(19) **Bundesrepublik Deutschland**
**Deutsches Patent- und Markenamt**

(10) **DE 698 34 885 T2** 2007.05.24

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 905 620 B1**
(21) Deutsches Aktenzeichen: **698 34 885.0**
(96) Europäisches Aktenzeichen: **98 307 130.9**
(96) Europäischer Anmeldetag: **04.09.1998**
(97) Erstveröffentlichung durch das EPA: **31.03.1999**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **14.06.2006**
(47) Veröffentlichungstag im Patentblatt: **24.05.2007**

(51) Int Cl.[8]: ***G06F 9/46*** (2006.01)

(30) Unionspriorität:
**25388997 18.09.1997 JP**

(84) Benannte Vertragsstaaten:
**DE, FR, GB**

(73) Patentinhaber:
**Sony Corp., Tokio/Tokyo, JP**

(72) Erfinder:
**Murata, Seiji, Shinagawa-ku, Tokyo 141, JP**

(74) Vertreter:
**Mitscherlich & Partner, Patent- und Rechtsanwälte, 80331 München**

(54) Bezeichnung: **Datenverarbeitungsverfahren und Aufzeichnungsmedium**

## Beschreibung

**[0001]** Die vorliegende Erfindung betrifft eine) Datenverarbeitungsverfahren und -vorrichtung sowie ein Aufzeichnungsmedium. Ein beispielhaftes Ausführungsbeispiel betrifft einen durch ein Betriebssystem realisierten Ablaufverwaltungsmechanismus. Das Ausführungsbeispiel betrifft auch ein ein solches Betriebssystem speicherndes Aufzeichnungsmedium sowie eine ein solches Aufzeichnungsmedium einsetzende Datenverarbeitungsvorrichtung.

**[0002]** Betriebssysteme können in die folgenden drei Arten sortiert werden: Betriebssysteme mit so genannten fetten Kernroutinen, Betriebssysteme mit so genannten Mikro-Kernroutinen und Betriebssysteme mit so genannten Nano-Kernroutinen.

**[0003]** In einem Betriebssystem der fetten Kernroutinenart sind alle Funktionen des Systems in eine fette Kernroutine gepackt. Bezug nehmend auf Fig. 12 wird, wenn ein Anwendungsprogramm das Betriebssystem auffordert, einen Dienst bereitzustellen, der von einer Änderung im Teilprozessstatus begleitet wird, ein Systemaufruf gemacht, um von der Kernroutine ein Modul entsprechend dem angeforderten Dienst, wie beispielsweise „Mailer" oder „Synchronisation" aufzurufen. Anschließend wird, um es einem solchen Modul zu ermöglichen, die Zustände von Teilprozessen zu verändern, ein so genannter Funktionsruf durchgeführt, um einen „Scheduler" aufzurufen, der die Ausführungsabfolge der Teilprozesse steuert, um den Ablauf dieser Teilprozesse zu verwalten, wodurch die eine Veränderung im Teilprozessstatus beinhaltende Verarbeitung durchgeführt wird.

**[0004]** Um verschiedene Funktionen in das Betriebssystem mit einer fetten Kernroutine zu packen, ist ein Programmierer erforderlich, der ausreichend Kenntnis über die Kernroutine hat, weil alle Funktionen in der Kernroutine gepackt sein müssen. Weiter kann eine Änderung von Funktionen nicht einfach bewirkt werden, da diese Funktionen in der Kernroutine gepackt sind. Somit haben die Betriebssysteme mit fetten Kernroutinen im Allgemeinen eine beschränkte Vielseitigkeit.

**[0005]** Bezug nehmend nun auf das Betriebssystem der Mikro-Kernroutinenart nimmt die Mikro-Kernroutine nur Grundfunktionen, wie beispielsweise Hardware-Steuerfunktionen auf, während andere Funktionen des Betriebssystems außerhalb der Mikro-Kernroutine implementiert sind. Diese Art Betriebssystem ist deshalb frei von dem Problem begrenzter Vielseitigkeit, das den Betriebssystemen der fetten Kernroutinenart inhärent ist, weil nur ausgewählte Funktionen in der Kernroutine gepackt sind.

**[0006]** Wie in Fig. 13 dargestellt, erlaubt das Betriebssystem des Mikro-Kernroutinentyps nicht nur das Laufen von Anwendungsprogrammen, sondern auch von Modulen, die Betriebssystemfunktionen, z.B. „Synchronisation", „Mailer" und „Schedule", wie in Fig. 13 dargestellt, vorsehen, auf der Mikro-Kernroutine. Deshalb unterliegen diese Module, die die Betriebssystemfunktionen vorsehen, der Abwicklungssteuerung des in der Mikro-Kernroutine geladenen Abwicklers.

**[0007]** In dem Betriebssystem des Mikro-Kernroutinentyps werden Wechselwirkungen zwischen Anwendungsprogrammen und Modulen mittels eines Kommunikationsmechanismus „Mailer" ausgeführt, der durch die Mikro-Kernroutine bereitgestellt wird. Gleichzeitig steuern die Module, welche die Betriebssystemfunktionen bereitstellen, die Ausführung von Anwendungsprogrammen durch direktes Aufrufen des „Scheduler" in der Mikro-Kernroutine über eine Schnittstelle, die ebenfalls durch die Mikro-Kernroutine bereitgestellt wird.

**[0008]** Bezug nehmend nun auf Fig. 14 wird in einem Betriebssystem des Nano-Kernroutinentyps nur ein Aufgabenumschaltmechanismus, der das Umschalten der Aufgaben unternimmt, aus den Inhalten einer Mikro-Kernroutine extrahiert. Der extrahierte Aufgabenumschaltmechanismus bildet eine Nano-Kernroutine.

**[0009]** Somit wird in dem Betriebssystem des Nano-Kernroutinensystems der Aufgabenumschaltmechanismus, der eine Abhängigkeit von der Art der CPU und dergleichen hat, extrahiert und abstrahiert. Dies bietet eine größere Einfachheit der Programmierung sowie eine höhere Kernroutinenmobilität.

**[0010]** Um eine verbesserte Vielseitigkeit von Betriebssystemen sowie eine höhere Kernroutinenmobilität und eine größere Einfachheit der Programmierung zu erzielen, wird ein Betriebssystem des Mikro-Kernroutinentyps gegenüber einem Betriebssystem des fetten Kernroutinentyps bevorzugt, und ein Betriebssystem des Nano-Kernroutinentyps wird gegenüber einem Betriebsystem des Mikro-Kernroutinentyps bevorzugt.

**[0011]** Ein Betriebssystem des Mikro- oder des Nano-Kernroutinentyps erfordert jedoch eine große Anzahl

von Aufgabenumschaltvorgängen. Dies deshalb, weil der Kommunikationsmechanismus und der Abwickler der Mikro-Kernroutine bei jeder Ausführung irgendeiner Ausführungsentität wie beispielsweise den Anwendungsprogrammen, die auf dem Betriebssystem betrieben werden, Objekten, welche das Betriebssystem vorsehen, und dergleichen aufgerufen werden müssen. Folglich ist die Gesamtleistung des Betriebssystems des Mikro- oder des Nano-Kernroutinentyps schlechter als jene des Betriebssystems des fetten Kernroutinentyps.

**[0012]** Das ein Nano-Kernroutinensystem verwendende Betriebssystem erfordert ein Aufgabenumschalten selbst für den Zweck des Rufens des Kommunikationsmechanismus oder des Abwicklers. Diese Art Betriebssystem vergrößert deshalb weiter den Kosten-Overhead aufgrund der Verwendung einer größeren Anzahl von Aufgabenumschaltvorgängen gegenüber den beim Betriebssystem des Mikro-Kernroutinentyps angefallenen Kosten.

**[0013]** Somit beeinträchtigt die Verwendung einer Nano-Kernroutine in unerwünschter Weise die Leistung des gesamten Systems aufgrund einer Vergrößerung der Anzahl der Aufgabenumschaltvorgänge. Eine Lösung für dieses Problem kann darin bestehen, das Aufrufen des Abwicklers zu unterdrücken, wenn ein Aufgabenumschalten durchgeführt wird. Ein bloßes Unterdrücken des Aufrufens des Abwicklers erschwert jedoch die Abwicklungsverwaltung und damit den Systembetrieb.

