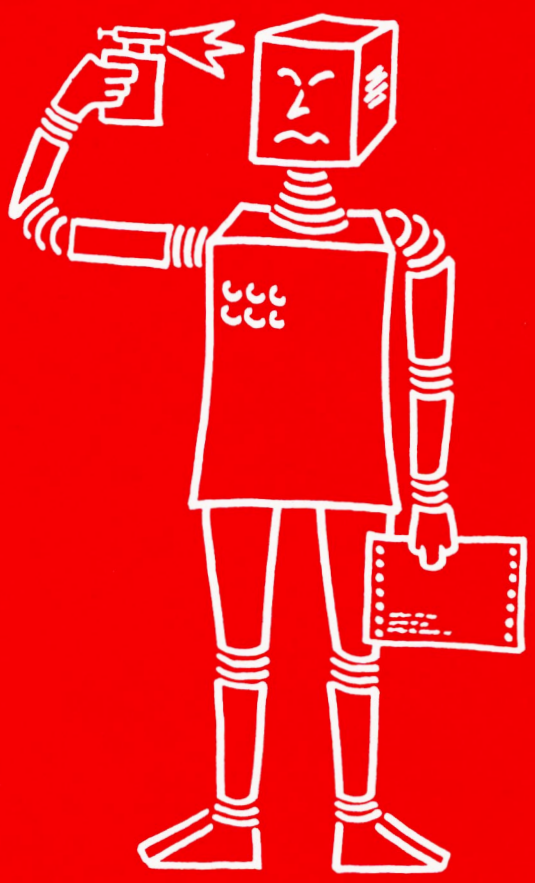
SEKI-PROJEKT

SEKI MEMO

Institut für Informatik III
Universität Bonn
Bertha-von-Suttner-Platz 6
D 5300 Bonn 1, W. Germany

Institut für Informatik I
Universität Karlsruhe
Postfach 6380
D-7500 Karlsruhe 1, W. Germany



MEMO SEKI-BN-81-07

Ein verteiltes Betriebssystem für CSSA

von

Dieter Wybranietz

Ein verteiltes Betriebssystem für CSSA

Dieter Wybranietz

SEKI-Projekt
Institut für Informatik
der Universität Bonn
Abt. III
Bertha-von-Suttner-Platz 6
D-5300 Bonn 1

Zusammenfassung: CSSA ist eine interaktive Programmiersprache zur Definition von asynchron parallel arbeitenden Prozessen, die auf universellen Multicomputersystemen ausgeführt werden können. Im Gegensatz zu vergleichbaren verteilten Betriebssystemen, die sich an einer vorhandenen Hardwarestruktur orientieren, wurde hier ein Betriebssystem entwickelt, in dem hardwareunabhängig von einem vorgegebenen Berechnungsmodell und einer Programmiersprache ausgegangen wurde. Dies führte zu einem Betriebssystem, daß vollständig in das Gesamtkonzept von CSSA integriert ist und nicht nur die von CSSA geforderten Dienstleistungen erbringt, sondern darüberhinaus in Verbindung mit CSSA weitere Möglichkeiten und Anwendungen im Hinblick auf Mehrrechnersysteme bietet. Den Hintergrund dieser Arbeit bildet die zur Zeit stattfindende erste Implementierung von CSSA auf einem Multicomputersystem.

A Distributed Operating System for CSSA

Dieter Wybranietz

SEKI-Projekt
Institut für Informatik
der Universität Bonn
Abt. III
Bertha-von-Suttner-Platz 6
D-5300 Bonn 1

Abstract: CSSA is an interactive programming language allowing the definition of asynchronous concurrently working processes, which can be executed on universal multicomputer systems. Here an operating system is described, that has been developed independently of any hardware environment, but has been constructed starting from a computational model and a programming language. This approach led to an operating system, which stands in contrast to comparable distributed operating systems, where special hardware structures affected the design. The CSSA operating system is completely integrated in the language concept of CSSA. It not only provides the services requested by CSSA, but also supports further applications with regard to multicomputer systems. A first implementation of a CSSA operating system on a multicomputer system is currently in progress.

Inhaltsverzeichnis

1. Einführung in das Sprachkonzept von CSSA

CSSA (Computing System for Societies of Agents) ist eine anwendungsorientierte interaktive Programmiersprache, mit deren Hilfe asynchron parallele Prozesse auf Mehrrechnersystemen definiert werden können. Bei der nachfolgenden Beschreibung des CSSA-Konzepts wird auf die Mechanismen für sequentielle Berechnungen, die sich im wesentlichen am Standard heutiger höherer Programmiersprachen orientieren, nicht eingegangen.

Ein CSSA-Programm besteht aus autonomen abgeschlossenen Moduln (Agenten), die durch das Übersenden von Nachrichten miteinander kommunizieren. Jeder Agent führt sequentiell ablaufende Berechnungen aus, während Agenten untereinander asynchron nebenläufig arbeiten. Agenten besitzen keinen gemeinsamen Speicher; die einzige "Zugriffsmöglichkeit" eines Agenten auf einen anderen Agenten besteht darin, diesen als Empfänger einer Nachricht anzugeben. Damit ein Agent A einem Agenten B eine Nachricht zusenden kann, muß A den Agenten B kennen: A muß eine Bekanntschaft mit B besitzen. Ein solches Netz von CSSA-Agenten kann als gerichteter Graph dargestellt werden: die Agenten stellen dabei die Knoten und die Bekanntschaften die Kanten dar. Da während der Ausführung eines CSSA-Programms neue Agenten generiert werden, Agenten terminieren und Bekanntschaften gelöscht oder verschickt werden können, ist ein Agentennetz dynamisch.

Agenten werden aus einem Agentenschema (Script) durch einen Generator erzeugt. Scripts sind Typdefinitionen für Agenten und können über Script-Bekanntschaften eindeutig identifiziert werden. In einem Script werden Operationsdefinitionen zu Bündeln zusammengefaßt, die Facetten genannt werden. Operationen werden durch einen Operationsbezeichner, ein Entry-Pattern und einen Rumpf, der eine Folge von Anweisungen enthält, definiert. Operationsdefinitionen beschreiben Auswahl, Empfang und Verarbeitung von Nachrichten. Da für eine Operation mehrere Nachrichten zur Verarbeitung anstehen können, müssen die Eingänge von Operationsdefinitionen nicht nur Bindungsplätze zur Verfügung stellen, sondern auch Selektionskriterien für die Auswahl von Nachrichten angeben. Die Eingänge von Operationsdefinitionen sind daher durch den Operationsbezeichner selbst und durch ein Entry-Pattern definiert. Entry-Patterns beschreiben hierbei die Struktur der zulässigen Nachrichten und stellen Bindungsplätze etwa im Sinne formaler Parameterlisten bei Prozeduren bereit.

