ÜBER DAS WARTEN BEIM RECHNEN – SYNCHRONISATIONSSTRATEGIEN IN PARALLELEN UND INTERAKTIVEN SYSTEMEN

Lasse Scherffig, Georg Trogemann

Einleitung

In der Informatik werden unter dem Begriff Synchronisation (Herstellen von Gleichlauf, griech.: *sýn* – zusammen, *chrónos* – Zeit) Methoden und technische Vorrichtungen zusammengefasst, die der zeitlichen Abstimmung von Rechen- und Kommunikationsaktivitäten dienen. Synchronisation umfasst alle systeminternen Maßnahmen, die dafür sorgen, dass wichtige Ereignisse entweder gleichzeitig oder in einer bestimmten Reihenfolge stattfinden. Aus dieser allgemeinen Perspektive ist das Synchronisationsproblem der Informatik *systemimmanent*, d.h. ohne explizit beabsichtigt zu sein, folgt die Notwendigkeit zur Synchronisation von Rechenprozessen aus der Natur des Rechnens selbst und ist damit tief und untrennbar im zentralen Begriff des Algorithmus eingeschrieben. Unterschiedliche Präzisierungen des Algorithmenbegriffs (etwa: Turingmaschinen, -Kalkül, -rekursive Funktionen, unbeschränkte Registermaschinen) haben sich, trotz starker formaler Unterschiede, nicht nur in Bezug auf ihre Mächtigkeit als äquivalent erwiesen.

Es lassen sich eine Reihe gemeinsamer Charakteristiken aufzählen, die allen effektiven Verfahren gemein sind. Dazu gehören unter anderem Elementarität, Endlichkeit und Determiniertheit. Elementarität heißt lediglich, dass Algorithmen aus gewissen elementaren, nicht weiter zerlegbaren Grundoperationen aufgebaut sind. Endlichkeit verlangt, dass es nur eine begrenzte Anzahl unterschiedlicher Grundoperationen gibt, der Algorithmus sich als endliche Folge solcher Operationen beschreiben lässt und bei Ausführung nach endlich vielen Schritten anhält. Determiniertheit bedeutet, dass nach Durchführung einer elementaren Operation eindeutig bestimmt ist, welche als nächste auszuführen ist. Rechnen heißt also immer schon, elementare Aktionen in die richtige Reihenfolge zu bringen, sie folglich zu synchronisieren. Wesentlich an Algorithmen ist, dass die Reihenfolge der Operationen teilweise selbst Ergebnis des Rechnens ist. Verschiedene Kontrolloperationen (*if()*, *while()* etc.) dienen dazu, den

nächsten auszuführenden Schritt während des Ablaufs aus bereits berechneten Zwischenergebnissen zu bestimmen. Rechenprozesse berechnen also nicht nur Ergebnisse, sondern auch die Reihenfolge ihrer eigenen Berechnung. Um dieses dynamische In-Reihe-bringen auf technischer Ebene zu gewährleisten, müssen implizit oder explizit Mechanismen implementiert sein, die das korrekte Ineinandergreifen der Schritte realisieren. Auf diese Weise lässt sich bereits die Realisierung des Zehnerübertrags durch ausgefeilte Achsenkopplungen in mechanischen Rechenmaschinen als Lösung eines Synchronisationsproblems beschreiben. Die durchschlagende Synchronisationsmethode für elektronische Rechner besteht im Unterwerfen aller Signale unter einen zentralen Takt. Der zentrale Takt ist der Trick des Ingenieurs, die komplexen unterschiedlichen Signallaufzeiten in den Schaltkreisen so weit zu bändigen, dass eine zuverlässige und robuste diskrete Zustandsmaschine resultiert, in der strikte zeitliche Reihenfolgebedingungen für die eintreffenden Signale gewährleistet werden können.

Die Theorie der Berechenbarkeit kümmert sich um Operationsreihenfolgen und deren Korrektheit. Die Problematik der Ressourceneffizienz, d.h. die Frage, was mit vertretbarem Aufwand an Ressourcen wie Zeit und Speicherplatz usw. berechnet werden kann, wird dagegen in der Komplexitätstheorie und der Theorie paralleler und verteilter Systeme behandelt. Solche Effizienzüberlegungen, die darauf abzielen, Berechnungen möglichst schnell oder unter gemeinsamer Nutzung vorhandener Ressourcen durchzuführen, ziehen weitere Synchronisationsprobleme nach sich. Eine Reihe zugehöriger Synchronisationsmodelle und -verfahren wurden im Zusammenhang mit der frühen Betriebssystementwicklung ab den 1960er Jahren eingeführt. Die Rechnerressourcen wie Speicher und Ein-/Ausgabegeräte mussten vom Betriebssystem so verwaltet werden, dass keine Systemverklemmungen (sogenannte Deadlocks) auftreten. Deadlocks können dann entstehen, wenn Prozesse auf Betriebsmittel warten, die bereits von anderen Prozessen belegt werden und sich dadurch wechselseitig blockieren. Derartige Verklemmungen müssen entweder prinzipiell vermieden oder zumindest automatisch erkannt und beseitigt werden.

Bereits eine oberflächliche Betrachtung einfacher sequentieller Algorithmen zeigt, dass viele Teilberechnungen prinzipiell parallel ausgeführt werden könnten, da kausale Abhängigkeiten nur an wenigen Stellen der Berechnung bestehen. Die gleichzeitige, nebenläufige Ausführung von Teilberechnungen ist immer dann möglich, wenn Rechenschritte nicht auf Werte ihrer nebenläufigen Partner angewiesen sind. Das Forschungsgebiet der parallelen Algorithmen zielt darauf ab, das erzwungene strikte Nacheinander sequentieller Berechnungen aufzulösen und nur an jenen

Stellen Synchronisationsanforderungen einzuführen, wo es die Logik des Verfahrens zwingend verlangt. Parallele Algorithmen, die auf massiv parallelen oder verteilten Rechensystemen ausgeführt werden, können in vielen Fällen (z. B. Wettervorhersage, Strömungssimulation, Simulation chemischer Prozesse) erheblich beschleunigt werden. Der Ansatz zur Entwicklung paralleler Algorithmen kann aber auch sehr viel grundlegender verstanden werden. Wenn nicht mehr gefragt wird, auf welche Weise bereits bekannte sequentielle Algorithmen parallelisiert werden können, sondern wie asynchrones Rechnen als grundlegendes Prinzip definiert werden kann, führen die Fragestellungen und Lösungen zur Synchronisation über die klassische Algorithmentheorie hinaus.

Durch Fragen der Synchronisation ist die Informatik unter anderem gezwungen, Zeit als absolute Größe einzuführen und zu behandeln. Die Theorie der Berechenbarkeit kennt zwar elementare Rechenschritte und Schrittfolgen, also ein lineares Nacheinander, aber eigentlich keine explizite Zeit. Daraus leitet sich bereits eine grundlegende Eigenschaft informatischer Synchronisationsprinzipien ab. Für die Korrektheit von Rechenprozessen ist die Reihenfolge der einzelnen Schritte entscheidend, nicht die tatsächlich benötigte absolute Zeit. Solange Rechner in sich abgeschlossene Berechnungen durchführen und nicht mit ihrer Umwelt in Kontakt treten, sind logisch korrekte Zeitverhältnisse vollkommen ausreichend. Die logische Zeit setzt unterschiedliche Ereignisse in lineare Relation zueinander und baut im Wesentlichen auf nur drei Zeitgrößen auf: Ereignisse finden *vor*, *nach* oder *gleichzeitig* mit anderen Ereignissen statt. Dieses relationale Zeitmodell reicht nicht mehr aus, sobald der Benutzer ins Spiel kommt.[1] In interaktiven Systemen tritt die subjektive Zeit des Benutzers auf den Plan, die einen absoluten Zeittakt verlangt. User können im Gegensatz zu elektronischen Schaltungen nicht schneller oder langsamer ›getaktet‹ werden. Rechner und Nutzer sind vielmehr unter der Maßgabe des subjektiven menschlichen Zeitempfindens und der Fähigkeit des Nutzers, Informationen aufzunehmen und zu verarbeiten, zu synchronisieren.

1 Jeder kennt den Effekt, wenn ältere Animationssoftware, die nicht mit der äußeren absoluten Zeit synchronisiert ist, sondern nur die Korrektheit des Ablaufs gewährleistet, auf einem neuen Computer gestartet wird. Die Bildfolgen werden zwar weiterhin absolut korrekt berechnet und dargestellt, aber aufgrund der Leistungssteigerung neuer Prozessoren zeitlich oft so stark gestaucht, dass das Bildgeschehen viel zu schnell abläuft und mitunter überhaupt nicht mehr verfolgbar ist.

In erster Näherung kann Synchronisation in der Informatik als Technologie des korrekten und effizienten Wartens aufgefasst werden. Es handelt sich um eine lose Methodensammlung, die sich um Fragen der Korrektheit von Berechnungsabläufen unter Maßgabe ihrer Effizienz und der Zufriedenheit des Benutzers bemüht. Der vorliegende Beitrag beschreibt verschiedene Synchronisationsansätze der Informatik exemplarisch aus vier Perspektiven: 1. der physischen Realisierung von Synchronisationsmechanismen in Hardware, 2. der Synchronisation paralleler Prozesse auf der Ebene der Software, 3. formale Modelle der Synchronisation, 4. aus der User-Perspektive, die sowohl die Maschine als auch ihre Umwelt unter dem Stichwort der Interaktion in den Blick nimmt.

Hardware

Charles Babbage: Anticipating-Carry

Zu einem Zeitpunkt, als die Präzisierungen des Algorithmenbegriffs in ihrer heutigen Form noch nicht vorlagen, stand die Maschinisierung algorithmischen Rechnens bereits an der Schwelle ihrer Realisierbarkeit. In der Geschichte der Informatik nimmt die *Analytical Engine* von Charles Babbage eine besondere Stellung ein. Diese Maschine wurde zwar nie fertig gestellt, wäre aber die erste programmierbare und Turing-vollständige Maschine gewesen. Die Entwürfe lagen 1838 weitgehend vor.[2]

Die Analytical Engine ist ein mechanischer Apparat, der ausschließlich mit Zahnrädern, Achsen und Hebeln arbeitet. Grundelemente des Rechnens sind senkrechte Achsen, auf denen Zahnräder mit je zehn Stellungen angeordnet sind. Jede dieser Stellungen entspricht einer von zehn möglichen Ziffern. Das Rechnen mit der Analytical Engine basiert also, anders als das Rechnen mit modernen Computern, auf Dezimalzahlen. Da jede Achse vierzig Zahnräder umfasst, entspricht jedes Zahnrad einer Ziffer einer vierzigstelligen Zahl. In der Terminologie der modernen Informatik umfasst damit die Wortlänge der Analytical Engine, die Grundgröße eines ihrer Register, vierzig Dezimalstellen (was etwa 133 bit entspricht). Das Lesen und Schreiben von Zahlen erfolgt durch Drehen der Achsen. Gelesen werden Zahlen durch eine Drehung ihrer Achse um 360 Grad. Ein Hebel gibt dabei den Zahlenwert jeder Ziffer als diskrete Zahl von Drehschritten an ein anderes Zahnrad weiter. Dieses Verfahren löscht die Zahl im Moment des Auslesens (*destructive readout*).

In diesem komplexen mechanischen System ist die Addition zweier Ziffern einfach zu realisieren: Das Zahnrad, das einen der Summanden darstellt, wird so gedreht, dass sein Zahlenwert auf ein anderes Zahnrad übergeht. Nach der Addition sind beide Summanden verloren und nur ihre Summe erhalten. Anders als in einer sequentiellen Addition ermöglicht die Anordnung aller vierzig Stellen einer Zahl auf einer gemeinsamen Achse das Addieren aller Stellen einer Zahl zur gleichen Zeit. Die Analytical Engine rechnet auf Wortebene parallel. Diese parallele Verarbeitung wirft aber Probleme auf, sobald es beim Drehen einer Ziffer von 9 auf 0 zu einem Übertrag (*carry*) kommt.