**[0014]** Insbesondere bewirkt die Unterlassung eines Abwickleraufrufs, dass der Abwickler zur Abwicklungsverwaltung notwendige Informationen nicht empfängt, was in einem Widerspruch zwischen dem tatsächlichen Systemstatus und dem durch den Abwickler verstandenen Status resultiert. Folglich ist der Abwickler nicht in der Lage, eine geeignete Abwicklungsverwaltung durchzuführen. Das Unterlassen eines Abwickleraufrufs ergibt auch das folgende Problem. Im Allgemeinen führt der Abwickler, wenn die Warteschlage in einem Teilprozess unter der Ablaufsteuerung durch einen Abwickler geändert wird, eine Neuabwicklung derart durch, dass Verarbeitungen höherer Prioritäten bevorzugt ausgeführt werden. Eine solche Neuabwicklung wird nicht durchgeführt, wenn auf den Aufruf des Abwicklers verzichtet wird. Verarbeitungen höherer Prioritäten können deshalb unerwünscht verzögert werden.

**[0015]** Demgemäß will ein Ausführungsbeispiel der vorliegenden Erfindung ein Datenverarbeitungsverfahren mit einer Nano-Kernroutine vorsehen, bei welcher die Anzahl von Anrufen für einen Abwickler ohne Verschlechtern des Betriebs eines Systems, das dieses Verfahren einsetzt, minimiert ist. Ein Ausführungsbeispiel der vorliegenden Erfindung will auch ein Aufzeichnungsmedium mit einem Programm vorsehen, das das Datenverarbeitungsverfahren implementiert, sowie eine Datenverarbeitungsvorrichtung mit einem solchen Aufzeichnungsmedium.

**[0016]** Diesbezüglich ist gemäß einem Aspekt der vorliegenden Erfindung ein Datenverarbeitungsverfahren vorgesehen, bei welchem, wenn ein Aufgabenumschalten stattgefunden hat, um den auszuführenden Teilprozess umzuschalten, die Historie des Aufgabenumschaltens aufgezeichnet wird. Dann wird, wenn ein Abwickler zum Steuern der Ausführungsabfolge der Teilprozesse aufgerufen wird, eine durch den Abwickler verwaltete Warteschlage durch Zurückverfolgen der aufgezeichneten Historie des Aufgabenumschaltens aktualisiert.

**[0017]** So wird in dem Datenverarbeitungsverfahren der vorliegenden Erfindung eine Warteschlage aktualisiert, indem die Aufzeichnung einer Aufgabenumschalthistorie zurückverfolgt wird, wenn ein Abwickler aufgerufen wird. Der Abwickler kann deshalb den Ablauf korrekt verwalten, selbst wenn irgendein Aufgabenumschalten ohne Aufrufen des Abwicklers durchgeführt worden ist.

**[0018]** Das Aktualisieren kann durch Setzen eines vorbestimmten Merkers an einem Teilprozess, der ein Aufgabenumschalten bewirkt, das eine Veränderung der Reihenfolge in der Warteschlange erfordert, und Aufrufen des Abwicklers als Reaktion auf ein Auftreten eines neuen Aufgabenumschaltens, um die Reihenfolge in der Warteschlage zu verändern, wenn der Teilprozess, der das Aufgabenumschalten bewirkt hat, diesen Merker trägt, durchgeführt werden.

**[0019]** Das Setzen des Merkers kann so ausgeführt werden, dass der Teilprozess, an dem der Merker gesetzt ist, sich gemäß der Art des Aufrufs des Abwicklers unterscheidet. Dies gewährleistet, dass eine Neuabwicklung in der Warteschlage zu Zeiten ausgeführt wird, die geeigneter sind.

**[0020]** Gemäß einem weiteren Aspekt der vorliegenden Erfindung ist ein computerlesbares Aufzeichnungsmedium mit einem Betriebssystem vorgesehen, das einen Abwickler besitzt, der die Ausführungsreihenfolge von Teilprozessen steuert. Das Betriebssystem implementiert ein Datenverarbeitungsverfahren, bei dem, wenn ein Aufgabenumschalten stattgefunden hat, um den auszuführenden Teilprozess umzuschalten, die His-

torie des Aufgabenumschaltens aufgezeichnet wird. Dann wird, wenn ein Abwickler zum Steuern der Ausführungsreihenfolge von Teilprozessen aufgerufen wird, eine durch den Abwickler verwaltete Warteschlange aktualisiert, indem die aufgezeichnete Historie des Aufgabenumschaltens zurückverfolgt wird.

**[0021]** So wird in dem durch das im Aufzeichnungsmedium der Erfindung enthaltene Betriebssystem implementierte Verfahren eine Warteschlange aktualisiert, indem die Aufzeichnung der Aufgabenumschalthistorie zurückverfolgt wird, wenn ein Abwickler aufgerufen wird. Der Abwickler kann deshalb den Ablauf korrekt verwalten, selbst wenn irgendein Aufgabenumschalten ohne Aufrufen des Abwicklers durchgeführt worden ist.

**[0022]** Das Aktualisieren kann durch Setzen eines vorbestimmten Merkers an einem Teilprozess, der ein Aufgabenumschalten bewirkt, das eine Veränderung der Reihenfolge in der Warteschlange bewirkt, und Aufrufen des Abwicklers als Reaktion auf ein Auftreten eines neuen Aufgabenumschaltens, um die Reihenfolge in der Warteschlage zu ändern, wenn der Teilprozess, der das neue Aufgabenumschalten bewirkt hat, den Merker trägt, durchgeführt werden. Dies gewährleistet, dass die Reihenfolge in der Warteschlage zu geeigneten Zeiten neu abgewickelt wird, wodurch eine Unannehmlichkeit wie beispielsweise ein Verzögern einer Verarbeitung mit einer hohen Priorität beseitigt wird.

**[0023]** Das Setzen des Merkers kann so ausgeführt werden, dass der Teilprozess, an dem der Merker gesetzt wird, sich entsprechend der Art des Aufrufs des Abwicklers unterscheidet. Dies gewährleistet, dass eine Neuabwicklung in der Warteschlange zu Zeiten ausgeführt wird, die geeigneter sind.

**[0024]** Gemäß einem noch weiteren Aspekt der vorliegenden Erfindung ist eine Datenverarbeitungsvorrichtung vorgesehen, mit einer Einrichtung zum Aufzeichnen der Historie des Aufgabenumschaltens, wenn ein Aufgabenumschalten stattgefunden hat, um den auszuführenden Teilprozess umzuschalten; und einer Einrichtung zum Aktualisieren einer durch den Abwickler verwalteten Warteschlange, wenn ein Abwickler zum Steuern der Ausführungsreihenfolge von Teilprozessen aufgerufen wird, durch Rückverfolgen der aufgezeichneten Historie des Aufgabenumschaltens.

**[0025]** So wird in der Datenverarbeitungsvorrichtung der vorliegenden Erfindung eine Warteschlange aktualisiert, indem die Aufzeichnung der Aufgabenschalthistorie zurückverfolgt wird, wenn ein Abwickler aufgerufen wird. Der Abwickler kann deshalb den Ablauf korrekt verwalten, selbst wenn irgendein Aufgabenumschalten ohne Aufrufen des Abwicklers durchgeführt worden ist.

**[0026]** Das Aktualisieren kann durchgeführt werden durch Setzen eines vorbestimmten Merkers an einem Teilprozess, der ein Aufgabenumschalten bewirkt, das eine Veränderung der Reihenfolge in der Warteschlange erfordert, und Aufrufen des Abwicklers als Reaktion auf ein Auftreten eines neuen Aufgabenumschaltens, um die Reihenfolge in der Warteschlange zu ändern, wenn der Teilprozess, der das neue Aufgabenumschalten bewirkt hat, den Merker trägt. Dies gewährleistet, dass die Reihenfolge in der Warteschlange zu geeigneten Zeiten neu abgewickelt wird, wodurch eine Unannehmlichkeit wie beispielsweise ein Verzögern einer Verarbeitung mit einer hohen Priorität beseitigt wird.

**[0027]** Das Setzen des Merkers kann so ausgeführt werden, dass der Teilprozess, an dem der Merker gesetzt ist, sich entsprechend der Art des Aufrufs des Abwicklers unterscheidet. Dies gewährleistet, dass eine Neuabwicklung in der Warteschlange zu Zeiten ausgeführt wird, die geeigneter sind.