Nachrichten bestehen aus einem Operationsbezeichner und einer linearen Liste elementarer Datenobjekte. Beim Entry-Match werden Nachrichten durch Pattern-Match-Versuche gegen die Eingänge von Operationen ausgewählt. Nur bei einem erfolgreichen Entry-Match leitet eine Nachricht die Ausführung einer Operation ein. Seiteneffekte erfolgloser Match-Versuche werden rückgängig gemacht.

Ein Agent befindet sich immer nur in einer aktiven Facette und kann nur die Nachrichten verarbeiten, für die eine Operation in dieser Facette definiert ist. Facetten können durch lokale Deklarationen hierarchisch geschachtelt werden wie auch nebenein-

ander auf gleichem Schachtelungsniveau existieren.

Ein Agent kann durch die Auswertung des Generatorausdrucks

new <script-id> [<expression-list>]

generiert werden; <script-id> bezeichnet hierbei eine Script-Bekanntschaft. Aus der optionalen Ausdrucksliste wird eine initiale Information erzeugt, die dem neu zu generierenden Agenten übergeben wird. Nur bei einem gelungenen Versuch, diese Information durch einen Pattern-Match (Creation-Match) gegen das im Script angebbare Creation-Pattern zu übernehmen, liefert der Generatorausdruck eine Bekanntschaft mit dem neu erzeugten Agenten.

Die Nachrichten werden bei dem als Empfänger angegebenen Agenten in einem speziell verwalteten Puffer, der Mailbox, abgelegt. Der Ablaufzyklus eines Agenten sieht so aus, daß er nach seiner Generierung bzw. nach Beendigung einer Operation von der Mailbox solange neue Nachrichten anfordert, bis ein Entry-Match erfolgreich verläuft. In diesem Fall wird die Nachricht in der Mailbox gelöscht und der Agent führt die entsprechende Operation aus. Wird keine passende Nachricht gefunden, wird entweder eine in jeder Facette definierbare spezielle Operation, die idle operation, ausgeführt, falls diese vorhanden ist, oder auf das Eintreffen neuer Nachrichten gewartet. Für die Auswahlstrategie bei der Entnahme von Nachrichten aus der Mailbox gilt nur die Bedingung, daß sie fair sein muß. Für die Übertragungszeit der Nachrichten wird angenommen, daß sie endlich und positiv aber unbestimmt ist, d. h. insbesondere, daß sich Nachrichten überholen können.

Die in CSSA vorhandenen Kommunikationsmechanismen ermöglichen von einer völlig asynchronen bis hin zur handshaking-nahen Kommunikation einen flexibel programmierbaren Kopplungsgrad zwischen Sender und Empfänger. Die Ausführung der Anweisung

send <op-id> [<expression-list>] **to** <target>
[**reply to** <port>] ;

leitet die Übermittlung einer Nachricht an einen Zielagenten ein; fehlt die Ausdrucksliste, wird nur der Operationsbezeichner (op-id) verschickt. Der optional angebbare reply-Teil der send-Anweisung bewirkt, daß zusätzlich die Bekanntschaft des Sender-Agenten und eine Port-Adresse mitverschickt werden, damit eine Antwort an den richtigen Agenten zurückgesandt und eine Zuordnung der Antworten zu den send-Anweisungen ermöglicht wird. Eine derartige send-Anweisung verzögert den weiteren Ablauf des ausführenden Agenten nicht, d. h. es können mehrere send-Anweisungen hintereinander ausgeführt werden, ohne daß auf mögliche Antworten vorher abgesandter Nachrichten gewartet werden muß. Dadurch wird erreicht, daß Aufträge zum frühestmöglichen Zeitpunkt vergeben und die Ergebnisse erst zum spätestmöglichen Zeitpunkt empfangen werden können. Das Konstrukt zum Empfangen von Antworten sieht so aus:

```
receive <port> [<pattern>] do
        <body>
end receive ;
```

Wird in der Mailbox keine "passende" Antwort für diesen Port gefunden, wird mit der nächsten Anweisung fortgefahren. Soll auf eine Antwort gewartet werden, so gibt es hierfür ein wait-Konstrukt.

Wurde eine Operation im Sinne einer Auftragsbeziehung aktiviert (reply-Teil in der send-Anweisung wurde angegeben!), dann kann im Rahmen dieser Operation eine Antwort an den Auftraggeber zurückgesandt werden. Dies wird mit Hilfe der Anweisung

```
reply [<expression-list>] ;
```

erreicht. Bemerkenswert ist hierbei, daß der Empfänger keinen expliziten Zugriff auf die Bekanntschaft des Senders hat.

Eine präzisere Beschreibung des CSSA-Konzepts und eine Diskussion desselben ist in [3] zu finden.

2. Das CSSA-System

Das CSSA-System besteht im wesentlichen aus drei Komponenten:

- Compiler
- Interface-Agent (IF-Agent)
- Betriebssystem (BS)

Der Compiler übersetzt unabhängig voneinander Scripts und legt diese in einer Script-Bibliothek ab.

Der Interface-Agent (IF-Agent) stellt die Benutzerschnittstelle dar; über ein Terminal soll der Benutzer mit Hilfe des IF-Agenten direkt mit den anderen Agenten kommunizieren können. Dadurch ist ein Benutzer wie ein "normaler" Agent ein Bestandteil des Agentennetzes. Über den IF-Agent kann man ein Agentennetz generieren, starten und steuern. Zu diesem Zweck ist der IF-Agent als Interpreter realisiert, der syntaktisch abgeschlossene CSSA-Anweisungen sofort ausführt. So kann man im IF-Agenten auch Operationen definieren, die wie bei allen anderen Agenten durch Nachrichten und einen erfolgreichen Entry-Match aktiviert werden können. Der Ablaufzyklus des IF-Agenten unterscheidet sich von dem "normaler" Agenten insofern, als daß der Benutzer die Möglichkeit hat, sich Nachrichten aus der Mailbox anzusehen und festzulegen, welche Nachricht als nächste bearbeitet werden soll. Nach Beendigung einer Operation erhält der Benutzer die Kontrolle wieder zurück. Neben dem vollständigen CSSA-Sprachumfang stehen dem Anwender zusätzlich noch einige Testhilfen zur Verfügung.

