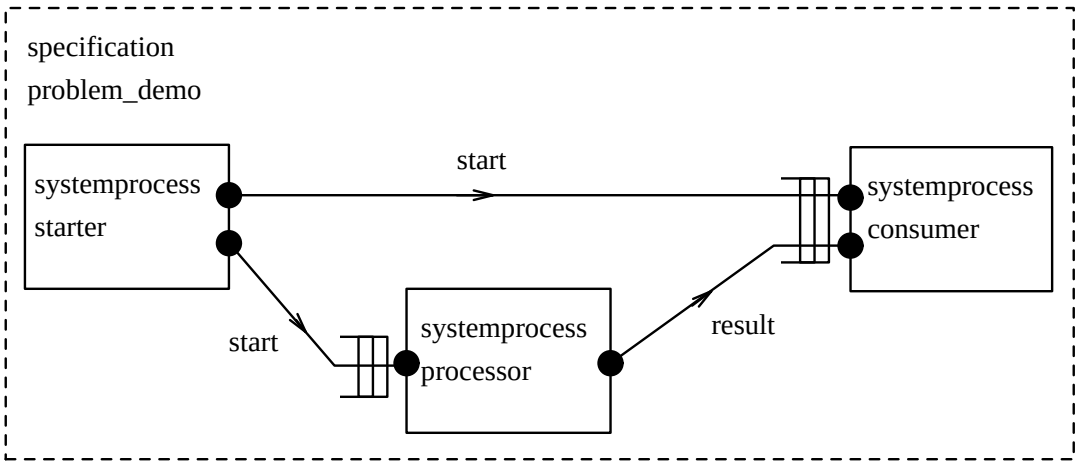

Abbildung 1: Die Systemstruktur des Demonstrationsbeispieles

stellung von nebenläufigen kommunizierenden Systemen dar, und sie sind oft die Quelle von unvorhergesehenen, unerklärlichen Fehlern.[2] In Abschnitt 3.2 werden wir zeigen, daß diese Races von existierenden Übersetzern nicht korrekt — gemäß der Semantik von Estelle — gehandhabt werden.

3.1 Eine verflixte Spezifikation

Die Struktur einer Estelle-Spezifikation, die einer parallelen Implementation Schwierigkeiten bei der Einhaltung der Estelle-Semantik macht, findet sich in Abbildung 1, die vollständige Spezifikation selbst im Anhang. Das System besteht aus drei Modulinstanzen: Die Modulinstanz `starter` gibt ein Startsignal an die Modulinstanz `processor`, die daraufhin irgendwelche Berechnungen ausführt und das Ergebnis an die Modulinstanz `consumer` weiterreicht. Zusätzlich unterrichtet `starter` aber auch `consumer` von dem Startsignal, und zwar atomar in derselben Transition, in der auch `processor` benachrichtigt wird.

Folglich muß das Startsignal bei `consumer` auf jeden Fall vor dem Ergebnis `result` eintreffen. Denn die Wirkung der Transition von `starter` besteht darin, beide Nachrichten in die jeweiligen Empfängerwarteschlangen einzureihen. Erst danach kann `processor` seine Nachricht empfangen und damit wiederum sein Ergebnis an `consumer` senden. Die zuerst eintreffende Nachricht wird auch zuerst von `consumer` bearbeitet, da `consumer` eine gemeinsame Warteschlange (`common queue`) für alle eingehenden Nachrichten besitzt. Schließlich besitzt die Modulinstanz `consumer` für jede der beiden Nachrichten eine eigene Empfangstransition, und sie gerät in einen Verklemmungszustand, falls die beiden Nachrichten (unter Verletzung der Semantik von Estelle) in vertauschter Reihenfolge eintreffen

[2] Formale Methoden sind ein Versuch, solche Situationen explizit erkennbar zu machen, um sie dann beheben zu können. Daher auch unser Bemühen, die Implementationen mit der formalen Semantik von Estelle exakt zur Deckung zu bringen.

sollten.

3.2 Existierende Übersetzer

Die meisten der bisher existierenden Übersetzer für Estelle erzeugen sequentiellen Code zur Ausführung auf nur einem Rechner, womit dort besondere Schwierigkeiten bei der Erhaltung der Atomarität gar nicht erst entstehen.