**[0028]** Ein besseres Verständnis der vorliegenden Erfindung wird aus der folgenden beispielhaften Beschreibung klar, wenn sie zusammen mit den beiliegenden Zeichnungen gelesen wird. Darin zeigen:

**[0029]** Fig. 1 eine schematische Darstellung eines Aufbaus eines TV-Systems mit einem Ausführungsbeispiel der vorliegenden Erfindung;

**[0030]** Fig. 2 eine Darstellung einer hierarchischen Struktur eines die vorliegende Erfindung verwendenden beispielhaften Betriebssystems;

**[0031]** Fig. 3 eine Darstellung eines herkömmlichen Ablaufverwaltungsmechanismus, die insbesondere einen Prozess zum Senden einer Mitteilung von einem Objekt A zu einem anderen Objekt B sowie eine hierarchische Struktur eines Betriebssystems zeigt;

**[0032]** Fig. 4 eine Darstellung des herkömmlichen Ablaufverwaltungsmechanismus, die mittels eines Zeitdiagramms eine Objektwechselwirkung zwischen dem Objekt A und dem Objekt B, um das Senden der Mittei-

lung vom Objekt A zum Objekt B zu ermöglichen, zeigt;

**[0033]** Fig. 5 eine Darstellung eines die vorliegende Erfindung einsetzenden Ablaufverwaltungsmechanismus, die insbesondere einen Prozess zum Senden einer Mitteilung von einem Objekt A zu einem anderen Objekt B sowie eine hierarchische Struktur eines Betriebssystems zeigt;

**[0034]** Fig. 6 eine Darstellung des Ablaufverwaltungsmechanismus eines Ausführungsbeispiels der Erfindung, die mittels eine Zeitdiagramms eine Objektwechselwirkung zwischen dem Objekt A und dem Objekt B, um das Senden einer Mitteilung vom Objekt A zum Objekt B zu ermöglichen, zeigt;

**[0035]** Fig. 7 eine Darstellung eines Übergangs eines Teilprozessstatus;

**[0036]** Fig. 8 ein Flussdiagramm eines Algorithmus eines verzögerten Warteschlangenmechanismus;

**[0037]** Fig. 9 ein Flussdiagramm eines Algorithmus zum Setzen von „currentThread" in dem verzögerten Warteschlangenmechanismus;

**[0038]** Fig. 10 eine Darstellung eines Prozesses zum Senden einer Mitteilung von einem Objekt A zu einem weiteren Objekt C mittels eines asynchronen Abwicklerrufmechanismus, die auch die hierarchische Struktur eines Betriebssystems zeigt;

**[0039]** Fig. 11 ein Zeitdiagramm einer Objektwechselwirkung zwischen dem Objekt A und dem Objekt C zum Senden einer Mitteilung vom Objekt A zum Objekt C mittels des asynchronen Abwicklerrufmechanismus;

**[0040]** Fig. 12 eine Darstellung eines Betriebssystems, das eine fette Kernroutine einsetzt;

**[0041]** Fig. 13 eine Darstellung eines Betriebssystems, das eine Mikro-Kernroutine einsetzt; und

**[0042]** Fig. 14 eine Darstellung eines Betriebssystems, das eine Nano-Kernroutine einsetzt.

## 1. Hardwareumgebung

**[0043]** Fig. 1 zeigt beispielhaft eine die vorliegende Erfindung integrierende Hardwarearchitektur. Obwohl in der folgenden Beschreibung speziell ein TV-System genannt wird, ist es selbstverständlich, dass die vorliegende Erfindung auch auf eine Vielzahl von Arten von Datenverarbeitungsvorrichtungen angewendet werden kann, welche Betriebssysteme einsetzen. Zum Beispiel kann die vorliegende Erfindung effektiv in audiovisuellen Systemen, d.h. so genannten AV-Systemen, Bürogeräten, Computern und dergleichen sowie in den Fernsehsystemen, die nun beschrieben werden, verwendet werden.

**[0044]** Bezug nehmend auf Fig. 1 empfängt das TV-System als ein Ausführungsbeispiel der Datenverarbeitungsvorrichtung der Erfindung Signale von einer Station über die Luft (Antenne) oder durch ein Kabel. Das TV-System zeigt ein Bild basierend auf dem empfangenen Signal auf einer Kathodenstrahlröhre oder einem Flüssigkristallanzeigegerät an, während es akustische Signale von einem Lautsprecher ausgibt.

**[0045]** Dieses TV-System hat neben gewöhnlichen TV-Funktionen eine Funktion zum Empfangen von Programmen und Daten die ihm von außen gegeben werden. Insbesondere hat das TV-System, wie es in Fig. 1 dargestellt ist, eine mit einem Bus **2** durch eine Bus/EA-Brücke **1** verbundene TV-Funktionseinheit, einen mit dem Bus **2** durch eine Bus/Speicher-Brücke **4** verbundenen Prozessor **5**, einen ROM (Festwertspeicher) **6** und einen RAM (Direktzugriffsspeicher) **7**, die durch die Bus/Speicher-Brücke **4** ebenfalls mit dem Prozessor **5** verbunden sind, ein mit dem Bus **2** verbundenes Bedienfeld **8** und ein externes Speichergerät **9** sowie ein Kommunikationsgerät **10**, die ebenfalls mit dem Bus **2** verbunden sind.

**[0046]** Die TV-Funktionseinheit **3** besitzt eine Funktion zum Erzeugen von Bildern und Stimmen basierend auf über die Luft oder ein Kabel empfangenen Signalen und ist mit dem Bus **2** durch die Bus/EA-Brücke **1** zur Kommunikation mit anderen Geräten des Systems verbunden.

**[0047]** Der Prozessor **5** steuert Funktionen verschiedener Geräte, die das TV-System bilden. Diesbezüglich ist der Prozessor **5** über die Bus/Speicher-Brücke **4** mit dem Bus **2** verbunden. Mit dem Prozessor **5** sind über die Bus/Speicher-Brücke **4** auch der ROM **6** und der RAM **7** verbunden.

**[0048]** Der ROM **6** speichert ein Betriebssystem, Anwendungsprogramme und dergleichen, die die durch den Prozessor **5** durchzuführenden Steuerungen implementieren. In diesem Ausführungsbeispiel setzt das Betriebssystem eine Nano-Kernroutine ein und ist objektorientiert. Somit ist das Betriebssystem ein objektorientiertes Betriebssystem mit Nano-Kernroutine.

**[0049]** Der RAM **7** wird als ein Arbeitsbereich für den Prozessor **5** benutzt. Somit benutzt der Prozessor **5** den RAM **7** beim Ausführen des Betriebssystems und der Anwendungsprogramme als Arbeitsbereich, wodurch die Funktionen verschiedener, das TV-System bildender Geräte gesteuert werden.

**[0050]** Das Bedienfeld **8** dient als ein Eingabegerät, das durch den Benutzer eingegebene Betriebsbefehle empfängt, wie beispielsweise Befehle zum Kanalwechsel oder zur Lautstärkeregelung. In der Praxis hat das Bedienfeld **8** ein Eingabegerät wie beispielsweise eine Konsole mit mehreren Knöpfen, ein Zeigegerät wie beispielsweise eine Maus, und dergleichen. Durch das Bedienfeld **8** eingegebene Signale werden über den Bus **2** und die Bus/Speicher-Brücke **4** an den Prozessor **5** geliefert. Basierend auf den durch das Bedienfeld **8** eingegebenen Signalen führt der Prozessor **5** verschiedene Berechnungen durch, um die Komponenten zu steuern.

**[0051]** Das externe Speichergerät **9**, das ein Festplattengerät sein kann, speichert Bilddaten, Steuerdaten und Anwendungsprogramme, die von einem externen System über das Kommunikationsgerät **10** heruntergeladen wurden. Somit dient das Kommunikationsgerät **10** als eine E/A-Einheit, die einen Signalaustausch mit dem externen System unternimmt und zum Beispiel ein Modem oder einen Terminaladapter aufweist.