Aus den vorhergehenden Beschreibungen lassen sich jetzt grob die Aufgaben eines CSSA-Betriebssystems zusammenstellen:

- Nachrichtentransport
- Mailbox-Verwaltung
- Scheduling (Prozessorverwaltung, Generieren/Terminieren von Agenten)
- Verwaltung des E/A-Systems
- Unterstützung des IF-Agenten (z. B. beim Testen)

3. Der Aufbau des CSSA-Betriebssystems

Im Gegensatz zu vergleichbaren verteilten Betriebssystemen (z. B. [1],[4],[5],[8],[12]), die sich an einer vorhandenen Hardwarestruktur orientieren, wurde hier ein BS entwickelt, in dem hardwareunabhängig von einem vorgegebenen Berechnungsmodell und einer Programmiersprache ausgegangen wurde. Dies führte zu einem BS, daß vollständig in das Gesamtkonzept von CSSA integriert ist. Darüberhinaus lassen sich durch diesen Ansatz sogar die Leistungen konventioneller Betriebssysteme heutiger Großrechner erreichen. Da CSSA für universelle Multicomputersysteme gedacht ist und somit die BS-Funktionen auf verschiedenen Prozessoren verfügbar sein müssen, ergaben sich für das CSSA-BS folgende naheliegende Forderungen:

- hardwareunabhängige Beschreibung des Systems
- klare übersichtliche Konzeption mit präzise abgegrenzten BS-Funktionen
- mit CSSA-Sprachmitteln implementierbare BS-Schnittstellen

Während der ersten Überlegungen zu einem CSSA-BS kamen wir zu der Entscheidung, das BS soweit wie möglich in CSSA selbst zu beschreiben. Bei elementaren und einigen CSSA-typischen BS-Funktionen (z. B. Treiber für E/A-Geräte bzw. Mailbox-Verwaltung) reichen die sprachlichen Möglichkeiten von CSSA ohne Erweiterungen nicht aus. Es wurde daher beschlossen, nur die für einen Benutzer sichtbaren BS-Schnittstellen in Form von CSSA-Agenten zu realisieren, während die übrigen BS-Funktionen als nicht in CSSA beschreibbare Prozesse bereitgestellt werden. Abgesehen von der sprachlichen Realisierung ist der vereinfachte Kommunikationsmechanismus der wesentlichste Unterschied zwischen Agenten und BS-Prozessen. Da die Mailbox und der Pattern-Match für das BS nicht erforderlich sind, wurde auf sie sowohl aus Effizienzgründen als auch wegen einer einheitlichen Nachrichtenübermittlung im nichtsichtbaren BS-Teil verzichtet. CSSA-send-, -receive-, und -wait-Anweisungen existieren mit einer analogen Semantik; anstelle der Bekanntschaft in der CSSA-send-Anweisung wird beim BS-send ein Os-Port (Operating System Port - nicht zu verwechseln mit CSSA-Ports) angegeben. Beim BS-send gibt es keinen optionalen reply-Teil mehr; zur CSSA-reply-Anweisung gibt es bei den BS-Prozessen kein Äquivalent. Solche Os-Ports sind Nachrichtenwarteschlangen, die nach der FIFO-Strategie abgearbeitet werden. Wie der Nachrichtenaus-

tausch zwischen BS-Prozessen abläuft, wird später noch genauer erläutert.

Zunächst sollen die einzelnen BS-Agenten bzw. BS-Prozesse vorgestellt und ihre Aufgaben spezifiziert werden. Das BS ist dabei in Form eines Schichtenmodells konzipiert, wobei die den oberen Schichten zugehörigen Prozesse Aufträge an untergeordnete Prozesse verteilen oder die Ausführung niedergeordneter Prozesse steuern und überwachen:

[9)] vom Anwender definierte Agenten (hier nur der Vollständigkeit wegen erwähnt)
8) Klasse von System-Agenten als BS-Schnittstelle: IF, OS, INPUT, OUTPUT, PRINTER, DIRECT
7) Prozeß- und Prozessor-Verwaltung (Communications Monitor (CM) / Scheduler)
6) Dateiverwaltungssystem (File Manager)
5) Verwaltung geräte-spezifischer Datenstrukturen, Steuerung peripherer Geräte (I/O-Handlers, Spoolers)
4) Mailbox-Verwaltung (Mailbox Manager)
3) lokaler CSSA-BS-Kern (Speicherverwaltung, Multiprogramming, Prozessorsteuerung, Laufzeitsystem)
2) CCN (CSSA Communication Network)
1) BCS (Basic Communication System)

Die Klasse der System-Agenten (zu 8.) stellt die einzige auf Benutzerebene sichtbare Schnittstelle zum BS dar. Von jedem Agenten können beliebig viele System-Agenten erzeugt werden. Eine Ausnahme bilden hierbei OS und in der BS-Grundversion auch IF: sie werden bereits beim Systemstart generiert. Aus der Sicht des Anwenders sind alle "BS-Kommandos" als Operationen im OS-Agenten definiert. Der OS-Agent dient als Schnittstelle zum BS, er entscheidet, welcher BS-Agent bzw. BS-Prozeß für die Bearbeitung eines Auftrages zuständig ist und leitet diesen in möglicherweise aufbereiteter Form weiter.

Den System-Ein-/Ausgabe-Agenten (vgl. 8.) werden Nachrichten wie OPEN/CLOSE, READ/WRITE etc. zugeschickt; dort werden die ein- bzw. auszugebenden Daten vorverarbeitet (Format-gesteuerte Ein-/Ausgabe, Fehleruntersuchung, Blockung von Sätzen, Aufbereiten zum Drucken etc.) und blockweise zum File Manager oder evtl. direkt zu einem I/O-Handler übertragen. Eine detaillierte Beschreibung des CSSA-Ein-/Ausgabe-Systems, so wie es sich für einen Benutzer darstellt, ist in [10] zu finden.

Die Communications Monitors (CM) regeln den Nachrichtentransport zwischen Agenten und nehmen die Umcodierung von Bekanntschaften in Hardware-Adressen vor. Zusätzlich werden hier Informationen über den Nachrichtenaustausch gesammelt, die der Scheduler z. B. für die Auswahl eines auszulagernden Agenten anfordern kann. Der Scheduler verwaltet darüberhinaus alle Prozessoren und lädt bei "new"-Anforderungen die entsprechenden Agenten. Von einem CM können mehrere Agenten betreut werden, während der Scheduler beispielsweise mehrere CM betreut. Die Konfiguration dieser beiden Arten von BS-Prozessen ist frei wählbar. Es wurden hier zwei ver-

schiedene Typen von BS-Prozessen eingeführt, um spezielle seitens der Hardware vorgegebene Kommunikationskanäle für den Nachrichtentransport besser nutzen und eventuelle Engpässe beim Scheduler vermeiden zu können. Wesentlicher Unterschied zwischen den CM und dem Scheduler ist, daß die CM nur über die notwendigen Informationen von den Agenten verfügen, die ihnen zugeteilt sind, während der Scheduler das komplette System kontrolliert und nur alleine Funktionen wie Laden eines Agenten, Auslagern etc. durchführen kann.