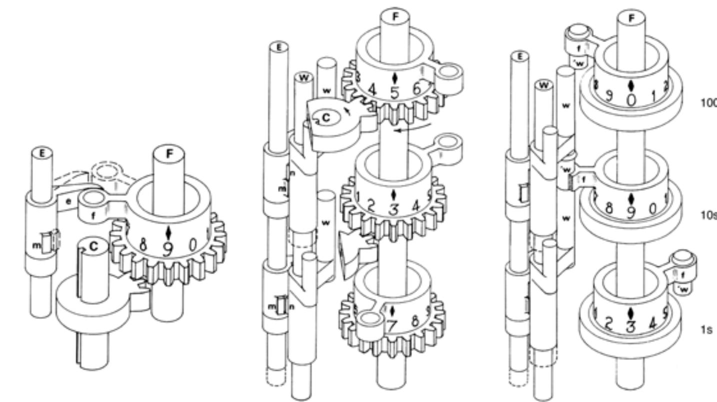

Abb. 1 - Carry und Anticipating-Carry in der Analytical Engine.

Um die Korrektheit der Berechnung in einem solchen Fall zu garantieren, bedarf es eines Mechanismus', der durch den Übertrag ausgelöst wird und ihn auf die – buchstäblich – nächsthöhere Ebene weiterleitet. Diesen realisiert Babbage unter Verwendung dreier weiterer Achsen. Auslöser des Übertrags ist dabei ein Hebel, der genau dann, wenn ein Zahnrad von 9 auf 0 übergeht, seine Drehung an eine benachbarte Achse weitergibt und damit eine *carry-warning* generiert. Nach der Drehung der Ziffernachse für die Addition wird nun die Achse der *carry-warnings* angehoben. Diese hebt an genau den Stellen, an denen eine *carry-warning* vorliegt eine weitere Achse an, an der sich nach oben gerichtete *fingers* befinden. Die *finger*s heben schließlich Zahnräder an, welche den Übertrag, wenn sie gedreht werden an die nächsthöhere Ebene weiterreichen.

2 Dazu und zum Folgenden siehe Bromley 1998.

Ist das Problem des Übertrags und damit das der Korrektheit der parallelen Addition auf diese Weise gelöst, entsteht jedoch ein neues Problem: Jeder Übertrag kann einen weiteren nach sich ziehen. Generiert das Einer-Rad beispielsweise einen Übertrag während das Zehner-Rad auf 9 steht, so wird nach diesem *carry* ein neuer *carry* – von der Ebene der Zehner auf die der Hunderter – erforderlich. Im schlimmsten Fall, dem *worst case*, von dem man bei der Implementierung von Algorithmen auszugehen hat, kann also ein *carry* bis zu 39 weitere Überträge erforderlich machen, was bei großen Wortlängen sehr ineffizient ist.

Babbage nennt seine Lösung für dieses Problem *anticipating-carry*. Er bringt hierfür an der Achse, die die *fingers* der Überträge trägt, einen weiteren *finger* (den *fixed wire*) an. Gleichzeitig erhält die Achse mit den Ziffern einen Hebel, an dem sich eine kurze Teilachse (der *movable wire*) befindet. Diese ist so positioniert, dass der *movable wire* genau dann, wenn das Rad auf 9 steht, mit dem *fixed wire* eine vollständige Achse bildet. *Fixed* und *movable wires* greifen so ineinander und leiten einen von unten kommenden *carry* sofort nach oben weiter. Ein *carry* wird so durch alle Ebenen, die eine 9 enthalten ohne Verzögerung so weit nach oben propagiert, wie Zahnräder über ihm auf der 9 stehen (diesen Vorgang nennt Babbage *completing a chain*).

Carry und *anticipating-carry* sind mechanisch so aufwändig, dass Babbage beschließt, Speicher und Arithmetik seiner Maschine strikt zu trennen. Das Speichern von Zahlen erfordert nur eine einzige Achse pro Zahl und erfolgt im *Store* der Maschine, während Berechnungen viele Achsen benötigen und daher in einem eigenen Teil der Analytical Engine, der *Mill*, stattfinden. Babbage nimmt mit der Trennung von *Store* und *Mill* eine wichtige Eigenschaft der Von-Neumann-Architektur um hundert Jahre vorweg: die Trennung von Rechenwerk und Speicherwerk.[3]

Für die parallele Addition in der Analytical Engine konstruiert Babbage einen Mechanismus zur Synchronisation paralleler Addition auf Wortebene,[4] der korrekte Ergebnisse bei einer hohen Zeitersparnis garantiert. *Anticipating-carry* basiert dabei auf einem Kanal, der während der Berechnung Kommunikation zwischen den Zahnrädern auf einer Achse und damit den Ziffern einer Zahl ermöglicht.

Architekturen parallelen Rechnens

Wie das historische Beispiel der *Analytical Engine* zeigt, ist paralleles Rechnen zwar schon von Anbeginn zentraler Gegenstand der Rechentechnik, im 20. Jahrhundert

dominieren aber für lange Zeit sequentiell arbeitende Maschinen und Architekturen. Schon John von Neumann hatte darauf hingewiesen, dass das Zeitverhalten der Elemente eines Computers asynchron, durch ihre jeweiligen Reaktionszeiten, oder synchron, durch einen zentralen Taktgeber, bestimmt sein kann. Eine synchrone Architektur erschien zur Zeit des EDVAC-Entwurfs vorteilhaft[5] und hat sich bis heute als vorrangige Technologie behauptet – obwohl von Neumann bis 1951 an der Entwicklung der asynchronen IAS-Maschine beteiligt war. Deren arithmetische Einheit wurde nicht nur unter Verzicht auf einen zentralen Taktgeber gebaut, sie arbeitete auch, wie die Analytical Engine, auf Wortebene parallel, wobei ihre Wortlänge ebenfalls 40 (allerdings binäre) Ziffern umfasste.[6] Auf dem Design der IAS-Maschine basierte unter anderem der ILLIAC II, der als erster vollständig asynchroner Rechner gilt.

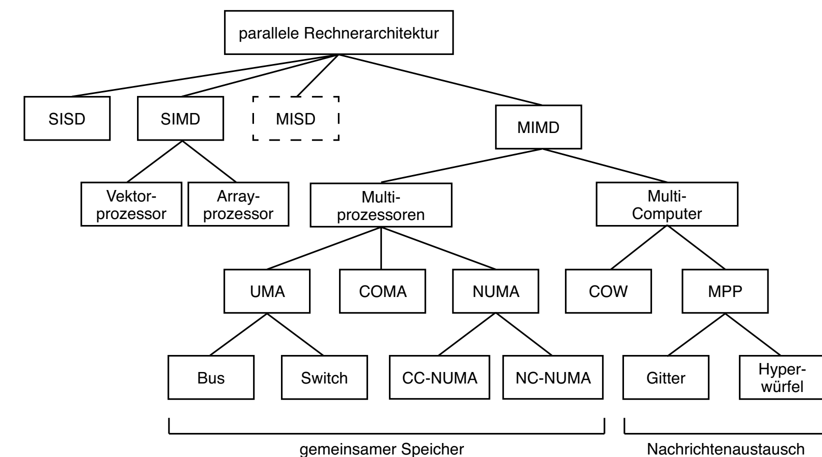

Abb. 2 - Klassifikation paralleler Rechnerarchitekturen.

Generell gilt, dass Strategien parallelen Rechnens auf unterschiedlichen Ebenen verwirklicht werden können, wobei die beschriebene Parallelisierung auf Wortbasis zu den ›feinkörnigen‹ Ansätzen zu zählen ist. In den zurückliegenden Jahrzehnten wurde

3 Als Central Arithmetical Part und Memory in: Neumann 1945.

4 Übrigens nicht als Erster: »It is interesting to note that Hoarding Carriage is seen in the little machine of Sir Samuel Moreland invented in 1666, and probably existed in that of Pascal still earlier.« Babbage 1888.

5 Neumann 1945: 5 f.

6 Bigelow 1980.

eine Vielzahl neuartiger Architekturen entwickelt, die sehr unterschiedliche Formen der Parallelität auf allen Ebenen der Hardware nutzen, von der feinsten Körnung des Rechnens auf Wortebene bis hin zu sehr groben funktionalen Einheiten, etwa gleichzeitig laufenden, unabhängigen Programmen. Zur Klassifikation der unterschiedlichen Rechnerarchitekturen hat Flynn bereits 1972 eines der ersten und bis heute gebräuchlichen Klassifizierungsschemata eingeführt.[7] Er charakterisiert Rechner als Operatoren auf Befehls- und Datenströmen und erhält auf diese Weise vier Klassen von Rechnern, die er SISD, SIMD, MISD und MIMD nennt. SISD-Rechner (Single Instruction Stream – Single Data Stream) entsprechen der klassischen Von-Neumann-Architektur. Als typische Vertreter der SIMD-Klasse (Single Instruction Stream – Multiple Data Stream), bei denen ein einziger Befehl gleichzeitig auf vielen Datenelementen ausgeführt wird, gelten Feldrechner und systolische Arrays. SIMD-Systeme arbeiten besonders dann effizient, wenn große Mengen von Vektoren oder Matrizen zu verarbeiten sind, wie das üblicherweise bei der Lösung großer Differentialgleichungssysteme notwendig ist. MIMD-Systeme (Multiple Instruction Stream – Multiple Data Stream) erlauben die gleichzeitige Ausführung unterschiedlicher Befehle auf jeweils unterschiedlichen Datenelementen. Die Interpretation der MISD-Klasse ist in der Literatur umstritten. Während einige Autoren verschiedene Arten von Pipeline-Rechnern dieser Klasse zuordnen, halten andere die Klasse nicht für sinnvoll und beschränken sich auf SISD-, SIMD- und MIMD-Systeme.

In Abbildung 2 ist eine Verfeinerung der Flynn'schen Klassifikation nach Tanenbaum dargestellt. Von besonderem Interesse ist hierbei die Unterteilung der MIMD-Systeme in eng gekoppelte *Shared-Memory-Architekturen*, oft auch *Multiprozessoren* genannt, und lose gekoppelte *Message-Passing-Architekturen*, die auch als *Multicomputer* oder *Verteilte Systeme* bezeichnet werden. Während Multiprozessoren sehr eng über gemeinsame Speicher miteinander verbunden sind und auf diese Weise auf gemeinsame Daten zugreifen können, sind Multicomputer nur lose gekoppelt und kommunizieren über Nachrichten.

Durch den Zusammenschluss preiswerter PCs lassen sich heute prinzipiell beliebig skalierbare Rechenleistungen kostengünstig realisieren. Trotzdem sind Synchronisationsfragen auf niedrigen Hardware-Ebenen weiterhin aktuelles Forschungsthema. So bringt das Unterwerfen aller Teile eines Microchips unter einen zentralen Takt unweigerlich mit sich, dass viel Energie in die synchrone Verbreitung des Takts fließt, auch dann, wenn die getakteten Elemente eines Chips eigentlich nichts zu berechnen haben. Heute verspricht das Design asynchroner Chips daher in erster Linie

Energieersparnis, weshalb es vor allem in eingebetteten Systemen eine Rolle spielt. Besonders spezielle digitale Schaltungen, die festgelegte Aufgaben haben, wie beispielsweise Signalverarbeitungsfilter für Videodaten, lassen sich asynchron konstruieren. Durch *local handshakes* lässt sich hier sicherstellen, dass die Schaltelemente eines Chips genau dann Daten bearbeiten, wenn die ihnen vorgelagerten Elemente mit der Datenverarbeitung fertig sind. Synchronisation erfolgt hier also nicht durch die ständige Kommunikation eines globalen Taktgebers mit allen Teilen eines Chips, sondern durch lokale Absprachen, die gewährleisten, dass nicht benötigte Elemente auf Daten warten, ohne Energie zu verbrauchen.