**[0052]** Das TV-System führt nicht nur gewöhnliche TV-Funktionen durch, die durch die TV-Funktionseinheit **3** bereitgestellt werden, sondern hat auch verschiedene andere Funktionen. Zum Beispiel kann das TV-System Programme und Daten von externen Systemen über das Kommunikationsgerät **10** empfangen. Diese Funktion ermöglicht ein Aktualisieren von Versionen des Betriebssystems und der Gerätetreiber einfach durch Empfangen neuer Softwaremodule von einem externen Netzwerk über das Kommunikationsgerät **10**.

**[0053]** Dieses TV-System führt unter der Steuerung des Prozessors **5** das in dem ROM gespeicherte Betriebssystem aus und lässt die in dem ROM **6** oder im externen Speichergerät **9** gespeicherten Anwendungsprogramme auf dem Betriebssystem laufen, wodurch die Komponentengeräte gesteuert werden. Somit dient der ROM **6** als ein computerlesbares Speichermedium, welches das Betriebssystem und die Gerätetreiber speichert. Das Betriebssystem kann jedoch auch in dem RAM **7** oder im externen Speichergerät **9** gespeichert werden. Die Speicherung im RAM **7** oder im externen Speichergerät **9** ist bevorzugt, wenn ein Überschreiben des Betriebssystems und der Gerätetreiber notwendig ist.

## 2. Softwareumgebung

**[0054]** Es wird nun eine Beschreibung der Softwareumgebung des TV-Systems gegeben.

### 2-1 Darstellung der Betriebssystemarchitektur

**[0055]** Das in diesem TV-System benutzte Betriebssystem ist ein Betriebssystem des Nano-Kernroutinentyps, das eine Nano-Kernroutine verwendet. Wie in Fig. 2 dargestellt, ist ein Metakern integriert, um als Nano-Kernroutine zu dienen, die das Aufgabenumschalten unternimmt.

**[0056]** Dieses Betriebssystem des Nano-Kernroutinentyps ist objektorientiert. Daher sind die Ausführungsentitäten, die auf dem als die Nano-Kernroutine dienenden Metakern laufen, Objekte. Die Objektwechselwirkung wird durch Mitteilungen erzielt.

**[0057]** Dieses Betriebssystem ist in der Lage, gleichzeitig mehrere Programmausführungsumgebungen vorzusehen. In der folgenden Beschreibung werden die durch das Betriebssystem vorgesehenen Programmausführungsumgebungen als Metaräume bezeichnet. Dieses Betriebssystem sieht drei Arten von Metaräumen vor, wie in Fig. 2 dargestellt, nämlich einen Metaraum mCOOP, einen Metaraum mDrive und einen Metaraum mCore. Jeder dieser Metaräume ist aus einer Vielzahl von Objekten aufgebaut. In der folgenden Beschreibung wird jeder Metaraum als ein Bauelement eines Metaraums als ein „Metaobjekt" bezeichnet.

**[0058]** Der Metaraum mCOOP wird benutzt, um ein Laufen eines objektorientierten Anwendungsprogramms, wie beispielsweise eines Programms zum Implementieren einer grafischen Nutzerschnittstelle, die eine Steuerung der Bedienfeldes 8 ermöglicht, zu ermöglichen. Der Anfangsbuchstabe „m" des Ausdrucks mCOOP ist

ein Zeichen, das einen Metaraum angibt, während „COOP" eine Abkürzung von „Concurrent Object Oriented Programming" ist.

**[0059]** Der Metaraum mDrive wird zum Laufen von Gerätetreibern, die Steuerungen von Hardwaregeräten unternehmen, verwendet. Der Anfangsbuchstabe „m" des Ausdrucks mDrive ist ein Zeichen, das einen Metaraum angibt, während „Drive" zeigt, dass dieser Metaraum zum Laufen von Gerätetreibern benutzt wird.

**[0060]** Der Metaraum mCore wird zum Laufen von Metaobjekten benutzt, welche die Metaräume mCOOP und mDrive bilden. Daher entspricht der Metaraum mCore einer Mikro-Kernroutine. Der Anfangsbuchstabe „m" des Ausdrucks mCore ist ein Zeichen, das einen Metaraum angibt, während „Core" angibt, dass dieser Metaraum einen „Kern" des Betriebssystems bildet.

**[0061]** Somit ist die hierarchische Struktur des Betriebssystems derart, dass, wie in Fig. 2 dargestellt, der Metaraum mCore, auf dem die Metaräume mCOOP und mDrive arbeiten, auf einem Hardwaregerät wie beispielsweise einer CPU läuft und die objektorientierten Anwendungsprogramme und die Gerätetreiber auf dem Metaraum mCOOP bzw. dem Metaraum mDrive laufen. Aufgabenumschaltvorgänge zwischen diesen Aufgaben wird durch den Metakern gesteuert.

**[0062]** Das Betriebssystem kann neben den Metaräumen mCOOP und mDrive auch andere Metaräume bereitstellen, die auf dem Metaraum mCore laufen können, wie beispielsweise ein Prozeduranwendungsprogramm, z.B. ein Anwendungsprogramm, das es der TV-Funktionseinheit ermöglicht, bewegte Bilder anzuzeigen. Auf eine Erläuterung anderer Metaräume als die Metaräume mCOOP und mDrive wird in der folgenden Beschreibung jedoch verzichtet, weil die Erfindung gleichermaßen mit verschiedenen Arten von Metaräumen ausgeführt werden kann, die auf dem Metaraum mCore arbeiten können. Daher werden in der folgenden Beschreibung speziell die Metaräume mCOOP und mDrive beispielhaft genannt.

## 2-2 Metaobjekte

**[0063]** Wie in Fig. 2 dargestellt, gibt es drei Arten von Metaobjekten, die den Metaraum mCOOP bilden: nämlich ein Objekt „mCOOP-Scheduler", ein Objekt „mCOOP-Mailer" und ein Objekt „mCOOP-Fehlerbehandler". Das Objekt mCOOP-Scheduler ist ein so genannter „Scheduler" bzw. Abwickler, der ein Metaobjekt ist, das eine Ablaufverwaltung für Anwendungsprogramme unternimmt, die auf dem Metaraum mCOOP laufen. Das Objekt mCOOP-Mailer ist ein Metaobjekt, das Mitteilungen zwischen den Anwendungsprogrammen, die auf dem Metaraum mCOOP laufen, ausführt. Das Objekt mCOOP-Fehlerbehandler ist ein Metaobjekt zum Ausführen einer Fehlerbehandlung. Der Metaraum mCOOP hat zusätzlich zu den oben speziell genannten Metaobjekten verschiedene Metaobjekte als Bestandteile.

**[0064]** Weiter Bezug nehmend auf Fig. 2 gibt es drei Arten von Metaobjekten, die den Metaraum mDrive bilden: nämlich ein Objekt „mDrive-Scheduler", ein Objekt „mDrive-Mailer" und ein Objekt „mDrive-Fehlerbehandler". Das Objekt mDrive-Scheduler ist ein so genannter „Scheduler" (bzw. Abwickler), der ein Metaobjekt ist, das eine Ablaufverwaltung für Gerätetreiber, die auf dem Metaraum mDrive laufen, unternimmt. Das Objekt mDrive-Mailer ist ein Metaobjekt, das Mitteilungen zwischen den Gerätetreibern, die auf dem Metaraum mDrive laufen, ausführt. Das Objekt mDrive-Fehlerbehandler ist ein Metaobjekt zur Ablaufunterbrechungsbehandlung. Der Metaraum mDrive hat zusätzlich zu den oben speziell genannten Metaobjekten verschiedene Metaobjekte als Bestandteile.

**[0065]** Bezug nehmend weiter auf Fig. 2 gibt es zwei Arten von Metaobjekten, die den Metaraum mCore bilden: nämlich ein Objekt „mCore-Scheduler" und ein Objekt „mCore-Mailer". Das Objekt mCore-Scheduler ist ein so genannter „Scheduler" (bzw. Abwickler), der ein Metaobjekt ist, das eine Ablaufverwaltung für Objekte unternimmt, die auf dem Metaraum mCore laufen. Das Objekt mCore-Mailer ist ein Metaobjekt, das Mitteilungen zwischen den Objekten, die auf dem Metaraum mCore laufen, ausführt. Der Metaraum mCore hat ebenfalls zusätzlich zu den zwei Arten von oben speziell genannten Metaobjekten verschiedene Metaobjekte als Bestandteile.