Der File Manager verwaltet das vollständige E/A-System des Rechners. Er alleine hat Zugriff auf einen Dateienkatalog und kontrolliert die Zuordnung logischer Dateinamen zu physikalischen Dateinamen oder zu bestimmten E/A-Geräten. So wird z. B. jede OPEN-Anweisung zum File Manager weitergeleitet und dieser untersucht, ob eine Datei vorhanden ist, ob eine Benutzungsberechtigung vorliegt, ob die Datei-Attribute verträglich sind oder ob ein angefordertes E/A-Gerät verfügbar ist. Werden keine Fehler festgestellt, wird für die zu eröffnende Datei ein Kontrollblock angelegt, eine interne Dateinummer vergeben und diese neben einigen anderen internen Informationen an den Auftraggeber zurückgeschickt. Der Zugriff auf eine Magnetplatte oder einen Drucker wird nicht vom File Manager, sondern von den Gerätesteuerprozessen (I/O-Handlers) vorgenommen.

Der Mailbox Manager verwaltet die Mailbox eines Agenten. Auf Anforderung liefert er eine Nachricht und wartet, bis ihm der Agent gemeldet hat, ob der Entry-Match erfolgreich war oder nicht; im ersten Fall wird die Nachricht gelöscht, im zweiten Fall wird eine neue Nachricht bereitgestellt oder auf das Eintreffen neuer Nachrichten gewartet. Bei der Auswahl der Nachrichten muß immer fair vorgegangen werden, d. h. Nachrichten, für die ein Match-Versuch nicht erfolgreich verlief, müssen nach einem bestimmten Verfahren immer wieder angeboten werden. Um hier den Aufwand zu reduzieren, nimmt der Mailbox Manager in einem begrenzten Rahmen eine Vorauswahl der Nachrichten vor, d. h. er versucht nur Nachrichten zu liefern, die eine Chance auf einen erfolgreichen Entry-Match haben. Die genaue Verfahrensweise des Mailbox Managers im Hinblick auf die Nachrichtenauswahl ist in [16] beschrieben.

Während der Mailbox Manager nur auf den Prozessoren verfügbar sein muß, auf denen Agenten geladen sind, müssen die untersten drei Schichten des BS auf jedem Prozessor vorhanden sein. Der lokale CSSA-BS-Kern unterstützt ein lokales Multiprogramming, ein Speicherverwaltungssystem, lokale Vorbereitungen für das Laden oder Auslagern von Prozessen und die Kommunikation zwischen lokalen Prozessen (die Verwaltung der Os-Ports). Auch das für die Agenten notwendige Laufzeitsystem ist in Form von reentrant programmierten Routinen vorhanden.

Der CCN-Prozeß hat eine zentrale Funktion auf einem Prozessor. Er ist dafür verantwortlich, daß Nachrichten an die richtige Instanz geschickt werden. Die komplette lokale Inter-Prozeß-Kommunikation wird über den CCN-Prozeß abgewickelt. Ein solcher zentraler Prozeß ist insofern notwendig, da alle Prozesse und

Agenten nur logische Adressen kennen und daher nicht wissen, auf welchen Prozessoren sich die anderen Prozesse befinden. Die für das Weiterleiten von Nachrichten erforderlichen Informationen entnimmt der CCN-Prozeß den beiden Tabellen MRT (Message Routing Table) und CCT (Communication Channel Table). MRT enthält in jeder Zeile eine logische Prozeßadresse, eine lokale Kanalnummer (LCC-No - Local Communication Channel Number) und eine optionale Anfangsadresse einer Liste von BS-Nachrichtentypen. Jeder eintreffenden Nachricht wird vom CCN-Modul mittels MRT in Abhängigkeit von dem in der Nachricht angegebenen Empfänger und evtl. von dem Nachrichtentyp eine lokale Kanalnummer zugeordnet. Ist der Empfänger nicht in MRT eingetragen oder ist in der Nachricht kein Empfänger angegeben (das ist tatsächlich möglich - siehe dazu Beispiel zu Abb. 3), so wird eine Default-Kanalnummer (Standardeintrag in MRT: ANY/<LCC-No>) vergeben. Die Tabelle CCT ordnet den einzelnen Kanälen Os-Port-Adressen zu; aus diesen Angaben kann der CCN-Modul die Zieladresse ermitteln. Die CCN-Prozesse aller Prozessoren bilden zusammen das CCN.

Waren die bis jetzt beschriebenen BS-Prozesse vielleicht mit Ausnahme der I/O-Handler noch weitgehend unabhängig von der Hardware, so gilt das nicht mehr für die BCS-Prozesse. Sie realisieren die Inter-Prozessor-Kommunikation und die Steuerung von E/A-Geräten auf unterster Ebene. Auch hier wird das BCS durch die Gesamtheit aller BCS-Prozesse repräsentiert.

Zur Struktur von Nachrichten soll nur gesagt werden, daß im wesentlichen drei Übertragungsprotokolle zu unterscheiden sind:

- CSSA-Nachrichten-Protokoll
- BS-Nachrichten-Protokoll
- BCS-Protokolle

Alle Protokolle werden, soweit sie nicht die Hardware betreffen, in [11] beschrieben werden.

4. Beispiele zum Ablauf von Betriebssystem-Funktionen

Einige Beispiele sollen das Zusammenspiel der einzelnen BS-Prozesse etwas mehr verdeutlichen. Abb. 1 zeigt dazu eine mögliche Konfiguration von Agenten und BS-Prozessen auf einem Prozessor. Die vom Benutzer programmierten Agenten sind mit UDA 1 bis UDA 3 (User Defined Agent) bezeichnet; MB 1 - MB 3 benennen die zugehörigen Mailboxes. Die einfachen Striche geben die Nachrichtenwege zwischen den Prozessen mit den Os-Port-Adressen an; die doppelten gestrichelten Linien sollen lediglich Zugriffsmöglichkeiten andeuten. Die links über dem Mailbox Manager angebrachte Bezeichnung L04 gibt die logische Adresse dieses Prozesses an; RTS bezeichnet das Laufzeitsystem (Run Time System). Abb. 2 zeigt die für das Verständnis der zu Abb. 1 gehörenden Beispiele erforderlichen Eintragungen in den Tabellen MRT und CCT.