Software

Semaphoren

Immer wenn verschiedene Algorithmen gemeinsame Variablen, Speicherbereiche oder Peripheriegeräte – wie etwa Drucker – verwenden, werden Verfahren benötigt, die die effiziente Verteilung von Ressourcen regeln und andererseits die Konsistenz der verarbeiteten Daten garantieren. Der niederländische Informatiker Edsger Dijkstra hat maßgeblich dazu beigetragen, diese Probleme in der frühen Betriebssystementwicklung zu lösen. Dijkstra ist dabei ein radikaler Vertreter einer Richtung, für die es in der Informatik vor allem um den formalen Nachweis der Korrektheit von Algorithmen geht. Paralleles Rechnen zeichnet sich für ihn daher vor allem dadurch aus, dass es die Determiniertheit algorithmischer Abläufe zu untergraben droht, da es nicht in der stringenten Reihenfolge sequenzieller Abläufe vonstattengeht.[8] Seine Antwort auf die Herausforderungen von Parallelität besteht daher in der Entwicklung von Algorithmen, die die Korrektheit von Berechnungen in nebenläufigen Prozessen bei hoher Ressourceneffizienz garantieren.

Dijkstra illustriert das Problem der Verteilung gemeinsam genutzter Ressourcen durch das »Dining Philosophers Problem«, das mittlerweile als kanonisch gilt:[9] Fünf Philosophen sitzen um einen Tisch. Auf dem Tisch befinden sich fünf Teller und

7 Flynn 1972.
8 Dijkstra 1997.
9 Ebd.

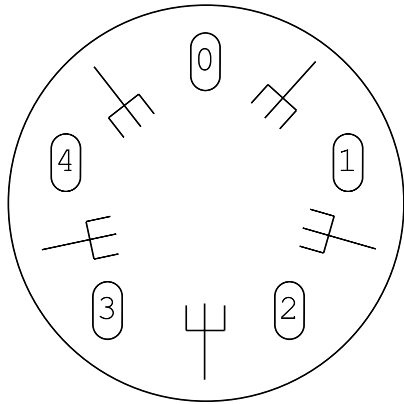

Abb. 3 - Das Problem der essenden Philosophen in extrem reduzierter Darstellung durch Dijkstra.

fünf Gabeln. Es gibt, so Dijkstra, ein »kompliziertes Nudelgericht.« Um dieses essen zu können, braucht ein Philosoph zwei Gabeln, eine links und eine rechts. Außer mit dem Essen sind Dijkstras Philosophen ausschließlich mit Denken beschäftigt. Jeder Philosoph lässt sich daher als sequentielles Programm sehen: Ein Philosoph denkt, nimmt eine Gabel, nimmt die andere Gabel, isst und legt die Gabeln nacheinander wieder ab. Dies alles in einer Schleife ewiger Wiederholung oder mit Dijkstras Worten, sowohl in natürlicher Sprache als auch mit Pseudo-Code formuliert:

The Life of a philosopher consists of an alternation of thinking and eating:

```
cycle begin think;
eat;
end
```

In diesem Szenario identifiziert Dijkstra zwei mögliche Probleme: *deadlock* und *livelock*. Greift ein Philosoph beispielsweise zur linken Gabel, während die rechte von seinem Nachbarn verwendet wird und wartet er daraufhin, bis die rechte frei wird, kann sich eine Verklemmung ergeben. Dies geschieht, wenn sein Nachbar wiederum auf die Gabel rechts von ihm warten muss und sich diese Kette rund um den Tisch fortsetzt: Kein Philosoph kann die zweite Gabel nehmen, da diese von einem wartenden Nachbarn benutzt wird. Damit wird auch kein Philosoph zu essen anfangen, was die Voraussetzung dafür wäre, dass die belegten Gabeln wieder frei würden. Die Philosophen verhungern. Das Eintreten eines *livelock* ist zwar unwahrscheinlicher, in der Welt diskreter Laufzeit aber dennoch möglich: Hierzu kommt es, wenn man *deadlocks* zu verhindern versucht, indem man die Philosophen anweist, eine einmal aufgenommene Gabel wieder abzulegen, wenn die zweite nicht verfügbar ist. Im *worst case*, in dem alle Philosophen zum exakt gleichen Zeitpunkt ihre Gabel aufnehmen, führt dies zu einer unendlichen Abfolge aus Aufnehmen und Ablegen aller Gabeln. Wieder verhungern die Philosophen.

Dijkstra formalisiert das Problem und definiert das Leben der Philosophen als Algorithmus. Innerhalb dieses formalen Systems gibt er eine Lösung an, deren Korrektheit mit formalen Methoden nachgewiesen werden kann. Diese Lösung ist berühmt geworden, da sie auf das Konzept des *Semaphor* zurückgreift.

Der Begriff Semaphor bezeichnet ursprünglich Signalmasten der optischen Telegrafie und mechanische Eisenbahnsignale. Dijkstra bedient sich des Wortes und verwendet es für eine spezielle Datenstruktur. Diese besteht aus einer ganzzahligen Variable und zwei möglichen Operationen: P und V – Initialen des niederländischen *verhoog* (erhöhe) und des Kunstwortes *prolaag*, gebildet aus *probeer te verlaagen* (versuche zu senken). Damit bleiben Semaphoren im Bild des Eisenbahnsignals, das gehoben und damit offen oder gesenkt und damit geschlossen sein kann. Anders als beim Eisenbahnsignal können hier allerdings mehrere Erhöhungen in Folge vorgenommen werden: Der Semaphor ist ein Zähler. Ist dessen Zahl größer als null, so ist eine Ressource verfügbar (ein Signal geöffnet, ein Gleis frei). Steht der Zähler auf null, scheitert der Versuch zu senken. Der für diesen Versuch verantwortliche Prozess muss warten, bis der Zähler wieder erhöht wurde. Nach dem Senken des Zählers muss der dafür verantwortliche Prozess seinerseits diesen wieder erhöhen und die Ressource freigeben.

Probeer te verlaagen kann dabei auf zwei Arten geschehen: Entweder ein Prozess versucht immer wieder den Semaphor zu senken, bis das möglich ist – was *busy waiting* (oder aktives Warten) genannt wird und ineffizient ist, da möglicherweise viele Ressourcen für gescheiterte Versuche verwendet werden. Oder er wird angehalten und wartet still bis der Zähler erhöht und er darüber benachrichtigt wurde. Dies setzt allerdings voraus, dass Prozesse von außen angehalten und gestartet werden können, eine Technik, die in der Hardware eines Computers realisiert sein muss und die weiter unten als *Interrupt* noch näher besprochen wird.

Um die Konsistenz eines Semaphors zu garantieren, muss der Zugriff auf P und V so geschützt sein, dass nicht etwa ein Prozess den Zähler senken kann, während ein anderer ihn erhöht. Das Unterbrechen von Prozessen, mit dem auch eine Unterbrechung während der Ausführung von P und V möglich wird, verlangt daher wieder nach Schutzmechanismen, die die Unterbrechung bestimmter Abläufe unmöglich machen. Diese Mechanismen müssen, wie die Möglichkeit der Unterbrechung selber,

auf Hardware-Ebene angesiedelt sein. Die Betrachtung von Synchronisation auf der Ebene von Software kommt also doch nicht ganz ohne die Ebene ihrer technischen Realisierung in Hardware aus.
Um das Problem der essenden Philosophen zu lösen, braucht Dijkstra einen Semaphor pro Philosoph und einen globalen für den Tisch. Das Leben eines Philosophen wird damit ungleich komplizierter: Es besteht nun darin, in einer Schleife ewiger Wiederholung zu denken, nur dann zu essen, wenn kein Nachbar isst und nach dem Essen die wartenden Nachbarn zu benachrichtigen, damit sie zu essen beginnen können.[10]

Logische Zeit

Anders als beispielsweise in zentral getakteten SIMD-Rechnern, ist es in verteilten Systemen, die über Nachrichten kommunizieren, nicht unbedingt möglich zu sagen, ob ein Ereignis vor oder nach einem anderen stattgefunden hat.[11] Die für ein erfolgreiches In-Reihe-bringen elementarer Operationen notwendige Aussage über ein *happens before*, ohne auf eine globale Uhr zu verweisen, wird damit zu einem entscheidenden Synchronisationsproblem. Eine Lösung dieses Problems besteht darin, die logische Zeit der Rechenschritte explizit zu machen. Jedem Prozess eines verteilten Systems wird dazu eine logische Uhr zugewiesen, die wie ein Zähler ihren Wert nach jedem Rechenschritt erhöht. Ein Vorher und Nachher zwischen verschiedenen Prozessen lässt sich nun erzeugen, wenn die Kommunikation zwischen zwei Prozessen mittels Nachrichten geschieht, die den Zeitpunkt, das heißt den Zählerstand ihrer logischen Uhr, im Moment ihres Abschickens kennen. Erhält ein Prozess nun eine Nachricht, so kennt er den logischen Zeitpunkt ihres Absendens und kann seine eigene logische Uhr damit in Einklang bringen, indem er sie beispielsweise auf einen Zeitpunkt vorstellt, der sowohl nach dem aktuellen Stand seiner logischen Uhr als auch nach dem der Nachricht liegt. Dieses Verfahren, bekannt als Lamport-Algorithmus, ordnet alle Abläufe eines verteilten Systems in ein Vorher und Nachher. Will man das Vorstellen der logischen Uhren verhindern, kann ein Prozess, erhält er eine Nachricht aus der Zukunft, auch *warten*, bis seine logische Uhr die logische Zeit der Nachricht erreicht und sie erst dann bearbeiten.

Formale Modelle der Synchronisation

Petri-Netze

Synchronisation in verteilten Systemen setzt – wie wir oben gesehen haben – voraus, dass Prozesse über Nachrichten miteinander kommunizieren können. Diese Erkenntnis bildet den Hintergrund einer abstrakten und dennoch intuitiven Beschreibung von Prozessen und ihrer Synchronisation, die in der theoretischen Informatik seit 1962 unter dem Namen Petri-Netze bekannt wurde. Petri-Netze sind mathematische Modelle für nebenläufige Systeme, die formale Beweise zur Verklemmungsfreiheit erlauben. Carl Adam Petri entwarf dieses Konzept vor dem Hintergrund der Erkenntnis, dass der Gleichlauf zweier Uhren nur durch Kommunikation zwischen ihnen erreichbar ist.[12]

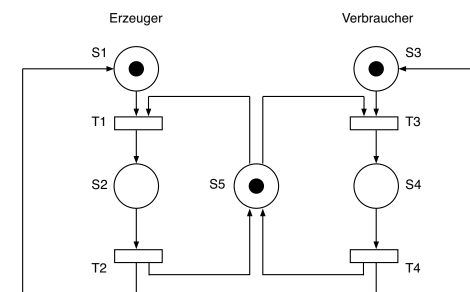

Abb. 4 - Petrinetz-Beispiel für das klassische Erzeuger-Verbraucher-Problem. Das Problem beschreibt auf abstrakte Weise die Regelung der Zugriffsreihenfolge durch einen Erzeuger- und einen Verbraucher-Prozess auf gemeinsame Elemente.