**[0066]** Obwohl in dem beschriebenen Ausführungsbeispiel jeder Metaraum sein eigenes Metaobjekt als einen Scheduler verwendet, ist dies nicht ausschließlich der Fall, und die Anordnung kann auch derart sein, dass ein Abwicklungs-Metaobjekt von allen Metaräumen gemeinsam benutzt wird. Bei einer solchen Anordnung dient zum Beispiel ein einzelnes Abwicklungs-Metaobjekt als ein Metaobjekt, das zu dem Metaraum mCOOP gehört, wenn es in diesem Metaraum benutzt wird.

**[0067]** Analog wirkt das gleiche Abwicklungs-Metaobjekt als ein Objekt, das zum Metaraum mDrive gehört, wenn es in diesem Metaraum benutzt wird, und als ein Objekt, das zum Metaraum mCore gehört, wenn es in diesem Metaraum benutzt wird.

**[0068]** Objekte, die auf einem Metaraum laufen, werden in der folgenden Beschreibung als „Basisobjekte" bezeichnet. Ein Aufrufen eines Metaobjekts, ausgeführt, um ein Basisobjekt in den Genuss von durch das Metaobjekt angebotenen Diensten kommen zu lassen, wird als ein „Metaruf" bezeichnet. Somit ist der Metaruf äquivalent zu einem Systemaufruf für ein Betriebssystem eines fetten Kernroutinentyps. Ein Aufgabenumschalten zwischen einem Basisobjekt und einem Metaobjekt wird durch den Metakern ausgeführt, der der Nano-Kernroutine entspricht.

**[0069]** In der folgenden Beschreibung wird der Begriff „Scheduler" (bzw. Abwickler) benutzt, um die Abwicklungs-Metaobjekte einschließlich dem Objekt mCOOP-Scheduler, dem Objekt mDrive-Scheduler und dem Objekt mCore-Scheduler gemeinsam zu bezeichnen.

## 2-3 Herkömmlicher Ablaufverwaltungsmechanismus

**[0070]** Vor der Beschreibung der Ausführungsbeispiele wird eine Beschreibung einer herkömmlichen Ablaufverwaltung gegeben, die in einem Betriebssystem des Nano-Kernroutinentyps angewendet wird. Die Beschreibung erfolgt unter speziellem Bezug auf **Fig. 3** und **Fig. 4**, die einen Vorgang zum Senden einer Mitteilung von einem Objekt A, welches ein Anwendungsprogramm ist, das auf dem Metaraum mCOOP läuft, zu einem anderen Objekt B, das ebenfalls ein Anwendungsprogramm ist, das auf dem gleichen Metaraum mCOOP läuft, zeigen. **Fig. 3** zeigt zusammen mit einer hierarchischen Struktur des Betriebssystems einen Prozess zum Senden einer Mitteilung vom Objekt A zum Objekt B. Auf eine Darstellung des Metaraums mDrive wird in **Fig. 3** verzichtet. **Fig. 4** ist ein Zeitdiagramm der Objektwechselwirkung zwischen den Objekten A und B, die ausgeführt wird, um ein Senden einer Mitteilung vom Objekt A zum Objekt B zu ermöglichen.

**[0071]** Die Prozedur ist wie folgt.

### (1) Metaruf an mCOOP-Mailer

**[0072]** Als erster Schritt wird ein Metaruf durch das Objekt A gemacht, um das Objekt „mCOOP-Mailer" auf dem Metaraum mCOOP aufzurufen. So wird, um eine Mitteilung vom Objekt A zum Objekt B zu ermöglichen, das Objekt „mCOOP-Mailer", das eine Mitteilung zwischen Objekten im Metaraum mCOOP unternimmt, aufgerufen.

**[0073]** Insbesondere wird zum Zweck des Aufrufens des Objekts „mCOOP-Mailer", das den Metaruf vom Objekt A behandelt, ein Aufruf gemacht, wie durch den Pfeil A1 angedeutet, um das Objekt „mCore-Scheduler" des Metaraums mCore, auf dem das bestimmte Objekt „ mCOOP-Mailer" läuft, aufzurufen. Somit sind die Objekte im Metaraum mCore unter der Ablaufsteuerung des Objekts „mCore-Scheduler". In diesem Zustand wird das bestimmte Objekt „mCOOP-Mailer" im Metaraum mCOOP aufgerufen, wie durch den Pfeil A2 angedeutet.

### (2) Senden an mCOOP-Scheduler

**[0074]** Dann sendet das Objekt „mCOOP-Mailer", um das Objekt B zu starten, eine Mitteilung an das Objekt „mCOOP-Scheduler" im Metaraum mCOOP. Insbesondere erfordert das Starten des Objekts B eine Ablaufverwaltung der im Metaraum mCOOP laufenden Objekte. Die oben genannte Mitteilung, die notwendig ist, um das Objekt B zu starten, wird deshalb zum Objekt „mCOOP-Scheduler" gesendet, das eine Ablaufverwaltung der Objekte im Metaraum mCOOP ausführt.

**[0075]** Genauer beginnt das Senden der Mitteilung vom Objekt „mCOOP-Mailer" zum Objekt „mCOOP-Scheduler" mit einem durch den Pfeil A3 angegebenen Ruf zum Aufrufen des Objekts „mCore-Mailer", welches das Metaobjekt mit der Aufgabe des Mitteilens zwischen im Metaraum mCore arbeitenden Objekten ist. Anschließend wird, wie durch den Pfeil A4 angedeutet, das Objekt „mCore-Scheduler", das ein Metaobjekt mit der Aufgabe der Ablaufverwaltung der Objekte im Metaraum mCore ist, gerufen, da das Mitteilungszielobjekt „mCOOP-Scheduler" auf dem Metaraum mCore läuft. Somit unterliegen die Objekte im Metaraum mCore der durch das Objekt „mCore-Scheduler" durchgeführten Ablaufsteuerung. Unter dieser Ablaufsteuerung wird das Objekt „mCOOP-Scheduler" ausgelöst, wie durch den Pfeil A5 angedeutet, und die obige Mitteilung, die zum Starten des Objekts B notwendig ist, wird zu diesem Objekt „mCOOP-Scheduler" geschickt.

(3) Antworten an mCOOP-Mailer

**[0076]** Bei Empfang der Mitteilung ändert das Objekt „mCOOP-Scheduler" den Zustand des Objekts B entsprechend der empfangenen Mitteilung und sendet eine Antwort auf diese Mitteilung an das Objekt „mCOOP-Mailer".

**[0077]** Genauer wird dieser Antwortvorgang wie folgt durchgeführt. Um das Senden der Antwort vom Objekt „mCOOP-Scheduler" zum Objekt „mCOOP-Mailer" zu ermöglichen, wird ein Ruf gemacht, wie durch den Pfeil A6 angedeutet, um das Objekt „mCore-Mailer" aufzurufen, das das Mitteilen zwischen den im Metaraum mCore lautenden Objekten unternimmt. Dann wird, um das Antwortzielobjekt „mCOOP-Mailer" auf dem Metaraum mCore arbeiten zu lassen, ein Ruf gemacht, wie durch den Pfeil A7 angedeutet, um das Objekt „mCore-Scheduler" aufzurufen, welches das Metaobjekt mit der Aufgabe der Ablaufverwaltung für die Objekte im Metaraum mCore ist. Unter der Ablaufverwaltung für die Objekte im Metaraum mCore, die durch das Objekt „mCore-Scheduler" durchgeführt wird, wird das Objekt „ mCOOP-Mailer" ausgelöst, wie durch den Pfeil A8 angedeutet, und die Antwort vom Objekt „mCOOP-Scheduler" wird zum ausgelösten Objekt „mCOOP-Mailer" geschickt.

(4) Wiederaufnehmen der Basisobjekte

**[0078]** Die Verarbeitung der Objekte A und B wird begonnen, wenn die durch das Objekt „mCOOP-Mailer" durchgeführte Verarbeitung abgeschlossen ist.