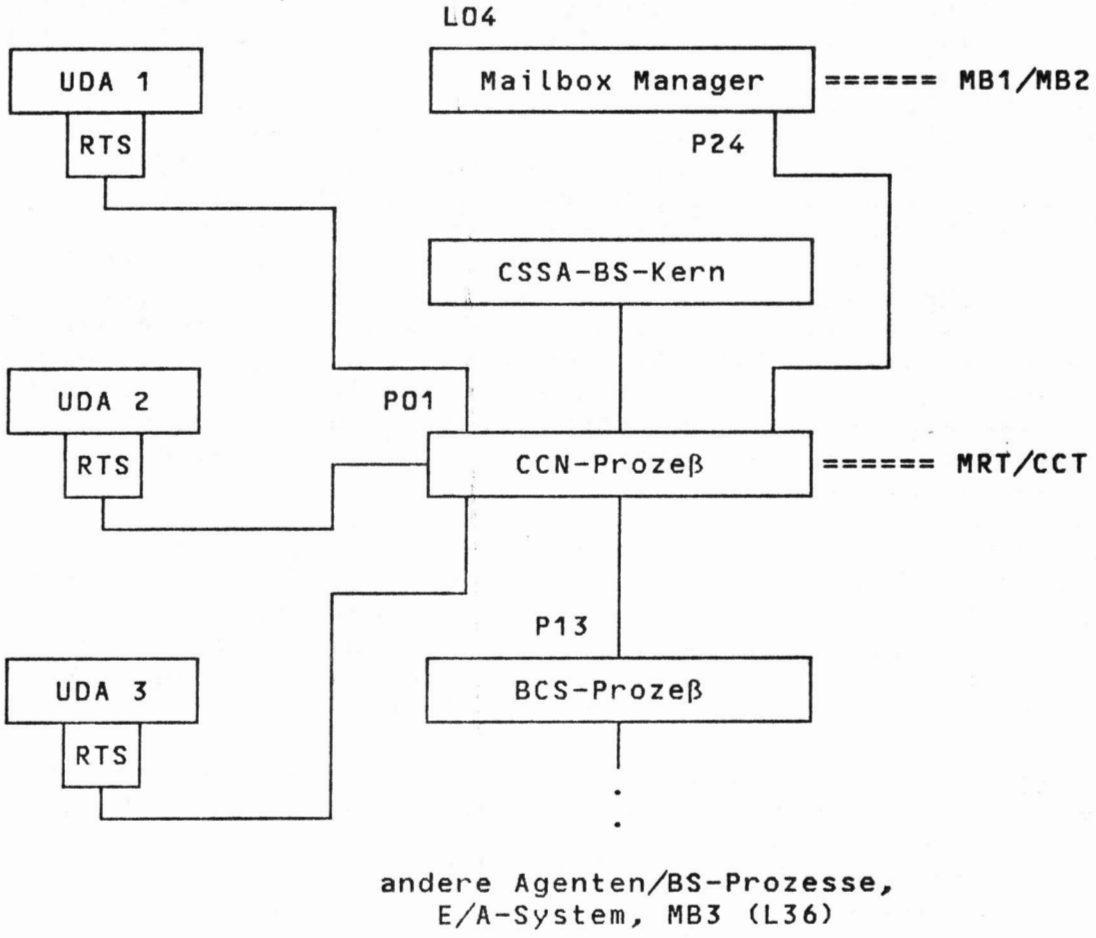


Abb. 1: Mögliche Konfiguration von Agenten und BS-Prozessen auf einem Prozessor

Wir gehen davon aus, daß die drei Agenten UDA 1, UDA 2 und UDA 3 zu einem sich in Ausführung befindlichen CSSA-Programm gehören. UDA 1 hat eine Operation beendet und das Laufzeitsystem fordert mit Hilfe der BS-Funktion GETMSG von seiner Mailbox eine neue Nachricht an. Jedem Agenten wird bei seiner Generierung die logische Adresse des für ihn zuständigen Mailbox Managers mitgeteilt. Diese Adresse ist auch in der GETMSG-Anforderung enthalten. Das Laufzeitsystem schickt jede Nachricht an den Os-Port P01 (CCN-Modul). Der CCN-Prozeß liest die logische Adresse des Mailbox Managers (L04) und weist die lokale Kanalnummer C2 zu. Aus der Tabelle CCT kann die Os-Port-Adresse entnommen werden, an die die Nachricht geschickt werden soll (P24/Mailbox Manager).

Angenommen, UDA 3 wollte die gleiche Nachricht an seinen Mailbox Manager (L36) schicken; in diesem Fall würde die Nachricht über den CCN-Prozeß und Kanal C6 an den Os-Port P13 geleitet werden. Der BCS-Prozeß wäre jetzt für die Übertragung zu einem anderen Prozessor verantwortlich.

Message Routing Table (MRT)

Receiver	LCC-No
...	..
L04	C2
L36	C6
...	..
ANY	..

Communication Channel Table (CCT)

LCC-No	Os-Port
..	...
C2	P24
C6	P13
..	...
..	...

Abb. 2: Die Tabellen MRT und CCT zu Abb. 1

Einige weitere Beispiele sollen veranschaulichen, wie aufwendigere BS-Funktionen realisiert werden, die auf mehrere Prozessoren verteilt sind. Abb. 3 zeigt einen Ausschnitt aller an einem CSSA-Programm beteiligten Agenten und BS-Prozesse. Um die Übersichtlichkeit zu wahren, wurde auf die Darstellung der lokal auf einem Prozessor vorhandenen BS-Prozesse verzichtet. Es wird bei den nachfolgenden Beispielen ferner davon ausgegangen, daß sich jeder angegebene Prozeß auf einem eigenem Prozessor befindet; die Linien in Abb. 3 geben dabei lediglich logische Kommunikationskanäle an.

Im ersten Beispiel zu Abb. 3 möchte der Agent UDA 2 eine Ausgabe auf einem Drucker vornehmen. Er schickt dazu eine "new"-Anforderung an das BS, das ihm einen PRINTER System-Agenten zur Verfügung stellen soll. Die "new"-Nachricht wird über die CCN-Prozesse und den Prozessor, auf dem CM 1 (Communications Monitor) geladen ist, zum Scheduler übertragen. Dafür ist nicht erforderlich, daß UDA 2 die logische Adresse des Schedulers kennt; der Übertragungsweg wird allein durch den ANY-Eintrag in MRT und die daran geknüpften Nachrichtentypen gefunden: alle CCN-Prozesse "wissen" daher, wohin eine "new"-Nachricht geschickt werden muß. Der Scheduler beauftragt daraufhin den File Manager, den zum PRINTER-Agenten gehörigen Agenten-Kontrollblock (ACB - Agent Control Block) zu beschaffen. Der ACB enthält Informationen, die der Scheduler zum Laden eines Agenten braucht (z. B. Prozessortyp, Codelänge, geschätzter dynamischer Speicherplatzbedarf, Einsprungadresse für den Creation Match u. ä.). Anhand der Scheduling-Tabelle wird ein freier Prozessor ausgewählt (in Abb. 3 sei das der mit <PRINTER> gekennzeichnete Prozeß bzw. Prozessor) und dem dortigen CSSA-BS-Kern mitgeteilt, daß ein Agent geladen werden soll. Dem E/A-System wird der Auftrag erteilt, den Agenten PRINTER vom Hintergrundspeicher zu lesen und an den ausgewählten Prozessor zu schicken. Der CSSA-BS-Kern lädt den Agenten in einen geeigneten Speicherbereich. Der File Manager und der BS-Kern melden dem Scheduler, wenn der Ladevorgang beendet wurde. Daraufhin erhält der BS-Kern die bei der Generierung angebbare initiale Information und verzweigt zur Einsprungadresse des Agenten für den Creation

Match. War dieser erfolgreich, wird der Scheduler darüber informiert, vergibt eine logische Adresse (in diesem Fall eine Bekanntschaft) und sendet sie dem Auftraggeber der "new"-Anforderung und dem BS-Kern zu. Der BS-Kern startet daraufhin den neuen Agenten.