Von den Erstellern der existierenden Übersetzer, die Code zur verteilten Ausführung erzeugen, ist das im folgenden beschriebene Problem mit der Atomarität offensichtlich nicht bemerkt worden.

In unserer Spezifikation zeigt sich dieses Problem bei dem Versenden der beiden Nachrichten von `starter`. In Pet/Dingo etwa sieht die Realisierung folgendermaßen aus: Während der Bearbeitung einer Transition werden alle ausgegebenen Nachrichten zunächst in einer temporären Ausgangswarteschlange zwischengespeichert, damit keine Wirkung nach außen hin sichtbar wird, bevor die Transition nicht fertig bearbeitet ist. Im Falle von Systemmodulinstanzen wird diese Ausgangswarteschlange nach Beendigung der Transition dann zügig, aber sequentiell mit Hilfe einer `for`-Schleife geleert, wobei in jeder Iteration eine Nachricht versandt wird.

Ist nun aber eine von Pet/Dingo generierte Implementation von `starter` *sehr* langsam, so kann es geschehen, daß zwischen dem Übermitteln des Startsignals an `processor` und an `consumer` so viel Zeit verstreicht, daß `processor` seine Arbeit bereits erledigt und sein Ergebnis bereits an `consumer` übermittelt hat. Damit steht das Ergebnis in der gemeinsamen Empfängerwarteschlange vornean. In diesem Falle gerät `consumer` in einen Verklemmungszustand, in den er gemäß der Semantik von Estelle niemals gelangen dürfte. Aufgrund der unerwarteten Reihenfolge der Nachrichten in der gemeinsamen Empfängerwarteschlange ist keine der beiden Empfangstransitionen bereit, die erste Nachricht zu entnehmen. (Denkbare Ursachen für solch eine Verzögerung könnten etwa eine Überlastung eines Kommunikationsmediums sein, die das Versenden der zweiten Nachricht verzögert, eine ungünstige Prozessorvergabestrategie, oder auch einfach nur der Ablauf einer Zeitscheibe.)

Auch in [Pet91] (vergleiche auch [KrGo93]), wo der UHH-Estelle-Übersetzer entworfen und implementiert wird, wird das Problem übersehen, obwohl das Thema der Erhaltung der korrekten Semantik von Estelle bei verteilter Implementation umfassend beleuchtet wird.

Ein Benutzer einer der Übersetzer zur verteilten Ausführung muß den Fehler nicht unbedingt bemerken. Die Korrektheit des spezifizierten Protokolls hängt möglicherweise nicht von der Atomarität der betreffenden Transitionen ab, oder die entsprechenden Teile des Laufzeitsystems haben bisher schnell genug gearbeitet, so daß sich der Fehler *noch* nicht gezeigt hat.

4 Semantik

Bei der Betrachtung der im letzten Kapitel beschriebenen Race-Situation muß man unterscheiden einerseits zwischen der Semantik der Estelle-Spezifikation und andererseits zwischen einer (parallelen und verteilten) Implementation dazu. In der Definition der Semantik macht die Atomarität keinerlei Probleme. Es ist lediglich nicht so leicht, diese Semantik in einer parallelen und verteilten Implementation umzusetzen.

Verteilte Systeme sind dadurch gekennzeichnet, daß es keine Instanz mehr gibt, die den gesamten Systemzustand kennt. Diese Information kann man nur durch eine globale Synchronisation erhalten, weshalb man schließen kann, daß in verteilten Systemen eine globale Synchronisation entweder nicht möglich oder aber zumindest (des Aufwandes wegen) nicht gewünscht ist.

Nicht-lokale, aber atomare Aktionen wie das Schalten der beschriebenen Estelle-Transitionen erfordern nun eine Synchronisation der beteiligten Instanzen, was nur durch zusätzlichen Aufwand (Warten, Protokoll dafür) möglich ist. Daher laufen diese Aktionen zunächst dem Konzept der verteilten Systeme zuwider. Es stellt sich die Frage, ob dieser zusätzliche Aufwand notwendig ist, oder ob es sinnvoll ist, die Definition der Semantik von Estelle zu verändern und die Atomarität höchstens noch für die lokalen Aspekte einer Transition zu fordern, so daß korrekte Implementationen leichter möglich werden. (Nicht-lokale Kommunikation zu verbieten, ist sicher keine Lösung.) Damit bleiben als Alternativen:

1. Aufgabe der Atomarität von Transitionen
2. Zulassen einer Laufzeit größer als Null für Nachrichten bis zur Einreihung in die Empfängerwarteschlange

Wie bereits erwähnt, ist das Konzept der Atomarität sehr grundlegend, es dient zusammen mit dem Interleaving zur Beschreibung der Semantik der Nebenläufigkeit in Estelle. Wenn man also nicht die Definition der Semantik völlig anders aufbauen will, muß man statt der Atomarität von Transitionen eine Atomarität auf kleinerem Maßstab einführen, z.B. auf dem von (Ausgabe-)Anweisungen. Dies würde die Beschreibung der Bedeutung einer Spezifikation erheblich komplexer machen. Nicht nur formale Korrektheitsüberprüfungen von Estelle-Spezifikationen, z.B. durch Erreichbarkeitsuntersuchungen, wären dann nicht mehr praktikabel, sondern es würde sich die Semantik auch erheblich weiter von dem intuitiven Verständnis des Spezifizierers entfernen, da er kaum in der Lage sein wird, alle Permutationen des Interleavings von einzelnen Anweisungen der Transitionen zu bedenken. Es ist bereits schwierig genug, dies auf der Ebene von Transitionen zu tun. Die Atomarität von Transitionen sollte also auf jeden Fall als Konzept erhalten bleiben.

Auch wenn wir zulassen, daß Nachrichten eine längere Zeit bis zum Empfänger brauchen, wird die Spezifikation erheblich unübersichtlicher. Da man sich kaum

auf eine feste Anzahl von Transitionsrunden festlegen möchte, nach der eine Nachricht eingetroffen sein muß, muß man ihr eine beliebig lange Laufzeit zugestehen. Dann kann es geschehen, daß auch eine synchron parallel arbeitende Sohn-Modulinstanz ihrer Bruder-Instanz eine Nachricht sendet, diese aber in der nächsten Transitionsrunde noch lange nicht eingetroffen ist.

Eine weitere Variante wäre, Laufzeiten nur zwischen asynchronen (System-) Modulinstanzen zuzulassen. Aber auch dies würde recht unübersichtlich sein, da wir nun zwei leicht verschiedene Arten von Nachrichtenkanälen hätten, die vom Spezifizierer leicht verwechselt werden können.

Jede der diskutierten Veränderungen der Semantik würde erhebliche Nachteile bringen. Wir sollten unser Augenmerk daher besser auf die Möglichkeiten richten, wie man auch die bestehende Semantik sowohl korrekt als auch effizient implementieren kann. Daß dies möglich ist, werden wir im folgenden Kapitel sehen.

5 Wege zu korrekten Implementationen

In Kapitel 5.1 werden wir betrachten, wie man sicher eine bezüglich der Estelle-Semantik korrekte Implementation erhalten kann, und in Kapitel 5.2, wie man diese Lösung stark optimieren kann.

Bei all diesen Betrachtungen nehmen wir die Erfülltheit einer einfachen Randbedingung an: In der Implementation muß der Empfang einer einzelnen Nachricht durch eine Empfangsroutine atomar geschehen, eventuelle lokale Zugriffskonflikte in der Implementation des Empfängers müssen gelöst sein. (Z.B. zwischen einer Scheduler-Routine und einer parallel dazu arbeitenden Empfangsroutine.) Dies läßt sich leicht mit den vorhandenen Mitteln des jeweiligen Betriebssystems, etwa Semaphoren oder ähnlichem, bewerkstelligen.

Weiterhin gehen wir von der praxisorientierten Annahme aus, daß in der Implementation zwischen dem Aufruf einer Senderoutine und dem Empfang der zugehörigen Nachricht durch eine Empfangsroutine Zeit vergehen kann.

5.1 Ein überarbeitetes Protokoll

Um die Atomarität von nicht-lokalen, atomaren (Sende-)Aktionen zu sichern (wie etwa die Transition der Modulinstanz `starter` in Kapitel 3), müssen sich die Implementationen der beteiligten Modulinstanzen synchronisieren. Dies bedeutet, daß sie ihr asynchrones Voranschreiten anhalten müssen und warten müssen, bis alle an der Aktion teilgenommen haben.