**[0079]** Diese Behandlung beginnt mit einem durch den Pfeil A9 angedeuteten Ruf zum Aufrufen des Objekts „mCore-Scheduler". Nachdem eine Verarbeitung, wie beispielsweise ein Umgruppieren, durch das Objekt „mCore-Scheduler" durchgeführt ist, wird das Objekt B als ein Basisobjekt gerufen, wie durch den Pfeil A10 angedeutet, wodurch die Verarbeitung gestartet wird. Es ist jedoch zu beachten, dass die Verarbeitung zuerst mit dem Objekt A startet, d.h. der Pfeil A10 das Objekt A ruft, falls der Verarbeitung des Objekts A gegenüber der Verarbeitung des Objekts B eine Priorität gegeben worden ist.

**[0080]** Die Prozedur für das Mitteilen vom Objekt A zum Objekt B unter der Steuerung eines herkömmlichen Ablaufverwaltungsmechanismus wurde beschrieben. Es ist zu erkennen, dass der herkömmliche Ablaufverwaltungsmechanismus eine übermäßig große Anzahl von auszuführenden Schaltungen von Objekten, d.h. zu viele Aufgabenumschaltvorgänge erfordert. In dem dargestellten System werden diese Aufgabenumschaltvorgänge durch den Metakern gesteuert und ausgeführt, der als eine Nano-Kernroutine dient. Eine solche übermäßig große Anzahl von Aufgabenumschaltvorgängen verschlechtert unerwünscht die Gesamtleistung des Systems. Daher war es schwierig, die Gesamtleistung des gesamten Systems unter Verwendung des herkömmlichen Ablaufverwaltungsmechanismus, der eine unpraktisch große Anzahl von Aufgabenumschaltvorgängen erfordert, zu verbessern.

2-4 Die Erfindung verkörpernder Ablaufverwaltungsmechanismus

**[0081]** Es wird nun eine Beschreibung eines die vorliegende Erfindung einsetzenden Ablaufverwaltungsmechanismus gegeben.

**[0082]** Im Allgemeinen haben Objektwechselwirkungen eine gegenseitige Abhängigkeit. Somit gibt es Fälle, in denen die Übergangsabfolge von Ausführungen zwischen Objekten unabhängig davon, ob ein Gruppieren ausgeführt wird oder nicht, nicht verändert wird. In solchen Fällen wird gemäß den Ausführungsbeispielen der vorliegenden Erfindung ein Gruppierungsvorgang durch absichtliches Sperren des Rufs des Abwicklers weggelassen. So wird gemäß dem vorliegenden Ausführungsbeispiel die Frequenz des Abwickleraufrufs aufgrund einer effektiven Nutzung der Serialität des Objektausführungsübergangs reduziert. Dies dient dem entsprechenden Reduzieren der Anzahl der Aufgabenumschaltvorgänge, was zur Verbesserung der Gesamtleistung des Systems beiträgt.

2-4-1 Ablaufverwaltungsprozess

**[0083]** Ein Ablaufverwaltungsprozess gemäß einem Ausführungsbeispiel der vorliegenden Erfindung wird nun durch Veranschaulichung eines Ausführungsbeispiels unter spezieller Bezugnahme auf **Fig. 5** und **Fig. 6** beschrieben. Um den Vergleich zwischen dem Ablaufverwaltungsprozess des Ausführungsbeispiels zu vereinfachen, wird wie in der Beschreibung des herkömmlichen Prozesses eine Mitteilung von einem Objekt A, das ein Anwendungsprogramm ist, das auf dem Metaraum mCOOP läuft, zu einem Objekt B, das ein weiteres An-

wendungsprogramm ist, das auf dem gleichen Metaraum mCOOP läuft, geschickt. Fig. 5 zeigt ähnlich Fig. 3 den Prozess des Mitteilens vom Objekt A zum Objekt B zusammen mit der hierarchischen Struktur des Betriebssystems, während Fig. 6 ähnlich Fig. 4 ein Zeitdiagramm der Objektwechselwirkungen während des Sendens der Mitteilung vom Objekt A zum Objekt B zeigt.

**[0084]** Der Prozess ist wie folgt.

(1) Metaruf zu mCOOP-Mailer

**[0085]** Als erster Schritt wird ein Metaruf durch das Objekt A gemacht, um das Objekt „mCOOP-Mailer" im Metaraum mCOOP aufzurufen. So wird das Objekt „mCOOP-Mailer", das ein Mitteilen zwischen Objekten im Metaraum mCOOP unternimmt, aufgerufen, um ein Mitteilen vom Objekt A zum Objekt B zu ermöglichen.

**[0086]** In diesem Fall wird das Objekt „mCOOP-Mailer" durch den Metaruf vom Objekt A direkt aufgerufen, wie durch den Pfeil P1 angedeutet. Das Objekt A wartet unbedingt auf den Abschluss der Verarbeitung durch das Objekt „mCOOP-Mailer". So existiert eine Serialität im Ausführungsübergang vom Objekt A zum Objekt „mCOOP-Mailer", sodass der Aufruf gemacht wird, um das Objekt „mCOOP-Mailer" ohne die Arbeit des Aufrufens des Objekts „mCore-Scheduler", das im herkömmlichen Prozess zum Zweck der Ablaufverwaltung aufgerufen wird, direkt aufzurufen.

(2) Senden an mCOOP-Scheduler

**[0087]** Dann sendet das Objekt „mCOOP-Mailer", um das Objekt B zu starten, eine Mitteilung an das Objekt „mCOOP-Scheduler" im Metaraum mCOOP. Insbesondere erfordert das Starten des Objekts B eine Ablaufverwaltung von Objekten, die auf dem Metaraum mCOOP laufen. Die obige Mitteilung, die notwendig ist, um das Objekt B zu starten, wird deshalb an das Objekt „mCOOP-Scheduler" geschickt, das das Metaobjekt mit der Aufgabe der Ablaufverwaltung der Objekte im Metaraum mCOOP ist.

**[0088]** Genauer beginnt das Senden der Mitteilung vom Objekt „mCOOP-Mailer" zum Objekt „mCOOP-Scheduler" mit einem durch den Pfeil B2 angedeuteten Ruf zum Aufrufen des Objekts „mCore-Mailer", das das Metaobjekt mit der Aufgabe des Mitteilens zwischen im Metaraum mCore arbeitenden Objekten ist. Anschließend wird, wie durch den Pfeil B3 angedeutet, das Objekt „mCOOP-Scheduler" direkt aufgerufen und die zum Starten des Objekts B notwendige obige Mitteilung wird zu diesem Objekt „mCOOP-Scheduler" geschickt.

**[0089]** Es ist selbstverständlich, dass das Objekt „mCOOP-Mailer" unbedingt auf den Abschluss der durch das Objekt „mCOOP-Scheduler" durchgeführten Prozesses wartet. Somit existiert eine Serialität im Ausführungsübergang vom Objekt „mCOOP-Mailer" zum Objekt „mCOOP-Scheduler". Deshalb wird der Ruf B3 gemacht, um das Objekt „mCOOP-Scheduler" ohne die Arbeit des Rufens des Objekts „mCore-Scheduler", das im herkömmlichen Prozess zum Zweck der Ablaufverwaltung aufgerufen wird, direkt aufzurufen. Somit wird gemäß dem Ausführungsbeispiel der Erfindung das Mitteilen zwischen Objekten im gleichen Metaraum, wenn es eine Serialität im Ausführungsübergang gibt, so ausgeführt, dass der Schritt des Aufrufens des Abwicklers übersprungen wird. Dieser Stil des Mitteilens wird in der folgenden Beschreibung als „Schnellsenden" bezeichnet.

(3) Antworten an mCOOP-Mailer

**[0090]** Bei Empfang der Mitteilung ändert das Objekt „mCOOP-Scheduler" den Zustand des Objekts B entsprechend der empfangenen Mitteilung und sendet eine Antwort für diese Mitteilung durch „Schnellsenden" an das Objekt „mCOOP-Mailer".