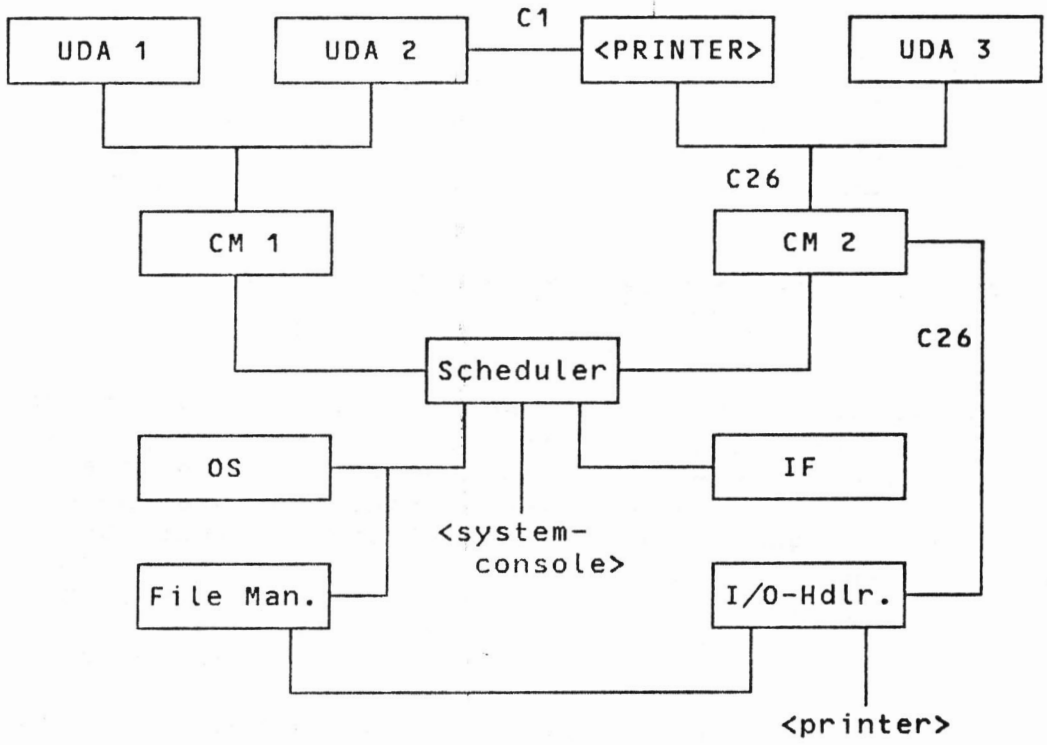


Abb. 3: Beispiel einer Agenten-/BS-Prozeß-Konfiguration

Wäre kein Prozessor frei gewesen, hätte der Scheduler die CM's beauftragt, einen oder mehrere auszulagernde Agenten auszuwählen. Die CM's hätten sich die notwendigen Informationen von den BS-Kernen der ihnen zugeteilten Prozessoren besorgt und jeweils einen oder mehrere Agenten ausgewählt, so daß genügend Speicherplatz für den neu zu ladenden Agenten vorhanden wäre. Die Ausführung der zum Auslagern vorgeschlagenen Agenten wird dabei unterbrochen. Der Scheduler kann sich jetzt seinerseits nach einem bestimmten Algorithmus für einen oder mehrere Prozesse entscheiden. Die nicht auszulagernden Agenten werden wieder gestartet und nur die ausgewählten Agenten werden über das E/A-System auf einen Hintergrundspeicher übertragen. Auskunft über den Zustand, in dem sich ein Agent befindet, erhalten die BS-Prozesse immer über den Agenten-Kontrollblock (ACB). Er wird beim Generieren eines Agenten kopiert und verbleibt bei diesem Agenten bis zu dessen Terminierung. Während der Programmausführung aktualisieren die BS-Prozesse den ACB was z. B. interne Daten wie belegter dynamischer Speicherbereich, Registerinhalte u. ä. anbelangt. Alle für das erneute Einlagern eines ausgelagerten Agenten benötigten Informationen sind daher in

dem mitabgespeicherten ACB enthalten.

Will der Agent UDA 2 nach einer erfolgreichen OPEN-Operation eine größere Menge von Datensätzen ausgeben, müßten für jede Zeile die Stationen CM 1 - Scheduler - CM 2 - PRINTER und für jeden Datenblock die Stationen PRINTER - CM 2 - Scheduler - File Manager - I/O-Handler - <printer> durchlaufen werden, bis die Daten tatsächlich auf dem Drucker erscheinen. Sollte die Hardware keine anderen Übertragungsmöglichkeiten zulassen, bliebe natürlich nichts anderes übrig. In höchstem Maße ineffizient wäre diese Vorgehensweise jedoch dann, wenn eine direkte Übertragungsstrecke zwischen den am eigentlichen Druckvorgang beteiligten Prozessoren vorhanden wäre (z. B. alle Prozessoren sind über einen gemeinsamen Systembus direkt adressierbar). Um in einem solchen Fall die Nachrichtenübermittlung zu beschleunigen, kann der Scheduler über Vergabe globaler Kommunikationskanäle die nicht beteiligten Prozessoren umgehen. Die globalen Kommunikationskanalnummern werden von den CCN-Moduln auf Anweisung des Schedulers in die für die Nachrichtenübermittlung erforderlichen Tabellen MRT und CCT eingetragen; der Scheduler verwaltet die globalen Kommunikationskanäle in einer eigenen Tabelle. Bearbeitet ein CCN-Prozeß eine Nachricht, für deren Empfänger oder deren Typ in MRT eine globale Kommunikationskanalnummer enthalten ist (BS-Prozesse können globale von lokalen Kommunikationskanalnummern unterscheiden), wird die globale Kanalnummer in die BS-Nachricht eingetragen. Bei von anderen Prozessoren kommenden Nachrichten wird zuerst untersucht, ob sie eine globale Kanalnummer enthalten; wenn ja, wird die Nachricht gemäß der CCT-Eintragung gleich weiter verschickt (MRT ist nicht notwendig). Ist keine globale Kanalnummer vorhanden, wird nach dem oben beschriebenen Verfahren vorgegangen. In dem betrachteten Beispiel könnte der Scheduler die globalen Kanalnummern C1 und C26 vergeben, über die eine direkte Verbindung zwischen den Prozessen hergestellt werden könnte, wenn die Hardware dies zuließe. Im in Abb. 3 dargestellten Fall soll z. B. eine direkte Verbindung zwischen PRINTER und dem I/O-Handler nicht möglich sein. Nach Beendigung einer Datenübertragung durch eine CLOSE-Operation oder das Auslagern oder Terminieren von Agenten werden die globalen Kanäle wieder aufgehoben, indem die CCN-Prozesse durch den Scheduler angewiesen werden, die entsprechenden Eintragungen in MRT und CCT zu löschen.