Ein Protokoll dazu könnte etwa so aussehen, daß der Empfänger einer nichtlokal versandten Nachricht automatisch in einen Wartezustand geht, bis er eine weitere Nachricht vom Sender bekommen hat, daß alle anderen Empfänger die Nachricht ebenfalls bekommen haben. Bei dem Übergang in den Wartezustand

muß er natürlich den Emfang sofort quittieren, damit der Sender die korrekte Zustellung überprüfen kann. Nach Eintreffen aller Quittungen kann der Sender dann alle Freigabenachrichten absenden. Bis zu diesem Zeitpunkt muß sich auch der Sender im Wartezustand befinden. Damit benötigen wir insgesamt zu jeder Nachricht noch zwei Quittungen.

In diesem Protokoll besteht die Gefahr von Deadlocks, falls sich die Implementationen zweier Modulinstanzen gleichzeitig gegenseitig Nachrichten senden wollen. Diese Situation kann aber leicht dadurch erkannt werden, daß sich die Implementation einer Modulinstanz bereits im Wartezustand befindet, wenn sie eine Aufforderung zum Warten erhält. In dieser speziellen Situation sind weitere Maßnamen (d.h. ein weiteres Protokoll) notwendig, um die konkurrierenden Aktionen zu sequentialisieren, und insbesondere, um sich auf eine Reihenfolge dafür zu einigen.

Man kann hier den Ansatz verfolgen, daß solche Situationen vermutlich selten auftreten werden. Damit kann das normale, einfachere Protokoll die Gefahr von Deadlocks in Kauf nehmen, und diese können dann in den seltenen Fällen, in denen sie tatsächlich auftreten, mit Hilfe der bekannten Algorithmen zur Abstimmung in verteilten Systemen aufgelöst werden (z.B. „niedrigste ID-Nummer gewinnt“).

5.2 Eine wesentliche Optimierung

Kommen wir nun zu der Optimierung. Das ganze, beschriebene *Protokoll kann vollständig entfallen* (natürlich bis auf die eigentliche Nachricht), solange für alle Transitionen eine der folgenden Bedingungen erfüllt ist, unter denen unser Atomaritätsproblem ausgeschlossen ist:

1. Die betrachtete Transition enthält *höchstens eine* `output`-*Anweisung*.

2. Es werden zwar mehrere *Nachrichten* in einer Transition versandt, aber *höchstens eine nicht lokal*. (D.h. nicht an eigene interne Interaktionspunkte und nicht an — nur im Wechsel mit dem Vater arbeitende — Sohn-Modulinstanzen.)

3. Es wird *maximal eine* Nachricht an eine Modulinstanz versandt, die *nicht zum selben Subsystem-Modulinstanzbaum* gehört.

4. Es werden zwar mehrere Nachrichten an verschiedene asynchron arbeitende Subsysteme versandt, aber zwischen den *Empfängern* ist jeweils *keine Kommunikationsverbindung* möglich.

Die letztere Bedingung ist erfüllt, wenn keine entsprechende Estelle-Kanaldefinition spezifiziert ist (auch nicht indirekt über ein drittes Modul). Für eine

vorgegebene Menge von Empfängern läßt sich diese Bedingung aufgrund der statischen Struktur der Spezifikation entscheiden. Selbst die betreffende konkrete Kommunikationsstruktur ist nach der Initialisierung statisch.

In unserer Beispielspezifikation ist, wie man leicht sieht, für die Transition der Modulinstanz `starter` keine der Bedingungen erfüllt, und es kommt zu dem beschriebenen Problem.

6 Ergebnisse

Die Erhaltung der Atomarität von Aktionen ist in nebenläufigen Systemen ein allgemeines und wichtiges Problem. In diesem Beitrag haben wir die Schwierigkeiten diskutiert, die entstehen, wenn atomare Estelle-Transitionen parallel implementiert werden sollen, und wir haben verschiedene Lösungsmöglichkeiten beleuchtet.

Eine Änderung der Semantik von Estelle scheint uns zur Behebung des geschilderten Problems mit der Atomarität von Transitionen nicht angebracht. Auch wenn bisher keines der existierenden Werkzeuge, die Implementationen erzeugen, die volle Semantik von Estelle exakt wahrt, ist dies auf die im vorigen Kapitel beschriebene Weise sehr wohl möglich; eine solche Implementation kann in den meisten Fällen sogar genauso effizient arbeiten wie bei den bisherigen Werkzeugen. Die im vorigen Kapitel angeführten Voraussetzungen für die Optimierung lassen sich großenteils leicht statisch, also zur Übersetzungszeit, abprüfen. Nur in den Fällen, wo wirklich Races drohen, wird ein aufwendigeres Protokoll notwendig.