**[0091]** Genauer wird dieser Antwortvorgang wie folgt durchgeführt. Um ein Senden der Antwort vom Objekt „mCOOP-Scheduler" zum Objekt „mCOOP-Mailer" zu ermöglichen, wird ein Ruf gemacht, wie durch den Pfeil B4 angedeutet, um das Objekt „mCore-Mailer" aufzurufen, das ein Mitteilen zwischen den auf dem Metaraum mCore laufenden Objekten unternimmt. Dann wird das Objekt „mCOOP-Mailer" direkt aufgerufen, wie durch den Pfeil B5 angedeutet, und die Antwort vom Objekt „mCOOP-Scheduler" wird zu diesem Objekt „mCOOP-Mailer" geschickt. Die Serialität existiert im Ausführungsübergang vom Objekt „mCOOP-Scheduler" zum Objekt „ mCOOP-Mailer". Deshalb wird der Ruf B5 gemacht, um ein direktes Senden der Antwort zum Objekt „ mCOOP-Mailer" ohne die Arbeit des Aufrufens des Objekts „mCore-Scheduler", das im herkömmlichen Prozess zum Zweck der Ablaufverwaltung aufgerufen wird, zu erlauben.

(4) Wiederaufnehmen der Basisobjekte

**[0092]** Die Verarbeitung in den Objekten A und B wird begonnen, wenn der durch das Objekt „mCOOP-Mailer" durchgeführte Prozess abgeschlossen ist.

**[0093]** Diese Behandlung beginnt mit einem durch den Pfeil B6 angedeuteten Ruf zum Aufrufen des Objekts „mCore-Scheduler". Nachdem eine Verarbeitung wie beispielsweise ein Umgruppieren durch das Objekt „mCore-Scheduler" durchgeführt ist, wird das Objekt B als ein Basisobjekt aufgerufen, wie durch den Pfeil B7 angedeutet, wodurch die Verarbeitung gestartet wird. Es ist jedoch zu beachten, dass die Verarbeitung zuerst mit dem Objekt A gestartet wird, d.h. der Pfeil B7 das Objekt A ruft, falls der Verarbeitung im Objekt A gegenüber der Verarbeitung im Objekt B eine Priorität gegeben worden ist.

**[0094]** Wie aus der obigen Beschreibung verständlich, verzichtet ein Ausführungsbeispiel der vorliegenden Erfindung auf die Schritte des Aufrufens des Abwicklers aufgrund der Nutzung der „Schnellsende"-Funktion und des Mechanismus, der einen direkten Ruf eines Metaobjekts ermöglicht. Somit reduziert das vorliegende Ausführungsbeispiel die Anzahl der durchzuführenden Aufgabenumschaltvorgänge, was eine Verbesserung in der Gesamtleistung des Systems gegenüber dem auf dem herkömmlichen Ablaufverwaltungsmechanismus beruhenden System bietet.

**[0095]** Obwohl in der obigen Beschreibung eine Mitteilungsverarbeitung als ein Beispiel einer Verarbeitung, die den Vorteil des Überspringens des Schritts des Aufrufens des Abwicklers genießt, speziell beschrieben ist, kann der gleiche Vorteil auch bei anderen Arten einer Verarbeitung erzielt werden. Zum Beispiel kann das Weglassen des Aufrufens des Abwicklers auch in einer Unterbrechungsverarbeitung bewirkt werden. In einem solchen Fall kann eine Unterbrechungslatenz aufgrund einer Beseitigung der zum Rufen des Abwicklers benötigten Zeit verkürzt werden. Somit kann das Weglassen des Rufens der Abwicklers in geeigneter Weise bei der Verarbeitung eingesetzt werden, die eine Bereitmeldungsausführung, wie beispielsweise eine als Reaktion auf eine Unterbrechung auszuführende Verarbeitung, eingesetzt werden.

2-4-2 Verzögerter Warteschlangenmechanismus

**[0096]** Ein bloßes Weglassen des Schritts des Rufens des Abwicklers kann jedoch eine Unstimmigkeit zwischen den tatsächlichen Zuständen von Objekten und jenen durch den Abwickler verstandenen bewirken, wodurch ein Risiko eines Fehlers im Systembetrieb entsteht. Gemäß dem vorliegenden Ausführungsbeispiel wird deshalb ein Mechanismus zum Gewährleisten des Zusammenpassens zwischen den tatsächlichen Zuständen von Objekten und den durch den Abwickler begriffenen verwendet. Dieser Mechanismus wird in dieser Beschreibung als ein „verzögerter Warteschlangenmechanismus" bezeichnet.

**[0097]** Vor der Beschreibung des verzögerten Warteschlangenmechanismus wird Bezug genommen auf **Fig. 7**, die die Zustände von Teilprozessen der Objekte veranschaulicht.

**[0098]** Ein Teilprozess ist jedem Objekt zugewiesen, und die durch den Abwickler durchgeführte Ablaufverwaltung wird auf der Teilprozessbasis ausgeführt. Es wird hier angenommen, dass der Teilprozess einen der drei Zustände „inaktiv", „ wartend" und „aktiv" annehmen kann. Der Status „inaktiv" bedeutet, dass der Teilprozess nicht aktiv ist, da das zugehörige Objekt nichts zu tun hat. Der Status „wartend" bedeutet, dass der Teilprozess nicht aktiv ist, da das zugehörige Objekt auf ein bestimmtes Ereignis wartet. Der Status „aktiv" bedeutet, dass das Objekt ausgeführt wird. Somit gibt der Status „aktiv" an, dass der Teilprozess aktiv ist oder dass der Teilprozess in einer Warteschlange unter der Steuerung des Abwicklers ist und auf eine Zuweisung der CPU-Verarbeitungszeit wartet. Der Abwickler ändert den Status des Teilprozesses eines Objekts und aktualisiert die Warteschlange im Teilprozess bei einer Aufforderung von einem anderen Objekt.

**[0099]** Es wird nun der verzögerte Warteschlangenmechanismus beschrieben.

**[0100]** Die Verwendung des verzögerten Warteschlangenmechanismus erfordert die Zuweisung einer Struktur zu jedem Teilprozess. Wenn ein Ausführungsübergang durch einen Aufgabenumschaltvorgang von einem Teilprozess (als ein „Ausgangsteil Prozess" bezeichnet) zu einem anderen Teilprozess (als ein „Zielteilprozess" bezeichnet) erfolgt, wird die Historie des Aufgabenumschaltens in der Struktur aufgezeichnet. Die „Historie des Aufgabenumschaltens" zeigt für jeden Teilprozess die Natur des Ausführungsübergangs sowie den Teilprozess, durch den jeder Teilprozess als Ergebnis des Ausführungsübergangs aktiviert worden ist. Somit enthält die Historie des Aufgabenumschaltens Informationen, die den Teilprozess identifizieren, der unmittelbar vor dem Ausführungsübergang ausgeführt worden ist, d.h. den Ausgangsteilprozess, Informationen, die den

Grund identifizieren, der den Zielteilprozess ausgelöst hat, z.B. einen Metaruf oder ein „Schnellsenden".

**[0101]** Wenn der Abwickler aufgerufen wird, liest der verzögerte Warteschlangenmechanismus die in der Struktur aufgezeichnete Historie, um so ein Zusammenpassen zwischen dem tatsächlichen Objektzustand und dem durch den Abwickler verstandenen Zustand zu erzielen. Insbesondere wird, wenn der Abwickler aufgerufen wird, der verzögerte Warteschlangenmechanismus gestartet, um die Historie der Aufgabenumschaltvorgänge, die seit dem letzten Ruf des Abwicklers durchgeführt wurden, zurückzuverfolgen. Der verzögerte Warteschlangenmechanismus ändert basierend auf der aufgezeichneten Historie den durch den Abwickler erkannten Objektzustand in Übereinstimmung mit dem tatsächlichen Objektzustand, wobei die Warteschlange in dem Teilprozess aktualisiert wird, wodurch ein Zusammenpassen zwischen dem tatsächlichen Objektzustand und dem durch den Abwickler verwalteten Objektzustand erzielt wird.

**[0102]** So sind die Teilprozesse, die unter den Betrieb des verzögerten Warteschlangenmechanismus gesetzt werden sollen, die Teilprozesse, die ohne den Schritt des Aufrufens des Abwicklers ausgelöst worden sind. Insbesondere enthalten diese Teilprozesse eine Kette von Teilprozessen von dem Teilprozess, der unmittelbar vor dem momentanen Aufruf des Abwicklers lief (dieser Teilprozess wird als „currentThread" bezeichnet) zurück zu dem Teilprozess, den der Abwickler als Reaktion auf den letzten Ruf ausgelöst hat (dieser Teilprozess wird als „activeThread" bezeichnet). So ist der durch den Abwickler als Reaktion auf den letzten Aufruf ausgelöste Teilprozess „activeThread" der Teilprozess, der durch den Abwickler als der derzeit laufende erkannt wird.