Die BS-Prozesse können ebenfalls Generatoren anwenden. Es ist daher nicht notwendig, daß zu jedem Zeitpunkt alle BS-Komponenten geladen sind. Wenn z. B. in einer Implementierung einer erweiterten Version des CSSA-BS ein System-Agent CRT vorhanden wäre, der Operationen für die Formatierung eines Bildschirms bereitstellte, könnte der File Manager den Scheduler über eine "new"-Nachricht veranlassen, bei Benutzung von CRT einen speziellen I/O-Handler zu laden, und dem Handler das gewünschte Terminal zuweisen.

5. Ausbaufähigkeit des CSSA-Betriebssystems

Bis zu diesem Abschnitt wurde die Grundversion des CSSA-BS vorgestellt. Ein entscheidender Nachteil ist hierbei, daß der IF-Agent bereits bei der Systeminitialisierung erzeugt wird und daher nur ein Benutzer an dem Rechner arbeiten kann. In einem Mehrrechnersystem wäre es jedoch wünschenswert, daß mehrere Personen die Anlage gleichzeitig benutzen können. Eine kleine Erweiterung der BS-Grundversion macht dies möglich: Jedem Terminal wird nach dem Systemstart ein I/O-Handler zugewiesen, der nur eine genau vorgeschriebene LOGON-Eingabe am Terminal zum Scheduler weiterleitet. Dort wird in einer Benutzertabelle AUT (Active User Table) nachgesehen, ob für das absendende Terminal ein Eintrag vorliegt. Ist das der Fall, wird die Nachricht an die in AUT angegebene Instanz gesandt. Fehlt eine Eintragung, wird mittels eines "new"-Ausdrucks ein IF-Agent generiert, diesem eine Bekanntschaft zugewiesen, ein globaler Kommunikationskanal zwischen dem IF-Agenten und dem I/O-Handler eingerichtet und eine entsprechende Eintragung in AUT vorgenommen. Dem IF-Agenten wird eine Initialisierungsnachricht zugesandt und er selbst schreibt eine Meldung über das erfolgreiche LOGON an den Benutzer.

Da alle CSSA-Agenten den IF-Agenten kennen, muß lediglich noch sichergestellt werden, daß Agenten verschiedener "disjunkter" Netze nur ihren eigenen IF-Agenten kennen. Weil der IF-Agent immer den Ursprung eines Agentennetzes darstellt, reicht es aus, die Bekanntschaft des IF-Agenten mit jeder "new"-Nachricht mitzuschicken und sie in den Agenten-Kontrollblock eines jeden Agenten bei dessen Erzeugung zu schreiben. Jeder Agent kann jetzt die Bekanntschaft seines IF-Agenten aus seinem Agenten-Kontrollblock entnehmen.

Ein dem LOGON entsprechendes LOGOFF-Kommando zum Abbruch einer CSSA-Sitzung ist nicht erforderlich: es wird bereits durch die terminate-Anweisung beim IF-Agenten realisiert. Das BS sorgt dafür, daß die Einträge in der Benutzertabelle gelöscht und alle Agenten des betroffenen Netzes entfernt werden. Welche Agenten zu welchem Netz gehören, kann der Scheduler der Scheduling-Tabelle entnehmen: für jede Bekanntschaft wurde auch die zugehörige IF-Bekanntschaft mit abgespeichert.

Weitere Überlegungen im Hinblick auf die Ausbaufähigkeit des CSSA-BS führten zu dem zuerst durchaus überraschenden Ergebnis, daß sich die Dienstleistungen konventioneller BS von Großrechnern leicht in das CSSA-BS integrieren lassen. Gewissermaßen von selbst ist das Starten von Hintergrundjobs (foreground initiated background job) möglich: vom IF-Agent aus kann ein Agent geladen und diesem durch eine Nachricht ein Auftrag erteilt werden. Während der Auftragsbearbeitung kann der IF-Agent beliebige CSSA-Anweisungen ausführen. Hat der Agent seinen Auftrag beendet, kann er dem IF-Agenten eine Meldung hierüber und evtl. das Ergebnis zusenden.

Der Nachteil, daß durch eine terminate-Anweisung des IF-Agenten alle zum Netz gehörenden Agenten entfernt werden, kann durch die

Einführung einer echten Stapelverarbeitung auch noch beseitigt werden. Wie dies vorgenommen werden kann, soll nur kurz skizziert werden: z. B. einem Kartenleser wird ein ähnlicher I/O-Handler wie einem Terminal zugeordnet. Nach einer LOGON-Eingabe, die in die erste Lochkarte eines Batch-Jobs gestanzt sein muß, werden über den Scheduler und das E/A-System alle folgenden Karten bis zu einer terminate-Anweisung in eine Spool-Datei geschrieben. Die Namen dieser Dateien werden von einem BS-Prozeß Spooler verwaltet, der, gesteuert durch den Scheduler und Eingaben über die System-Konsole, einen oder mehrere IF-Agenten generiert und Kommunikationskanäle so festlegt, daß die Eingaben für den einzelnen IF-Agenten von einer Spool-Datei gelesen werden und die Ausgaben auf eine zweite Datei erfolgen. Der IF-Agent führt alle Anweisungen genauso aus, als kämen sie vom Terminal. Nach Ausführung einer terminate-Anweisung wird der IF-Agent entfernt, die Ausgabedatei einem Druck-Prozeß übergeben, der die ihm zugewiesenen Dateien der Reihe nach ausdruckt, und der Spooler-Prozeß startet den gleichen Zyklus für den nächsten "Batch-Job".

Auch in anderen Sprachen definierte Prozesse können prinzipiell durch ein CSSA-BS ausführbar gemacht werden. Wenn auf einer Anlage z. B. bereits ein Fortran-Compiler vorhanden ist, der mit der Unterstützung eines ebenfalls vorhandenen BS läuft, kann ein System-Agent FORTRAN definiert werden, der die Umgebung des alten BS durch Prozeduren simuliert, die auf oberster Script-Ebene bereitgestellt werden, und in eine CSSA-BS-verträgliche Form konvertiert. Der Fortran-Compiler würde als Operation "compile" in den Agenten FORTRAN "eingebaut". Durch eine Anweisung der Art

send compile (<files,parameters>) **to** FORTRAN ;

könnte nach Erzeugen des FORTRAN-Agenten die Übersetzung eines Fortran-Programms gestartet werden.