In der Tradition der theoretischen Informatik stehend sind Petri-Netze Automaten. Sie erweitern die grundlegende Theorie endlicher Automaten um die Fähigkeit, nebenläufige Prozesse darzustellen. Der entscheidende Vorteil ist, dass die Verklemmungsgefahr oder auch prinzipielle Verklemmungsfreiheit nebenläufiger Prozesse, sobald sie als Petri-Netze modelliert sind, formal bewiesen werden kann. Wie endliche Automaten lassen sich Petri-Netze als Graphen darstellen. Die Knoten eines Petri-Netzes wer-

10 Dijkstras schon recht komplizierte Lösung ist, wie er anmerkt, dabei noch immer nicht gegen alle Unmöglichkeiten gesichert: Verschwören sich zwei Philosophen, kann der zwischen ihnen weiterhin verhungern. A. a. O.: 24.
11 Hierzu und zum Folgenden vergleiche Lamport 1978.
12 Petri 1962.

den Stellen oder Plätze genannt. Sie entsprechen den elementaren Grundoperationen des abgebildeten Algorithmus. Ihre Kanten, Transitionen genannt, regeln die Abfolge dieser Operationen. Zusätzlich zu Stellen und Transitionen verfügen Petri-Netze über Marken, die angeben, an welcher Stelle eines durch ein Petri-Netz modellierten Ablaufs sich ein Prozess gerade befindet. Transitionen regeln damit den Übergang einer Marke von einer Stelle zur nächsten. Dabei bilden die Stellen unmittelbar vor einer Stelle deren Vorbereich, während die Stellen danach ihren Nachbereich bilden. Nebenläufigkeit wird hier durch das Vorhandensein mehrerer Marken in einem Graphen abgebildet.

Synchronisation von Nebenläufigkeit wird genau dann möglich, wenn Abläufe über gemeinsame Stellen verfügen. Hier spricht man von Verzweigung (*fork*), wenn die Anzahl der Stellen im Nachbereich einer Stelle größer als eins ist, ist sie im Vorbereich größer als eins, spricht man von Zusammenführung (*join*). Eine Transition kann genau dann schalten und eine Marke weiterreichen, wenn sich in allen Stellen in ihrem Vorbereich eine Marke befindet. Liegt mehr als eine Stelle im Vorbereich, wird so lange gewartet, bis alle Stellen belegt sind. Nach dem Schalten (*Feuern*) der Transition liegt in allen Stellen ihres Nachbereichs eine Marke. An jeder Zusammenführung warten Prozesse, die nicht synchron ablaufen, also aufeinander. So entsteht aus dem wiederholten Zusammenführen und Verzweigen zweier Prozesse über gemeinsame Stellen Synchronisation.

Semaphoren können in Petri-Netzen als exklusive Marken dargestellt werden, die an einer Stelle liegen, die sowohl zum Vorbereich als auch zum Nachbereich eines geschützten Abschnitts des Netzes (wie etwa dem Abschnitt, der das Aufnehmen einer Gabel beschreibt) gehören. Kommt ein Prozess in Form einer zweiten Marke hinzu, kommt es zum *join* und es wird in den geschützten Abschnitt geschaltet. Die exklusive Marke fehlt nun, bis sie am Ende des Abschnitts durch einen *fork* wieder an ihren Platz zurückkehrt.

Synchronisation bedeutet für die theoretische wie für die angewandte Informatik also vor allem die Koordination verschiedener sequentieller Abläufe. In den verwendeten Modellen muss eindeutig zu klären sein, wann die Elemente eines Ablaufes relativ zu einem anderen stattfinden. Voraussetzung dafür ist Kommunikation, etwa das Versenden von Nachrichten oder die Verwendung gemeinsamer Knoten eines Graphen – was sich als äquivalent erweist. Innerhalb der Formalisierungen des Synchronisationsproblems können Effizienz und Korrektheit von Berechnungen garantiert werden. Zeitersparnis unter dem Druck garantierter Korrektheit, Konsistenz und

dem Ausschluss möglicher *dead-* und *livelocks* basiert dabei im Wesentlichen auf der grundsätzlichen Bereitschaft, wann immer es zu Kommunikation kommt, aufeinander zu warten.

Das Gesetz von Amdahl

Die Erkenntnis, dass sich viele Teilberechnungen sequentieller Algorithmen teilweise parallelisieren lassen, könnte Anlass zu der Hoffnung sein, dass sich durch eine genügend hohe Zahl gleichzeitig rechnender Prozessoren die Laufzeit von Algorithmen beliebig verringern lässt. Dies ist heute wieder eine sehr aktuelle Idee, seit das Ende der Taktratensteigerung – zumindest auf der Basis heutiger Elektronik-Chips – erreicht scheint und deshalb nicht mehr nur Großrechner über mehr als einen Prozessorkern verfügen. Bereits 1967 konnte der Computerarchitekt Gene Amdahl aber allgemein zeigen, dass die mögliche Zeitersparnis durch paralleles Rechnen in der Praxis obere Schranken kennt. Das hat im Wesentlichen zwei Gründe: Der eine besteht darin, dass Algorithmen immer einen seriellen Anteil besitzen, einen Anteil also, der nicht parallel gerechnet werden kann. Der zweite besteht darin, dass die Aktivitäten paralleler Abläufe – wie wir bei Babbage gesehen haben – synchronisiert werden müssen, um ihre Korrektheit zu garantieren, was zusätzlichen Rechenaufwand bedeutet. Diese Überlegungen Amdahls wurden im Folgenden (und nicht von Amdahl selber) in eine Formel überführt.[13]

$$s - \frac{1}{a + o(P) + \frac{1-a}{P}} \leq \frac{1}{a}$$

Diese gibt den *speedup* s als Funktion von *P*, der Anzahl der verwendeten Prozessoren, und *a*, des sequentiellen, also nicht parallelisierbaren Anteils eines Algorithmus', an. Der Synchronisationsaufwand zwischen den einzelnen Prozessoren hängt von ihrer Anzahl ab, was mit *o(P)* als linearer Zusammenhang in der Landau-Notation für Laufzeitabschätzungen in die Formel eingeht.

Der rechte Teil dieser Ungleichung macht dabei deutlich, dass der maximale *speedup* ausschließlich vom sequentiellen Anteil eines Algorithmus' abhängt. Die

13 Vergleiche dazu die Anmerkungen von Guihai Chen in Amdahl 1967: 4.

Erkenntnis ist fundamental: Hat beispielsweise ein Algorithmus einen nicht reduzierbaren sequentiellen Anteil von nur 1 Prozent, beträgt der maximale Laufzeitgewinn *s=100*, auch dann, wenn Millionen von Prozessoren für die parallelisierbaren 99 Prozent eingesetzt werden und damit die Rechenzeit für diesen Hauptteil des Algorithmus' schließlich gegen Null geht. Abgesehen von den Problemen, die Korrektheit paralleler Berechnungen zu garantieren, gibt es also eine grundsätzliche Beschränkung für die Zeitersparnis und damit den Gewinn an Effizienz, die nicht von der Anzahl der verwendeten Prozessoren abhängt.

Gerichtete Graphen

Das Gesetz von Amdahl liefert nur eine obere Schranke für den Leistungsgewinn durch parallele Bearbeitung einer Aufgabe. Für die detaillierte Untersuchung spezieller Parallelisierungsprobleme wurden Modelle entwickelt, die weitaus präzisere Abschätzungen erlauben. Eine naheliegende Methode ist die Erweiterung der klassischen Petri-Netze zu zeitbehafteten Petri-Netzen, in denen die Transitionen zu zeitverbrauchenden Einheiten erweitert werden. Eine in Bezug auf das Zeitverhalten jedoch intuitivere Vorgehensweise ist die Modellierung paralleler Programme als Präzedenzgraphen.

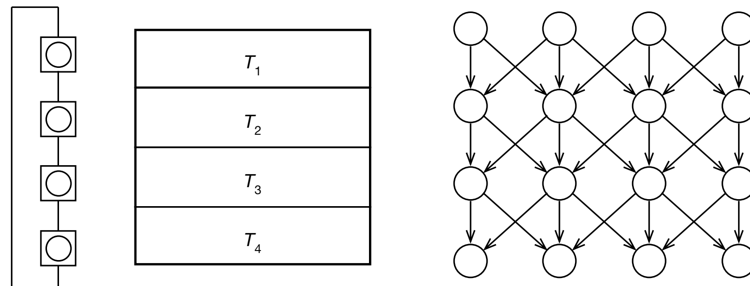

Abb. 5 - Ein Variablenfeld wird in vier Bereiche partitioniert, die parallel von einer korrespondierenden Prozessorring-Architektur gerechnet werden (linkes Bild). Der Graph rechts zeigt den resultierenden Präzendenzgraphen, wenn jeder Prozessor auf die Fertigstellung der Iterationsschritte seiner beiden unmittelbar angrenzenden Nachbarn wartet.

Eine häufige Anwendung in der Parallelverarbeitung ist das verteilte Berechnen von Funktionen über einem n-dimensionalen Datenfeld. Hier wird jedem Prozessor ein Teil des Feldes zugeordnet, dessen Werte nun iterativ berechnet werden. Die zu berechnende Gleichung, beispielsweise eine partielle Differentialgleichung, benötigt an den Grenzen der Datenfelder dabei normalerweise Werte ihrer Nachbarn. Jedes Feld ist daher ständig auf die Ergebnisse der Berechnungen angrenzender Felder angewiesen. Bevor eine neue Iteration gestartet werden kann, müssen die direkten Nachbarn ihren Iterationszyklus ebenfalls beendet haben.

Um genaue Voraussagen über das Laufzeitverhalten solcher Berechnungen zu machen, kann man ihr Verhalten abstrakt modellieren. Ein Präzendenzgraph[14] dient dabei als Modell des Verhaltens eines solchen Systems über die Zeit. Während die Knoten dieser Graphen die unabhängigen (und seriell ausgeführten) Berechnungen für die Teile des Feldes modellieren, bilden seine Kanten die Synchronisationsanforderungen zwischen den einzelnen Teilen ab. Jedem einzelnen Knoten kann dabei eine stochastische Verteilung zugewiesen werden, die das Laufzeitverhalten der sequentiellen Berechnungseinheiten statistisch beschreibt. Durch die Synchronisationsanforderungen in den Kanten entwickelt das Gesamtsystem ein Laufzeitverhalten, das durch die Notwendigkeit, immer wieder auf Ergebnisse aus benachbarten Knoten zu warten, weitaus komplexer als die einzelnen Verteilungen sein kann. Dieses Laufzeitverhalten entfaltet sich über die einzelnen Ebenen des Graphen, wobei jede Ebene das Voranschreiten der Berechnungen zwischen zwei erforderlichen Synchronisationen beschreibt.

Im Falle regulärer Graphen (Beispiel Abbildung 5) zeigt das Zeitverhalten des Gesamtsystems eine interessante Eigenschaft: Obwohl die Berarbeitungszeit der einzelnen Teilaufgaben im Einzelfall stark schwanken kann, zeigt das statistische Verhalten insgesamt bereits nach wenigen Iterationen eine große Regelmäßigkeit. Die Wartezeiten zwischen den einzelnen Iterationsschritten werden – in der Terminologie stochastischer Prozesse – *stationär*, d.h. Mittelwert und Varianz der Gesamtlaufzeitverteilung konvergieren sehr schnell gegen einen festen Wert. Es stellt sich gewissermaßen eine zeitliche Regelmäßigkeit höherer Ordnung ein. Die einzeln beobachtete Synchronisationswartezeit nach einem Iterationsschritt wird nach wie vor stark schwanken, die Wahrscheinlichkeiten des Wartens nähern sich einer Konstanten.

14 Fleischmann 1990.

Interaktion: Synchronisation von Rechner und Umwelt

Whirlwind

Da viele, wenn auch nicht alle Rechner der 1960er Jahre nicht parallel sondern sequentiell arbeiten, entspringen die Arbeiten von Dijkstra und Petri, anders als die von Babbage, nicht den Problemen im Umgang mit parallel arbeitender Hardware. Prozess-Synchronisation wird historisch vielmehr dann zum zentralen Gegenstand der Informatik als sequentiell arbeitende Maschinen unterbrechbar werden und damit das Ausführen paralleler Berechnungen auf ihnen möglich wird. Die technische Voraussetzung dafür wird erstmals im Projekt Whirlwind geschaffen.