Bei der Untersuchung der Atomarität im Kontext der nebenläufigen und der verteilten Systeme stellte sich heraus, daß der entscheidende Begriff in der Definition von Atomarität derjenige der *Beobachtbarkeit* von Zustandsänderungen ist, wobei diese Beobachtbarkeit in solchen Systemen oft, aber nicht immer, stark eingeschränkt ist.

Solange, bis sichergestellt ist, daß das jeweils verwendete Werkzeug die Semantik von Estelle genau einhält, hat ein Spezifizierer mit der im letzten Kapitel angeführten Liste Bedingungen in der Hand, um selbst zu entscheiden, ob bei seiner Spezifikation möglicherweise Schwierigkeiten auftreten werden. Darüberhinaus kann er durch Beachten dieser Liste je nach Problemstellung möglicherweise von vorneherein vermeiden, ein zumindest ineffizientes System zu spezifizieren. Nur wenn es die Semantik seiner Anwendung ausdrücklich erfordert, sollte er nicht-lokale, atomare Aktionen mit mehr als zwei Partnern spezifizieren, sonst sollte er sie besser explizit in mehrere Aktionen (d.h. in verschiedene Estelle-Transitionen) auflösen. Diese Erkenntnis ist im übrigen auch über den Kontext von Estelle hinaus gültig und kann bei jedem Entwurf eines nebenläufigen Systems hilfreich sein.

Literatur

[BrGo93] 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", Boston, Mass. (26.–29. Okt. 1993). North-Holland.

[Hog89] Hogrefe, D. *Estelle, LOTOS und SDL. Standard-Spezifikationssprachen für verteilte Systeme*. Springer (1989).

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

[ISO89] 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).

[KrGo93] Kreuz[3], D. und Gotzhein, R. *A compiler for the parallel execution of Estelle specifications*. In König, H. (Hrsg.), „Formale Methoden für Verteilte Systeme", FOKUS-Band 8, GI/ITG-Fachgespräch Juni 1992 (1993). Saur-Verlag.

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

[SiSt93] Sijelmassi, R. und Strausser, B. *The PET and DINGO tools for deriving distributed implementations from Estelle*. Comp. Networks and ISDN Systems **25**(7), 841–851 (Feb. 1993).

Anhang

```
specification problem_demo; (* by Jan Bredereke, Dec. 10, 1992 *)
  default common queue; (* Only one common input queue for all IPs *)
                        (* of a module instance. *)
  channel start_chan(sender, receiver);
    by sender: start;
  channel result_chan(sender, receiver);
    by sender: result; (* If you like, provide result parameters. *)
  module starter_m systemprocess;
    ip to_consumer: start_chan(sender);
       to_processor: start_chan(sender);
  end;
```

[3] geborener Peter

```
  body starter_b for starter_m; (* ... running on a VERY slow site. *)
    state ready, done;
    initialize to ready begin end;
    trans from ready to done
      begin
        output to_processor.start;  (* Both interactions SHOULD be ... *)
        output to_consumer.start;   (* ... sent atomically. *)
      end;
  end;
  module processor_m systemprocess;
    ip from_starter: start_chan(receiver);
       to_consumer: result_chan(sender);
  end;
  body processor_b for processor_m;
    trans when from_starter.start
      begin
        (* compute something here *)
        output to_consumer.result
      end;
  end;
  module consumer_m systemprocess;
    ip from_starter: start_chan(receiver);
       from_processor: result_chan(receiver);
  end;
  body consumer_b for consumer_m;
    state idle, waiting;
    initialize to idle begin end;
    trans from idle to waiting
      when from_starter.start   (* Notice that processing has started. *)
        begin end;
    trans from waiting to idle
      when from_processor.result
        begin end;(* (Only) after begin of processing, receive result. *)
  end;
  modvar (* of "specification problem_demo" *)
    starter: starter_m;  processor: processor_m;  consumer: consumer_m;
  initialize
    begin
      init starter with starter_b;
      init processor with processor_b;
      init consumer with consumer_b;
      connect starter.to_consumer to consumer.from_starter;
      connect starter.to_processor to processor.from_starter;
      connect processor.to_consumer to consumer.from_processor;
    end;
end.
```