Auf die Möglichkeit, den CCN-Prozessen zusätzlich die Aufgabe zu erteilen, benachbarte Prozessoren zu überwachen, dadurch Prozessorausfälle zu erkennen und zu versuchen, den verursachten Schaden lokal zu begrenzen, soweit das aufgrund der aktiven Agentennetze möglich ist, soll im Rahmen dieser Arbeit nur hingewiesen werden.

Die für die soeben beschriebenen Erweiterungen erforderlichen Prozesse sind relativ einfach bereitzustellen und lassen sich problemlos in die vorhandene BS-Umgebung einfügen.

Ein entscheidender Vorteil eines so erweiterten CSSA-BS liegt darin, daß es im gesamten System nur eine Sprache gibt: CSSA. Sie ist die Dialogsprache am Terminal, die Auftragssprache für die Stapelverarbeitung und natürlich die Anwendersprache. Durch diese homogene BS-Schnittstelle läßt sich die Lösung üblicherweise BS-spezifischer Aufgaben für spezielle Anwendungen sehr leicht auf Benutzerebene verlagern. So kann z. B. ein eigener File Manager programmiert werden, der die gesamte Ein-/Ausgabe innerhalb eines Agentennetzes abwickelt. Probleme wie exklusive Benutzung diverser peripherer Geräte gemäß unterschiedlicher Prioritäten einzelner

Agenten können so anwendungsspezifisch gelöst werden.

6. Bemerkungen zum Implementierungsaufwand

Der auf den ersten Eindruck hin vielleicht immens erscheinende Implementierungsaufwand kann erheblich reduziert werden, wenn man Funktionen bzw. Leistungen bereits vorhandener BS verwendet. Ein gutes Beispiel hierzu sind der File Manager und die I/O-Handlers; sie brauchen nur in Bezug auf die CSSA-BS-eigene Inter-Prozeß-Kommunikation angepaßt zu werden. Man kann weiterhin versuchen, vorhandene oder von einigen Firmen angebotene BS-Kerne zur Realisierung der lokalen Kommunikation zu benutzen. Für Mehrprozessorsysteme, die auf der Basis von 8- oder 16-Bit Mikroprozessoren aufgebaut sind, gibt es für einige weit verbreitete Prozessortypen (8080/8085, Z80, 8086 etc.) bereits recht leistungsfähige BS-Kerne, mit denen sich die Verwaltung der os-ports auf einfache Weise lösen läßt (z. B. INTEL's iRMX 80/iRMX 86).

7. Schlußbemerkungen

Hier wurde ein BS für eine spezielle Sprache und kein BS für eine vorgegebene Hardwarestruktur entwickelt, wie dies bei vergleichbaren verteilten BS der Fall ist. Durch eine klare und übersichtliche Verteilung von BS-Funktionen auf einzelne, weitgehend autonome Prozesse kann das CSSA-BS so konfiguriert werden, daß es auf unterschiedlichen Mehrrechnersystemen implementierbar ist und dazu noch eine effiziente Ausführung von CSSA-Programmen erlaubt. Das CSSA-BS wurde so in das CSSA-Sprachkonzept integriert, daß durch eine vollständig mit CSSA-Sprachmitteln implementierte Benutzerschnittstelle vielseitige Anwendungsmöglichkeiten eröffnet werden. Ein erweitertes CSSA-BS kann ohne grundsätzlich neue interne Mechanismen allen Komfort heutiger BS von Großrechenanlagen anbieten und hat den wesentlichen Vorteil, daß die Dialog-, die Batch-Auftrags- und die Anwendersprache identisch sind. Obwohl das BS in diesem Fall ganz auf das CSSA-Konzept abgestimmt ist, können bereits vorhandene in anderen Sprachen definierte Prozesse ausführbar gemacht werden und sind somit auch weiterhin verfügbar.

Literaturverzeichnis

1) W. Bauer; Der Betriebssystemkern des FFM-Multicomputersystems; Forschungsinstitut für Funk und Mathematik (FGAN), Wachtberg-Werthoven, Jan. 1979, (Nr. 276)

2) P. H. Enslow; Multiprocessor Organisation - A Survey; Computing Surveys, Vol. 9, No. 1, March 1977

3) H. L. Fischer, P. Raulefs, H. Voss; The Programming Language CSSA for Multicomputer-Systems; SEKI-Report, Inst. f. Informatik III, Universität Bonn, 1981

4) G. Goos et al.; The Operating System BSM - Viewed as a Community of Parallel Processes; RZ der TU München, Rep. 7208, March 1972

5) A. K. Jones et al.; StarOS, A Multiprocessor Operating System for the Support of Task Forces; Proc. 7th Symp. on Operating System Principles; ACM SIGOPS, 1979

6) A. K. Jones, P. Schwarz; Experiences Using Multiprocessor Systems - A Status Report; CMU 1979 (CMU-CS-79-146)

7) R. M. Metcalfe, D. R. Boggs; Ethernet: Distributed Packet Switching for Local Computer Networks; CACM, July 1976

8) J. K. Ousterhout; Partitioning and Cooperation in a Distributed Multiprocessor Operating System; CMU 1980 (CMU-CS-80-112)

9) R. F. Rashid; An Inter-Process Communication Facility for UNIX; CMU 1980 (CMU-CS-80-124)

10) G. Schüler, D. Wybranietz; Bericht über den Entwicklungsstand der CSSA-Umgebung; Interner Bericht, Inst. f. Informatik III, Univ. Bonn, 1980

11) G. Schüler, D. Wybranietz; Entwurf und Teilrealisierung eines Betriebssystems auf einer Multicomputer-Architektur; Diplomarbeit, Inst. f. Informatik III, Univ. Bonn, (in Vorbereitung)

12) W. D. Sincoskie, D. J. Farber; SDOS/OS: A Distributed Operating System for the IBM Series/1; ACM SIGOPS, July 1980

13) M. H. Soloman, R. A. Finkel; Roscoe: A Multi-Microcomputer Operating System; University of Wisconsin-Madison, Techn. Rep. No 321, May 1978

14) L. Svoboda, B. Liskov, D. Clark; Distributed Computing Systems: Structure and Semantics; MIT March 1979

15) A. S. Tannenbaum, S. J. Mullender; An Overview of the AMOEBA Distributed Operating System; ACM SIGOPS, July 1981

16) H. Voss; Beschreibung der Mailbox-Verwaltung als Teil des Betriebssystemkerns für eine CSSA-Implementierung. MEMO SEKI-BN-80-08, Inst. f. Informatik III, Univ. Bonn, 1980

17) M. V. Wilkes, R. M. Needham; The Cambridge Model Distributed System; ACM SIGOPS, June 1980