Am Servomechanics Lab des MIT begann 1943 die Entwicklung eines Flugsimulators, der in der Geschichte der Informatik insgesamt eine wichtige Rolle spielen sollte: der Aircraft Stability and Control Analyzer (ASCA). Zunächst war dieser als Analogrechner geplant, der die Dynamik von Flugzeugen simulieren und auf passende Ein- und Ausgabegeräte umsetzen sollte. Als zentrales Problem des Projekts stellte sich aber bald heraus, dass ein solcher Analogrechner nicht in der Lage war, Simulationen in einer Geschwindigkeit durchzuführen, die notwendig ist, um die Illusion eines Flugzeuges in Aktion zu erzeugen: Die Fähigkeit zur Arbeit in »Echtzeit« ließ zu wünschen übrig.[15] Durch einen ehemaligen Studienkollegen vom digitalen Rechnen begeistert, setze Jay W. Forrester, der Leiter des Projekts, daraufhin durch, dass auch der ASCA als Digitalrechner konstruiert werden sollte.[16] Im Zuge der Umstellung wurde der Computer 1946 in Whirlwind umbenannt.

Mit diesem Kurswechsel fiel gleichzeitig die Entscheidung für eine dramatische Erweiterung der Kompetenzen des Geräts. Was als spezielle analoge Maschine zur Flugsimulation geplant war, wurde zu einer universellen digitalen Maschine. Diese war zwar die schnellste ihrer Zeit; die analogen Entwicklungen, die ihr vorausgegangen waren, verfügten aber über hochkomplexe Ein- und Ausgabegeräte für die Flugsimulation, die mit einem digitalen Rechner nicht weiter verwendbar waren.[17] 1950 funktionierte Whirlwind, hatte jetzt aber ein Legitimationsproblem: Das Projekt hatte große Summen verschlungen, Interfaces, die Whirlwind zu einem Flugsimulator gemacht hätten, existierten aber nicht. Schon 1947 hatte Warren Weaver daher gefragt: »Was Whirlwind failing to be good biscuits by trying to be cake?«[18]

Die Rettung kam mit einer *optimum application* für das System: Luftabwehr.[19] Whirlwind sollte von einem Rechner, der Flugzeuge simuliert zu einem werden, der

feindliche Flugbahnen und freundliche Abfangkurse berechnet. Die Airforce arbeitete zu diesem Zeitpunkt ohnehin daran, Radardaten über Telefonleitungen zu übertragen – es lag also nahe, diese Radardaten direkt in Whirlwind einzuspeisen. Am Lincoln Lab des MIT wurde damit in den frühen 1950er Jahren aus Whirlwind das Semi Automatic Ground Environment (SAGE) Air Defense System.

Programmunterbrechung

Was Whirlwind von den wenigen anderen digitalen Universalrechnern seiner Zeit unterscheidet, ist die technische Voraussetzung für das Ankoppeln von Algorithmen an die Umwelt, ganz gleich ob Flugpersonal oder Radardaten. Da die Umwelt dem zentralen Takt des Rechners nicht unterworfen ist, erreichen ihn Umweltdaten in der Regel prozessor-asynchron. Die Ankopplung solcher asynchroner Datenquellen kann prinzipiell auf zwei Arten bewerkstelligt werden: Die einfachste Möglichkeit besteht im *polling*. Der Prozessor überprüft in regelmäßigen Abständen, ob neue Daten vorliegen, um sie dann einzulesen. Polling entspricht also dem *busy waiting* der Prozess-Synchronisation und ist ebenso ineffizient. Die andere Möglichkeit besteht in der Unterbrechung des Programmablaufs von außen, wann immer neue Daten eintreffen: dem *interrupt*.[20] Hier signalisiert eine spezielle Hardwarekomponente dem Prozessor, dass Daten vorliegen, um die er sich zu kümmern hat. Der Prozessor unterbricht daraufhin seine Arbeit, wobei er sich deren aktuellen Zustand zuvor merkt, bearbeitet die Daten und kehrt dann an seine ursprüngliche Arbeit zurück.

Mit Whirlwinds Fähigkeit der Programmunterbrechung eröffnen sich zwei neue Gebiete mit neuen Synchronisationsanforderungen: Zum einen ermöglichen prozessor-asynchrone Interrupts das effiziente Einbinden von Daten aus externen Quellen.

15 Redmond/Smith 1977.
16 A. a. O.: 52.
17 A. a. O.: 51 ff.
18 A. a. O.: 52.
19 Everett 1980: 375.
20 Zum Zeitpunkt der Konstruktion von Whirlwind hatte diese Methode noch keinen Namen. Rückblickend kann man davon ausgehen, dass Whirlwind bereits Direct Memory Access (DMA) zum Einlesen der Peripheriegeräte verwendet hat – eine Methode, die auf dem Interrupt basiert, bei der das Einlesen der Daten aber nicht durch den Prozessor geschehen muss, da der controller des Peripheriegeräts über direkten Zugriff auf den Speicher verfügt. Vergleiche hierzu Smotherman 1989.

Zum anderen ermöglichen aber prozessor-synchrone Interrupts das Unterbrechen eines Programmablaufs zu Gunsten eines anderen. Dies kann *pre-emptive* durch einen Prozess mit mehr Rechten als andere oder *cooperative* durch den aktuellen Prozess selber geschehen. Prozessor-synchrone Interrupts bilden die Vorraussetzung für Quasi-Parallelität: dem scheinbar parallelen Bearbeiten von Programmen, bei dem in Wirklichkeit immer wieder (und sehr schnell) zwischen verschiedenen sequentiellen Programmabläufen umgeschaltet wird. Quasi-Parallelität hat dabei alle Nachteile echten parallelen Rechnens – also einen mit der Anzahl der Prozesse linear steigenden Synchronisationsaufwand – ohne dessen Vorteil – Zeitersparnis innerhalb der Grenzen des Gesetzes von Amdahl. Für den Benutzer, und nur für den, entsteht durch quasi-paralleles Rechnen aber die Illusion von Gleichzeitigkeit. Echtzeit – im Gegensatz zur Laufzeit oder der logischen Zeit eines Vorher und Nachher – tritt mit Whirlwind also auf zweifache Weise wieder in die Informatik ein: durch die Kopplung von Rechner und Umwelt, sowie durch scheinbare – und damit nur im Bezug auf eine Umwelt und ihre Zeit sinnvolle – Gleichzeitigkeit.

Viele Jahre später wird Quasi-Parallelität als *multitasking* den endgültigen Sprung auf die Personal Computer beinahe aller Heimanwender schaffen – in Silizium gegossen in der Architektur von Prozessoren und in Software in den dazu vermarkteten Multitasking-Betriebssystemen. *Pre-emptive multitasking* setzt hier voraus, dass »Programme und Daten der einzelnen Anwender voneinander getrennt sind, ebenso wie auch das Betriebssystem vor Benutzerprogrammen geschützt sein muss«.[21] Dieser Schutz, der insbesondere auch ein Schutz des Betriebssystems vor den Ein- und Ausgabe-Interrupts ist, wird mit dem Prozessor-Betriebsmodus des *protected mode* umgesetzt und bildet die Basis der meisten heute verwendeten Betriebssysteme. Die daraus folgende Hierarchie von Berechtigungen, die sich für den Experten als umgehbar erweist, dem Benutzer aber als nicht-hintergehbar verkauft wird, lässt Friedrich Kittler den *protected mode* als Instrument einer Verschwörung der Hard- und Software-Industrie lesen, bei der es um die Überführung einer militärischen Logik in die Informatik geht. Diskursanalytisch erscheint ihm der besondere Schutz des Betriebssystems vor den Anwendern und damit der teilweise Ausschluss der Benutzer als eine Technologie gewordene Antwort auf die Frage nach dem Zugang zur Macht.[22]

Die Kopplung digitalen Rechnens an seine Umwelt schließt umgekehrt aber die Einbeziehung des Bedienpersonals in die Berechnung mit ein. Statt Flugpersonal ist dies bei Whirlwinds Nachfolger SAGE Flugabwehrpersonal, welches dem

Rechner unter anderem die schwierige Aufgabe abnimmt, freundliche von feindlichen Flugbewegungen zu unterscheiden – womit man die Benutzer hier gerade als eingeschlossen in eine militärische, oder zumindest maschinelle, Logik sehen kann.[23]

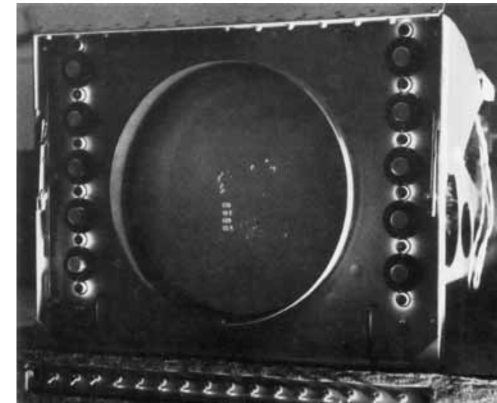

Abb. 6 - Benutzerschnittstellen von Whirlwind. Darstellung von Flugbewegungen auf einem Kathodenstrahlmonitor und Light-Gun.

Technisch erfolgt die Kopplung mittels einer *Light-Gun*, mit der Radardaten in ihrer zweidimensionalen Darstellung auf einem Kathodenstrahlmonitor ausgewählt werden können. Obwohl die Light-Gun als sehr avanciertes Eingabegerät, gegenüber etwa der Maus, erscheinen mag, ist das Gegenteil der Fall. Ihre Funktionsweise ist extrem hardwarenah, sie ist direkt auf Basis des Interrupts realisiert. Die verwendeten Kathodenstrahlmonitore sind zu dieser Zeit noch nicht in Bildpunkte zerlegt, ein Bild entsteht nicht in einer Pixelmatrix, sondern wird Objekt für Objekt gezeichnet. Die Light-Gun verfügt über einen lichtempfindlichen Sensor. Technisch ist die Light-Gun also keine *gun*, was für den Benutzer aber nicht sichtbar wird. Wird sie auf eine Stelle

21 Wie Friedrich Kittler ein Handbuch zum 80168-Prozessor zitiert. Vergleiche Kittler 1993: 211 f.

22 Ebd.: 213.

23 Claus Pias verweist hier auf den Neobehaviorismus und schreibt über SAGE: »Menschen sind daher – wie schon bei Skinner – Prozessoren oder devices«. Pias 2000: 59.

des Bildschirms gerichtet, sendet sie keinen Lichtstrahl aus, sondern registriert mit ihrem Sensor, wenn an diese Stelle ein Objekt gezeichnet und sie vom zugehörigen Kathodenstrahl getroffen wird. Über eine Rückleitung löst die Light-Gun einen Interrupt aus. Da das Zeichnen auf den Bildschirm objektweise erfolgt, ist das Objekt, dessen Zeichnung unterbrochen wurde, das ausgewählte. Das Ausrechnen von Koordinaten und die Entscheidung, was die Light-Gun getroffen hat, entfallen.

Time-Sharing

Mit der so realisierten Einbindung des Bedieners in das Rechnen in Echtzeit wirft die Programmunterbrechung neue Synchronisationsprobleme auf. In der Folge entstehen daher Versuche, die Synchronisation von Rechner und Nutzer auf ähnlich formale und damit sichere Füße zu stellen, wie die Prozess-Synchronisation.

Einer der ersten, die den Benutzer dazu explizit in den Blick nehmen, ist Joseph Carl Robnett Licklider. In seinem berühmten Aufsatz »Man-Computer Symbiosis« entwirft er 1960 das Bild ›kooperativer Interaktion‹ von Mensch und Computer, wobei der Begriff der Interaktion hier vielleicht zum ersten Mal fällt und der Synchronisation von Mensch und Rechner nachhaltig einen Namen gibt.[24] Motiviert wird sein Ansatz durch die Forderung nach Zeitersparnis. Offenbar in Unkenntnis der Arbeiten der Arbeitswissenschaft und des Taylorismus,[25] unterzieht er sich einem informellen Selbstversuch. Dessen Ergebnis ist die Erkenntnis, dass 85% seiner Arbeitszeit damit vergeudet wird, die Vorraussetzung für die eigentliche Arbeit zu schaffen – »to get into a position to think«. Um diese 85% zu reduzieren, soll Interaktion von Mensch und Computer kooperatives Denken in Echtzeit – »real-time thinking« – ermöglichen. Dem entgegen steht aber, so Licklider, die Eigenschaft des Rechners, nur dem vollständigen Abarbeiten vorformulierter Anweisungen zu dienen. Um das zu ändern, muss das Verhältnis der Geschwindigkeiten von Mensch und Maschine in den Blick genommen werden: Ein *speed-mismatch* sorgt dafür, dass ein einzelner Rechner für eine erfolgreiche Kooperation zu schnell arbeitet. Andererseits sind Computer zu teuer, um ständig auf den langsamen Menschen zu warten. Die Lösung basiert auf Programmunterbrechung und bekommt den Namen *Time-Sharing*: Der Rechner soll auf den Menschen warten. Statt aber im *busy waiting* seine Rechenzeit ausschließlich mit Warten zu vergeuden, soll er in dieser Zeit anderen Menschen oder Aufgaben zur Verfügung stehen und sich bei Bedarf unterbrechen lassen. Time-Sharing realisiert

also das In-Reihe-bringen elementarer Abläufe unter der Maßgabe, dass einige der elementaren Operationen von einem menschlichen Benutzer durchgeführt werden. Die Forschungsprojekte der 1960er Jahre zum Time-Sharing sind damit die direkte Voraussetzung für das oben genannte Multitasking der Gegenwart.

Diesen als wegweisend rezipierten Text schreibt Licklider allerdings nicht ohne einschlägige Vorgeschichte. Als Psychologe arbeitete er seit 1950 am Psychoakustik-Labor des MIT. Hier kam er über das Lincoln Lab mit der Informatik und speziell mit Whirlwind in Kontakt.[26] Licklider hatte also, als er die Vision einer *man-computer symbiosis* entwarf, die Grundzüge des Time-Sharing bereits erlebt: an Whirlwind und SAGE.

1962 wurde Licklider Chef des Information Processing Techniques Office der DARPA. Hier war er in der Position, seine Vision finanziell zu unterstützen.[27] Zu den so geförderten Projekten gehörte das Project MAC (Multi Access Computer) am MIT, welches ein Time-Sharing-Betriebssystem entwickelte und später (jetzt als Machine Aided Cognition) die Keimzelle des Artificial Intelligence Lab bildete und die Arbeit von Douglas Engelbart am Stanford Research Institute (SRI), von der unten noch zu sprechen ist.

Informationsverarbeitende Systeme

Die Bediener des SAGE – Personal der Airforce – mussten an den Umgang mit dieser völlig neuen Computertechnik erst gewöhnt werden. Am Systems Research Laboratory der RAND Corporation wurden zukünftige Radar-Bediener daher unter der Leitung von Allen Newell und Herbert A. Simon mit simulierten Daten trainiert. Newell und Simon kommen nach dieser Arbeit zu einem Schluss, der die Kognitionswissenschaft begründen soll: Menschen sind, wie Computer, Vertreter einer allgemeinen Klasse von Systemen – nämlich informationsverarbeitende.

24 Licklider 1960.
25 Zu dieser Tradition vergleiche Pias 2000: 23 ff. Bereits Babbage ist eine wichtige Figur sowohl für die Informatik als auch die Arbeitswissenschaft. Das Babbage-Prinzip, welches besagt, dass mit der Aufteilung der Arbeit auf unterschiedlich qualifizierte Arbeiter die Lohnkosten sinken entstammt direkt seiner Auseinandersetzung mit der Mechanisierung von Kopfarbeit. Vgl. Babbage 1935.
26 Waldrop 2001: 147.
27 Press 1993.

Beide vollziehen damit einen Bruch vor allem mit dem Behaviorismus, aber auch mit der Kybernetik. Von jetzt an wird das Problemlösen und das Treffen diskreter Entscheidungen (etwa der über Freund und Feind) zur Grundaktivität des Menschen. Dieser ist keine Black-Box mehr, sondern verfügt über Repräsentationen und Prozesse – so wie die Von-Neumann-Architektur über Speicher und davon getrennte Verarbeitung verfügt. Die Idee des Menschen als Informationsverarbeitendes System enstammt also der Beobachtung von Menschen, die Entscheidungen am Rechner treffen.[28]

H-LAM/T

War Time-Sharing mit der Vision angetreten, durch Synchronisation von Mensch und Rechner Zeit zu sparen, so wurde lange weder die Quantität dieser Ersparnis geklärt, noch hatte man sich der Korrektheit des interaktiven Rechnens angenommen. Zwar war die Korrektheit der Entscheidungen, die am SAGE getroffen wurden, sicher Gegenstand des Trainings durch Newell und Simon, die Frage aber, wie interaktives Rechnen mit maximalen Geschwindigkeitszuwächsen bei minimaler Fehlerquote zu erreichen sei, blieb vorerst ungeklärt.

Am 9. Dezember 1968 stellt Douglas Engelbart vom Augmentation Research Centre des SRI das *on-line system* NLS vor. Diese Vorstellung sollte als »The Mother of all Demos« in die Informatikgeschichte und -mythologie eingehen – unter anderem, weil hier zum ersten Mal die Computermaus vorgestellt wurde.[29] Der Demo vorausgegangen war die Entwicklung des NLS als eines Computersystems, das erstmals explizit für einen Benutzer geformt worden war, über den ein exaktes Bild in Form einer Theorie zur Verfügung stand.

Engelbarts Theorie hinter dem NLS ist eines der ersten formalen Modelle des Menschen am Rechner. Das H-LAM/T-Modell beschreibt den Benutzer als »Human using Language, Artifacts, Methodology, in which he is Trained«.[30] Die Inspiration für das System liefert der Linguist Benjamin Lee Whorf mit der Sapir-Whorf-Hypothese – dem linguistischen Relativitätsprinzip. Für Engelbart besagt dieses Prinzip, dass die verwendete Sprache determiniert, was für ihren Sprecher überhaupt sag- und denkbar ist. Engelbart erweitert das Prinzip von Sprache auf *Artifacts* und *Methodology* und schlussfolgert, dass verbesserte Artefakte den gesamten Prozess des Denkens und Handelns eines darauf trainierten Menschen beeinflussen werden.

In Engelbarts Modell bestehen Handlungen aus einer Hierarchie bewusster und unbewusster Teilhandlungen – elementarer Operatoren also, vergleichbar denen des Algorithmenbegriffs. Mensch-Maschine-Interaktion besteht nach Engelbart darin, dass bestimmte Prozesse aus der Hierarchie menschlichen Denkens über ihre Handlungen an die ebenfalls hierarchisch geordneten Prozesse im Rechner gekoppelt sind. Eine Verbesserung dieser Kopplung soll nun positive Folgen für die gesamte Hierarchie menschlicher Handlung haben. Um die Arbeit (und das Denken) des Menschen am Rechner zu optimieren, gilt es also, optimale Ein- und Ausgabegeräte zu finden – für Rechner im Modus des Time-Sharing, den Engelbart *on-line* nennt.

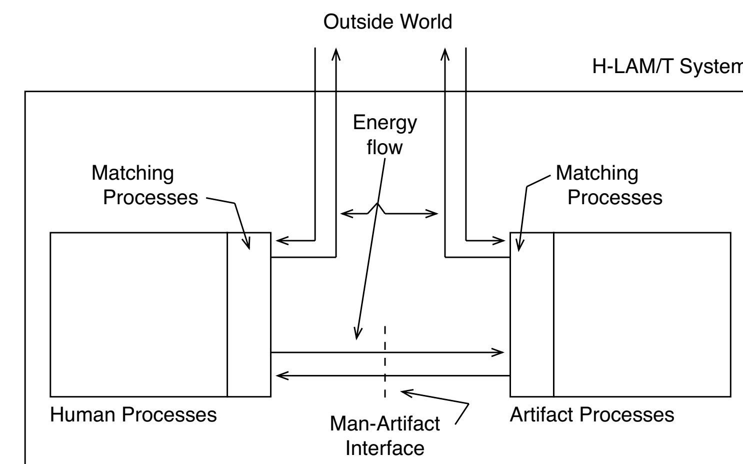

Abb. 7 - H-LAM/T nach Engelbart.

Die Kriterien für Optimalität sind dabei bekannt: Geschwindigkeit und Korrektheit. Um die optimalen Eingabegeräte für das NLS zu finden, führt das Team des SRI daher eine Studie zur Benutzbarkeit verschiedener Geräte durch.[31] Darin werden ein Joystick, ein Gerät zur Eingabe von Polarkoordinaten namens Grafacon, die Maus

28 Vergleiche hierzu Newell 1957 und Pias 2000: 59.
29 Obwohl oft die Erfindung der Maus als der entscheidende Durchbruch des NLS gesehen wird, ist doch davon auszugehen, dass alle technischen Komponenten sowie Praktiken der Interaktion via Maus zu diesem Zeitpunkt bereits verfügbar waren und am SRI lediglich zusammengeführt wurden. Vgl. Roch 1996.
30 Engelbart 1962.
31 Engelbart 1967.

und die Light-Gun gegeneinander getestet. Dazu kommt ein Gerät, um den Cursor, den die Gruppe als *bug* bezeichnet, mittels Kniebewegungen zu steuern – weniger einfallsreich wird es *knee-control bug position device* genannt. Alle neuen Eingabegeräte sollen im Verbund mit einer Tastatur eingesetzt werden. In einer Reihe von Experimenten bewältigen Benutzer – ähnlich den Bedienern des SAGE – einfache Selektionsaufgaben: Eine Gruppe von Zeichen wird auf dem Monitor gezeigt, dann muss die Leertaste gedrückt und anschließend ein Zeichen oder eine Teilgruppe der Zeichen mit dem entsprechenden Zeigegerät ausgewählt werden. Dabei werden Geschwindigkeit und die Rate korrekter Auswahl gemessen.

Bei der Interpretation der Testergebnisse nimmt die Gruppe neben Geschwindigkeit und Korrektheit aber auch andere Kriterien in den Blick. Ist die Light-Gun gerade für unerfahrene Benutzer das beste Gerät was diese Kriterien angeht, so spricht am Ende einiges für die Maus: Sie muss nicht hochgehoben werden und bleibt nach der Benutzung an ihrem Platz. Auch das Knie-Gerät ist vielversprechend, macht es doch das Umgreifen zwischen Tastatur und Auswahlgerät überflüssig. Für eine Bewertung im Sinne des H-LAM/T ist aber entscheidend, dass der Benutzer im Umgang mit dem entsprechenden *Artifact* geschult ist. Da für das Knie-Gerät keine trainierten Benutzer zur Verfügung stehen, wird es in der Auswertung weitgehend ausgeblendet. Dass die endgültige Entscheidung zu Gunsten der Maus fällt, ist demnach der Theorie des H-LAM/T zu verdanken. Denn anders als im heutigen Interaktionsdiskurs wertet die Gruppe die intuitive Nutzbarkeit der Light-Gun weniger hoch als die Performanz trainierter Nutzer im Dauereinsatz mit der Maus. Sollte diese Entscheidung im Falle der Maus die Computernutzung bis heute beeinflussen, wurde das Beharren auf die Notwendigkeit des Trainings dem NLS an anderer Stelle vielleicht zum Verhängnis: Die Benutzung der Tastatur setzte das langwierige Erlernen eines 5-Bit-Binärcodes voraus, was einer weiten Verbreitung des Systems entschieden im Weg stand.

The Magical Number Seven und Fitts' Law

Forschung, wie die am SRI, hatte den Erfolg der Anwendung von Methoden der experimentellen Psychologie auf die Konstruktion von Computertechnologie demonstriert. Gleichzeitig lag mit den Kognitionswissenschaften früh eine Modell-orientierte Wissenschaft vom Menschen vor, die diesen von Beginn an als Mensch am Rechner verstanden hatte. Auf der Suche nach neuen Modellen des menschlichen Verhaltens

am Rechner wurden in der Folge experimentelle Ergebnisse der Psychologie der 1950er Jahre wiederentdeckt und zu harten Gesetzen der Interaktionsforschung. Zwei dieser Gesetze erlangten für einige Zeit besondere Berühmtheit: *The Magical Number Seven* und *Fitts' Law.*

Der Psychologe George Miller fasste 1956 verschiedene Experimente zusammen, die alle den Versuch unternommen hatten, die Kapazität der sensorischen und motorischen Kanäle des Menschen sowie seines Gedächtnisses zu bestimmen.[32] Miller war aufgefallen, dass diese Kapazität für verschiedenste Modalitäten (wie etwa die Wahrnehmung von Tonhöhen, Lautstärke, Positionen im Raum aber auch die Speicherkapazität von Daten im Kurzzeitgedächtnis) immer in einer Größenordnung von sieben plus/minus zwei Informationseinheiten liegt. Eine Größe, die er als »Magical Number Seven, Plus or Minus Two« bezeichnete. Diese Zahl sollte zusammen mit einem weiteren Konzept starken Einfluss entfalten: *chunking.* Fasst eine Versuchsperson beispielsweise eine Zahlenfolge, die deutlich länger als sieben Zeichen ist, in *chunks* genannte Gruppen zusammen, gilt die Gesetzmäßigkeit der *magical number seven* nicht mehr für die einzelnen Zeichen, sondern für eben diese *chunks*. Für die Interaktionsforschung bleibt von dieser Erkenntnis vor allem das Gesetz, dass sieben plus/minus zwei Informationseinheiten, die es nun in optimalen *chunks* zu gruppieren gilt, die ›magische‹ Obergrenze der Kapazität des Kurzzeit- oder Arbeitsgedächtnisses darstellen – und dass es folglich möglich ist, allgemeine Kapazitäten und Verarbeitungszeiten von Computerbenutzern zu bestimmen.

Das wahrscheinlich einflussreichste Gesetz, welches das Zeitverhalten eines Benutzers mathematisch beschreibt, ist Fitts' Law. Wie die Ergebnisse von Miller entstammt auch dieses Gesetz einer experimentellen Psychologie, die in der Tradition der mathematischen Kommunikationstheorie von Claude Shannon verwurzelt ist und versucht, Motorik und Sensorik des Menschen als Kanäle zu verstehen, die Informationen übertragen. Fitts will zu Beginn der 1950er Jahre zeigen, dass die Kapazität des motorischen Systems unabhängig von der Art einer ausgeführten Bewegung konstant ist.[33] Dazu benutzt er das Shannon-Hartley-Gesetz, welches die maximale Kapazität C eines gestörten Kanals in Abhängigkeit von der Bandbreite B des Kanals und der *signal-to-noise-ratio* S/N beschreibt:

$$C = B \log_2 \frac{S+N}{N} = B \log_2 (1 + \frac{S}{N})$$

32 Miller 1956.

33 Fitts 1954.

In Fitts' Experimenten müssen Versuchspersonen verschiedene Bewegungen möglichst schnell wiederholt ausführen. Zunächst ist dies eine Hin-und-Her-Bewegung eines Metallstifts zwischen zwei Metallplatten, deren Größe und Abstand systematisch variiert werden. Bei Berührung der Platten mit dem Stift wird ein Kontakt geschlossen und die Bewegung damit messbar. Die Bandbreite der menschlichen Motorik als Kanal wird nun zur Frequenz dieser Bewegung, die *signal-to-noise-ratio* zum Verhältnis von Abstand und Größe der Zielplatten. Aus dem Signal-Rausch-Abstand wird damit ein *speed-accuracy-tradeoff*.

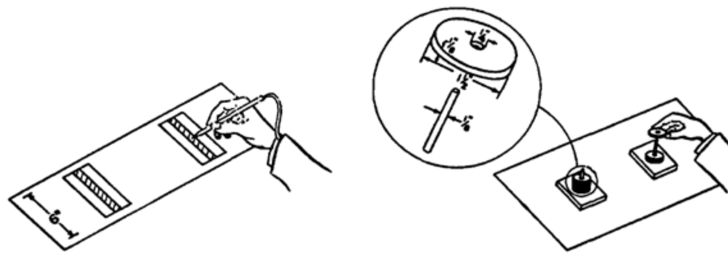

Abb. 8 - Fitts' Experimente zur Bestimmung der Kanalkapazität menschlicher Motorik.

Weitere Aufgaben sind das Umstecken von Metallscheiben von einem Stift auf den anderen, wobei das Größenverhältnis von Stift und Loch in der Scheibe variiert wird, sowie das Umstecken einer Reihe von Metallstiften zwischen zwei Leisten mit Löchern. Hier wird die Größe der Stifte variiert. Die gemessenen Werte werden nun festgehalten, während die Geschwindigkeit als Zeit pro Bewegung bestimmt wird. Diese Größe müsste nun, in Analogie zur benutzten Formel, der Bandbreite entsprechen und damit proportional zur Kapazität des motorischen Systems des Menschen sein. Und tatsächlich ist sie relativ konstant, was Fitts' Annahme konstanter Kapazität bestätigt.

Für die Interaktionsforschung wird die von Fitts verwendete Formel interessant, weil sie nicht nur Grundlage einer Bestimmung der Kanalkapazität menschlicher Motorik ist, sondern umgekehrt zu einem Gesetz zur Vorhersage werden kann. Wenn der Zusammenhang, den die Formel beschreibt, nämlich stimmt, dann folgt daraus, dass die Dauer korrekt ausgeführter Handlungen (wie das Treffen einer Metallscheibe oder die Auswahl eines Ziels mit der Maus) eine Funktion von Abstand A und Zielgröße W ist, die um konstante Werte verschoben und skaliert wurde.

$$T = a + b \log_2(1 + \frac{A}{W})$$

Die Werte von a und b sind dabei Materialkonstanten, die vom verwendeten Eingabegerät abhängen. Das Gesetz kann also einerseits den Zeitaufwand für die Zielauswahl (das *pointing*) mit Geräten, für die a und b experimentell bestimmt wurden, vorhersagen. Andererseits können die Werte für a und b verschiedener Geräte bestimmt und diese so miteinander verglichen werden.

Human-processor model

Auf solche Erkenntnisse der Psychologie bauend, werden weitere Modelle entwickelt, die die Synchronisation von Mensch und Rechner auf eine vollständige formale Basis stellen sollen: »Any cognitive model must emerge from an understanding of human problem-solving capabilities and information-processing capabilities«, erklärt etwa Ben Shneiderman. Wobei ihm als Beleg für die Endlichkeit der letzteren die magische Sieben dient.[34] Da Interaktion dabei im Wesentlichen als *problem solving* verstanden wird, sollen diese Modelle bei gegebenem Problem und Computersystem zu dessen Lösung (einem *Artifact* im Sinne des H-LAM/T) die Zeit vorhersagen, welche die Lösung brauchen wird. Da in dieser Denktradition das Problemlösen am Rechner (und nicht nur da) einem hierarchischen Prozess elementarer Informationsverarbeitungsschritte entspricht, stellt sich die Gesamtdauer einer Handlung als Summe der Dauer der Teilhandlungen dar. Die Zeit, die ein *task* braucht ist daher die Zeit für das ›Erfassen‹ und das ›Ausführen‹ des *task*. Erstere ist *task*-abhängig und muss experimentell bestimmt werden. Letztere lässt sich weiter zerlegen in Zeiten für das Ausführen bestimmter Operatoren. Das »Keystroke-level Model«,[35] das die Bildschirmarbeit mit Tastatur und Maus beschreibt, unterscheidet hier Operatoren für Tastendruck: *homing*, *pointing*, *drawing* sowie ›mentale Operatoren‹ und die Antwortzeit des Systems. Die Dauer der Ausführung der Operatoren wird mit existierenden Modellen – wie etwa Fitts' Law – vorhergesagt oder als empirische Konstante bestimmt. Eine eigene Notation erlaubt es dabei, die Handlungsabläufe am Rechner als Abfolge solcher Operatoren zu schreiben.

34 Shneiderman 1984: 267.

35 Card u. a. 1980.

Aus dem Keystroke-Modell wird später ein allgemeines Modell des Menschen am Rechner, das mit dem Namen »Goals, Operators, Methods and Selection rules« (GOMS) das Verständnis vom Computernutzer als (bestenfalls teilweise parallel arbeitenden) Algorithmus explizit macht. Dieses Modell wird von den Erfindern des Keystroke-Modells, darunter mit Allen Newell, einem der Begründer der Kognitionswissenschaft, erstmals unter dem Titel der »Human-Computer Interaction« festgehalten.[36] Mit dem Buch liegt eine vorerst vollständige Modellierung des Menschen als modulares System vor. Den einzelnen Modulen können dabei detaillierte Zahlen zu Kapazität sowie Zugriffs- und Verarbeitungszeiten zugeordnet werden. Auf oberster Ebene ist der Computernutzer in diesem Modell ein System von Modulen für Eingabe, Ausgabe und Verarbeitung. Er entspricht damit der Von-Neumann-Architektur, was wenig wundert, war diese doch explizit in Analogie zum menschlichen Zentralnervensystem gedacht worden.[37] Auf *Task*ebene ist dieser Nutzer ähnlich dem Algorithmusbegriff durch elementare Operatoren und deren regelgeleitetes Aufeinanderfolgen gekennzeichnet. Konsequenterweise wird dieser Mensch *human processor* genannt.

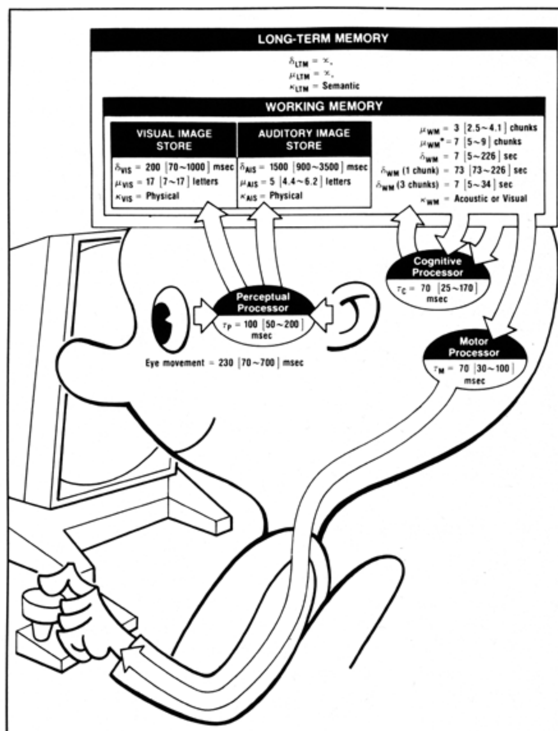

Abb. 9 - Der Human Processor in der Darstellung von Card et al.

Mit Hilfe des *human processor models* sollte die Konstruktion der Schnittstellen, an denen Mensch und Maschine miteinander synchronisiert werden, mit formalen Methoden erfolgen können. Wie die Erfinder des *human processor* rückblickend feststellen, war dies »a model precise enough to enable designers to perform back-of-the-envelope calculations, a model that identified key constraints in the design space.«[38] Als Beispiel für solche *key constraints* muss hier wieder das Gesetz herhalten, das sich bisher empirisch als das robusteste erwiesen hat und dessen Brauchbarkeit zwar unbestritten aber auch von großer Einfachheit gekennzeichnet ist. Auf die Abhängigkeit von Geschwindigkeit und Korrektheit einer *pointing*-Bewegung von Zielentfernung und -größe in Fitts' Law verweisend schreiben sie: »The model provided guidance for interface designers: make distant buttons large, for example.«

Ungeduldiges Warten

Neben den Versuchen, das Synchronisationsproblem auf formalem Weg zu lösen, wurde der *speed mismatch* von Rechner und Benutzer auch aus der Sicht der Organisations- und Arbeitswissenschaft betrachtet. Hier galt es, die Faktoren zu bestimmen, die den Menschen am Rechner, verstanden als *operator* in einem Arbeitskontext, zu maximaler Produktivität verhelfen. Produktivität ist dabei die Differenz zwischen richtigen und falschen Eingaben pro Zeit,[39] also maximale Korrektheit bei minimalem Zeitverbrauch oder: Effizienz.

In den 1980er Jahren werden so die Gesetze des Zusammenhangs der *system response time* mit der Produktivität der *operator* untersucht. Es stellt sich heraus, dass eine schnellere Antwortzeit keineswegs immer mit höherer Produktivität zusammenfällt.[40] Schon in dieser Forschung fällt darüber hinaus auf, dass die Gesetze aus GOMS und anderen Modellen nicht universell anwendbar zu sein scheinen. Die verschiedenen Summanden in der Summe der Dauer von Bildschirmarbeit, mit denen Modelle wie Keystroke oder GOMS arbeiten, erweisen sich als nicht unabhängig voneinander.

36 Card u. a. 1983.
37 »The three specific parts […] correspond the associative neurons in the human nervous system. It remains to discuss the equivalents of the sensory or afferent and the motor or efferent neurons. These are the input and the output organs of the device«. Neuman 1945: 3.
38 Card/Moran 1986: 106.
39 Barber/Lucas 1983: 973.
40 Shneiderman 1984: 280.

Vielmehr ist etwa die Lesegeschwindigkeit von der Geschwindigkeit des Bildschirmaufbaus abhängig.[41] Unterschiedliche Antwortzeiten eines Systems können zu unterschiedlichen, und nicht mehr miteinander vergleichbaren, Strategien der Problemlösung führen. Der *pace* eines Systems vermischt sich mit dem Arbeitsrhythmus des Menschen und beide sind nicht mehr eindeutig zu trennen.[42]

Dazu kommt die Erkenntnis, dass Produktivität auch mit subjektiv erlebter Zufriedenheit zusammenhängt. Zufriedenheit ist wiederum unter anderem von der *response time* des Rechners abhängig, was zu einem Geflecht wechselseitiger Einflüsse von Antwortzeit des Systems, Zufriedenheit, Fehler-Rate und Produktivität führt.[43] Die Synchronisation von Mensch und Maschine ist nämlich, vergleichbar der Prozess-Synchronisation, durch das wiederholte wechselseitige Warten aufeinander gekennzeichnet. Der schnelle Rechner, der im Time-Sharing auf den langsamen Menschen wartet, ist also nicht der einzige Wartende. Als im Zuge der weiten Verbreitung von Time-Sharing-Systemen der *operator* des Rechners zum direkt eingebundenen *user* wird, verschiebt sich die Frage von der nach »System Response Time, Operator Productivity and Job Satisfaction«[44] hin zu »What Makes Users Happy?«.[45] Als statistisch deutlichster *user satisfaction inducer* entpuppt sich hier die *response time*. Während nach Erkki Huhtamo der *waiting operator* der frühen Informatik die Lochkarten verarbeitenden Maschinen gleich einem Hohepriester betreute, erzeugen Time-Sharing und später Personal Computing einen ungeduldig wartenden *impatient user*.[46] Für diesen bedeutet Arbeiten, folgt man Lev Manovich, in erster Linie Warten.[47] Mit anderen Worten: Kooperative Interaktion im Modus des Time-Sharing bedeutet trotz Time-Sharing weiterhin auf das »getting into a position to think« zu warten. Der *impatient user* hat zwar seit den 1980ern ständig steigende Rechengeschwindigkeit und Bandbreite zur Verfügung, diese werden aber im beinahe gleichen Maße aufgebraucht – sei es durch Photorealismus und simulierte Physik in Computerspielen oder durch neue Applikationen und Formate in den Netzen. »Web surfing provides a perfect example«, schreibt Lev Manovich Mitte der 1990er, »a typical user may be spending equal time looking at a page and waiting for the next page to download.«[48] Bis heute ist das Web ein asynchrones und damit ein Warte-Medium geblieben. Dass es sich seit damals mit enormem Erfolg zu einem (angeblichen) Web 2.0 entwickeln konnte, wobei es seine wachsende Bandbreite und damit die potentielle Verringerung der Wartezeiten konsequent auffrisst, liegt eventuell an dem Faktor, den oben genannte Statistik lange vor der Entwicklung des Web auf dem zweiten Platz der *user satisfaction inducers* festgemacht hatte: die *number of users*, die am gleichen System arbeiten.

41 Ebd.: 270.
42 O'Donnell/Draper 1996.
43 Barber/Lucas 1983: 973.
44 Barber/Lucas 1983.
45 Rushinek/Rushinek 1986.
46 Huhtamo 1999: 106 ff.
47 A. a. O.: 106.
48 Manovich 1996.

Abbildungsnachweise

Abb. 1: Bromley 1998: 33, 34.
Abb. 2: Tanenbaum 1998: 552.
Abb. 3: Dijkstra 1971: 131.
Abb. 4: eigene Darstellung.
Abb. 5: Fleischmann 1990: 83, 85.
Abb. 6: Everett 1980: 376.
Abb. 7: Engelbart 1962: 20.
Abb. 8: Fitts 1954: 384, 386.
Abb. 9: Card u. a. 1983: 26.

Literatur

Amdahl, Gene M. 1967: Validity of the Single-Processor Approach to Achieving Large-Scale Computing Capabilities. In: AFIPS Conference Proceedings, Bd. 30. Washington, DC u. a: Thompson u. a.

Babbage, Charles 1835: On the Economy of Machinery and Manufactures. London: Charles Knight.

Babbage, Major-General Henry P. 1888: The Analytical Engine. In: Proceedings of the British Association (1888). URL: http://www.fourmilab.ch/babbage/hpb.html (zuletzt gesehen am 09.03.2010).

Barber, Raymond E./ Lucas, Jr. Henry C. 1983: System response time operator productivity, and job satisfaction. In: Communications of the ACM 26/11 (1983). S. 972–986 .

Bigelow, Julian 1980: Computer Development at the Institute for Advanced Study. In: Metropolis, Nicholas u. a. (Hrsg.): A History of Computing in the Twentieth Century. Orlando, FL: Academic Press. S. 291–310.

Bromley, Allan G. 1998: Charles Babbage's Analytical Engine, 1838. In: IEEE Annals of the History of Computing 20/4 (1998). S. 29–45.

Card, Stuart K. u. a. 1980: The keystroke-level model for user performance time with interactive systems. In: Communications of the ACM 23/7 (1980). S. 396–410.

Card, Stuart K. u. a. 1983: The Psychology of Human-Computer Interaction. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

Card, Stuart/Moran, Thomas 1986: User technology – from pointing to pondering. In: Proceedings of the ACM conference on the history of personal workstations 1986. S. 183–198.

Dijkstra, Edsger W. 1971: Hierarchical ordering of sequential processes. In: Acta Informatica 1/2 (Juni 1971). S. 115–138.

Dijkstra, Edsger W. 1997: The tide, not the waves. In: Peter J. Denning/Robert M. Metcalfe (Hrsg.): Beyond calculation: the next fifty years. New York, NY: Copernicus, 1997. S. 59–64.

Engelbart, Douglas C. 1962: Augmenting Human Intellect. A Conceptual Framework. Summary Report AFOSR-3223. Stanford Research Institute, Menlo Park, Ca. (Oktober 1962).

Engelbart, Douglas C. u. a. 1967. Display-Selection Techniques for Text Manipulation. In: IEEE Transactions on Human Factors in Electronics 1 (1967). S. 5–12.

Everett, Robert 1980: Whirlwind. In: Metropolis, Nicholas u. a. (Hrsg.): A History of Computing in the Twentieth Century. Orlando, FL: Academic Press, 1980. S. 365–384.

Fitts, Paul M. 1954: The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. In: Journal of Experimental Psychology 47/6 (1954). S. 381–391.

Fleischmann, Georg 1990: Leistungsbewertung paralleler Programme für MIMD-Architekturen: Modellbildung und mathematische Analyse. Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung. Universität Erlangen-Nürnberg: Dissertation.

Flynn, Michael J. 1972: Some Computer Organisations and Their Effectivness. In: IEEE Transaction on Computers C-21/9 (1972). S. 948–960.

Huhtamo, Erkki 1999: From Cybernation to Interaction. A Contribution to an Archeology of Interaction. In: Peter Lunenfeld (Hrsg.): The Digital Dialectic. New Essays on New Media. Cambridge, MA u. a.: MIT Press. S. 96–111.

Kittler, Friedrich 1993: Protected Mode. In: Draculas Vermächtnis. Technische Schriften. Leipzig: Reclam.

Licklider, J. C. R. 1960: Man-Computer Symbiosis. In: IRE Transactions on Human Factors in Electronics HFE-1 (März 1960). S. 4-11. URL: http://groups.csail.mit.edu/medg/people/psz/Licklider.html (zuletzt gesehen am 09.03.2010).

Lamport, Leslie 1978: Time, Clocks, and the Ordering of Events in a Distributed System. In: Communications of the ACM, 21/7 (1978). S. 558–565.

Manovich, Lev 1996: Global Algorithm 1.3: The Aesthetics of Virtual Worlds. Report From Los Angeles. Ctheory (1996). URL: http://www.ctheory.net/articles.aspx?id=34 (zuletzt gesehen am 08.02.2010).

Miller, George A. 1956: The Magical Number Seven, Plus or Minus Two. Some Limits on Our Capacity for Processing Information. In: The Psychological Review 63 (1956). S. 81–97. URL: http://www.musanim.com/miller1956 (zuletzt gesehen am 09.03.2010).

Neumann, John von 1945: First Draft of a Report on the EDVAC. Contract No. W–670–ORD–4926 Between the United States Army Ordnance Department and the University of Pennsylvania. Moore School of Electrical Engineering, University of Pennsylvania, Juni 1945.

Newell, Allen 1957: Information Processing. A new Technique for the Behavioral Sciences. Pittsburgh, Carnegie Institute of Technology: Dissertation.

O'Donnell, Paddy/Draper, Stephen W. 1996: How machine delays change user strategies. In: ACM SIGCHI Bulletin 28/2 (1996). S. 36–38.

Petri, Carl Adam 1962: Kommunikation mit Automaten. Bonn: Friedrich-Wilhelms-Universität, Rheinisch-Westfälisches Institut für instrumentelle Mathematik, Dissertation.

Pias, Claus 2000: Computer Spiel Welten. Weimar: Bauhaus-Universität, Dissertation.

Press, Larry 1993: Before the Altair: the history of personal computing. In: Communications of the ACM 36/9 (1993). S. 27–33.

Redmond, Kent C./Smith, Thomas M. 1977: Lessons from Project Whirlwind. In: IEEE Spectrum 14/10 (1977).

Roch, Axel 1996: Die Maus. Von der elektrischen zur taktischen Feuerleitung. In: Hans Ulrich Reck u. a. (Hrsg.): Lab. Jahrbuch der Kunsthochschule für Medien Köln. Köln: Verlag der Buchhandlung Walther König. S. 166–173.

Rushinek, Avi/Rushinek, Sara F. 1986: What makes users happy? In: Communications of the ACM 29/7 (1986). S. 594–598.

Shneiderman, Ben 1984: Response time and display rate in human performance with computers. In: ACM Computing Surveys 16/3 (1984). S. 265–285.

Smotherman, Mark: A Sequencing-based Taxonomy of I/O Systems and Review of Historical Machines. In: Computer Architecture News 17/5 (1989). S. 5–15.

Tanenbaum, Andrew S. 2006: Computerarchitektur. Strukturen – Konzepte – Grundlagen. München: Pearson Studium, 5. Auflage 2006.

Tanenbaum, Andrew S. 1998: Structured Computer Organization. Upper Saddle River, NJ: Prentice-Hall, 4. Auflage 1998.

Waldrop, M. Mitchell 2001: The Dream Machine. J. C. R. Licklider and the revolution that made computing personal. New York: Viking.