**[0103]** Der verzögerte Warteschlangenmechanismus führt eine Aktualisierung der Informationen betreffend den Teilprozesszustand, wie er durch den Abwickler erkannt wird, entsprechend der in der folgenden Tabelle 1 gezeigten Regel aus.

Tabelle 1

| | Ausgangsteilprozesszustand | | Zielteilprozesszustand | |
|---|---|---|---|---|
| | vor dem Aufgaben-umschalten | nach dem Aufgaben-umschalten | vor dem Aufgaben-umschalten | nach dem Aufgaben-umschalten |
| Metaruf | AKTIV | WARTEND | INAKTIV | AKTIV |
| Schnellsenden | AKTIV | AKTIV | INAKTIV | AKTIV |

**[0104]** So wird, falls der Zielteilprozess als Ergebnis eines durch einen Metaruf bewirkten Aufgabenumschaltens gestartet worden ist, der Ausgangsteilprozess nach diesem Aufgabenumschalten in den Zustand WARTEND gesetzt und der Zielteilprozess nach diesem Aufgabenumschalten wird in den Zustand AKTIV gesetzt. Falls dagegen der Zielteilprozess als Ergebnis eines durch ein Schnellsenden bewirktes Aufgabenumschalten gestartet worden ist, wird der Ausgangsteilprozess nach diesem Aufgabenumschalten in den Zustand AKTIV gesetzt und der Zielteilprozess nach diesem Aufgabenumschalten wird ebenfalls in den Zustand AKTIV gesetzt. Die Zustände des Ausgangsteilprozesses und des Zielteilprozesses sind unabhängig davon, ob das Aufgabenumschalten durch einen Metaruf oder ein Schnellsenden bewirkt worden ist, im Wesentlichen WARTEND.

**[0105]** Es wird nun eine detaillierte Beschreibung des Algorithmus des verzögerten Warteschlangenmechanismus unter spezieller Bezugnahme auf ein in **Fig. 8** dargestelltes Flussdiagramm gegeben. In diesem Flussdiagramm wird wie in einem in **Fig. 9** gezeigten Flussdiagramm der in jedem Schritt durchgeführte Prozess unter Verwendung von Ausdrücken beschrieben, die einer objektorientierten Programmiersprache C + + folgen.

**[0106]** Der verzögerte Warteschlangenmechanismus wird durch ein in dem Abwickler beschriebenes Verfahren „Scheduler::DelayedQueuing()" implementiert. Mit anderen Worten wird, wenn der Abwickler aufgerufen wird, das Verfahren „Scheduler::DelayedQueuing()" bevorzugt ausgeführt, um den verzögerten Warteschlangenmechanismus zu aktivieren.

**[0107]** In Schritt S1 dieses Verfahrens „Scheduler::DelayedQueuing()" wird ein Teilprozess „currentThread", der aktiv gewesen ist, unmittelbar bevor der Abwickler aufgerufen wurde, einer variablen „pThread" zugewie-

sen, die in dem verzögerten Warteschlangenmechanismus verwendet wird. Damit stellt die Variable „pThread", wenn Schritt S1 abgeschlossen ist, den Teilprozess dar, der aktiv gewesen ist, als der Abwickler gerufen wurde.

**[0108]** In Schritt S2 wird „pThread" mit „activeThread" verglichen, welches der Teilprozess ist, der durch den Abwickler als Reaktion auf den letzten Ruf des Abwicklers aktiviert wurde, d.h. der Teilprozess, der durch den Abwickler als derzeit aktiv erkannt wird. Falls „pThread" und „activeThread" nicht gleich sind, d.h. wenn „pThread" nicht den Teilprozess anzeigt, der das letzte Mal durch den Abwickler aktiviert wurde, geht der Prozess weiter zu Schritt S3. Falls jedoch „pThread" und „activeThread" gleich sind, d.h. wenn „pThread" den Teilprozess anzeigt, der das letzte Mal durch den Abwickler aktiviert wurde, springt der Prozess zu Schritt S10.

**[0109]** Schritt S3 bestimmt, ob der Ruf des durch „pThread" angegebenen Teilprozesses durch einen Metaruf oder durch ein „Schnellsenden" aufgerufen worden ist. Der Prozess geht weiter zu Schritt S4, falls in Schritt S3 bestimmt wird, dass der Ruf durch einen Metaruf bewirkt wurde, sonst geht der Prozess weiter zu Schritt S7. Damit wird Schritt S7 ausgeführt, wenn in Schritt S3 bestimmt wird, dass der Ruf durch ein „Schnellsenden" aufgerufen wurde.

**[0110]** Die Tatsache, dass der durch „pThread" angegebene Teilprozess durch einen Metaruf aufgerufen worden ist, bedeutet, dass dieser Teilprozess der Ausgangsteilprozess ist, der ein Aufgabenumschalten von dem Basisobjekt zu dem Metaobjekt verursacht hat. Schritt S4 setzt deshalb an diesem Teilprozess einen Warteschlangenaktualisierungsmerker. In **Fig. 8** ist der die Warteschlange aktualisierende Merker als „HOOK-Merker" ausgedrückt. Der die Warteschlange aktualisierende Merker wird als ein asynchroner Abwicklermechanismus verwendet, der später beschrieben wird. Nach dem Setzen des Merkers in Schritt S4, geht der Prozess weiter zu Schritt S5.

**[0111]** So wird Schritt S5 ausgeführt, weil der durch „pThread" angegebene Teilprozess durch einen Metaruf aufgerufen worden ist. Deshalb wird in Schritt S5 der Zustand des Teilprozesses, der unmittelbar vor dem durch „pThread" angegebenen Teilprozess ausgeführt wurde, entsprechend der in Tabelle 1 gezeigten Regel auf WARTEND gesetzt. Nach Abschluss der Verarbeitung in Schritt S5 geht der Prozess weiter zu Schritt S6.

**[0112]** In Schritt S6 wird der Inhalt von „pThread" so geändert, dass „pThread" einen Teilprozess angibt, der dem Teilprozess unmittelbar vorangeht, der durch „pThread" angegeben worden ist, zum Zweck eines Rückverfolgens der Aufgabenhistorie. Der Prozess kehrt dann zu Schritt S2 zurück, um die beschriebenen Schritte zu wiederholen.

**[0113]** In Schritt S7 wird wie im Fall von Schritt S4 ein Warteschlangenmerker auf den durch „pThread" angegebenen Teilprozess gesetzt. Nach Abschluss der Verarbeitung in Schritt S7 geht der Prozess weiter zu Schritt S8.

**[0114]** So wird Schritt S8 ausgeführt, weil der durch „pThread" angegebene Teilprozess durch ein „Schnellsenden" aufgerufen worden ist. Deshalb wird in Schritt S8 der Zustand des Teilprozesses, der dem durch „pThread" angegebenen Teilprozess unmittelbar vorangeht, entsprechend der in Tabelle 1 gezeigten Regel auf AKTIV gesetzt. Nachdem die Verarbeitung in Schritt S8 beendet ist, geht der Prozess weiter zu S9.

**[0115]** In Schritt S9 wird der Teilprozess, der dem durch „pThread" angegebenen Teilprozess unmittelbar vorangeht, d.h. der Teilprozess, dessen Zustand in Schritt S8 auf AKTIV gesetzt wurde, der Warteschlange des Teilprozesses hinzugefügt, der unter der Verwaltung des Abwicklers ist. Nach Beendigung der Bearbeitung in Schritt S9, geht der Prozess weiter zu Schritt S6.

**[0116]** Wie oben beschrieben, wird in Schritt S6 der Inhalt von „pThread" zum Zweck der Rückverfolgung der Aufgabenhistorie so geändert, dass „pThread" einen Teilprozess angibt, der dem Teilprozess unmittelbar vorangeht, der durch „pThread" angegeben worden ist. Der Prozess kehrt dann zu Schritt S2 zurück, um die beschriebenen Schritte zu wiederholen.

**[0117]** Wie oben erläutert, springt der Prozess auf einmal zu Schritt S10, falls der in Schritt S2 ausgeführte Vergleich angibt, dass „pThread" den gleichen Teilprozess wie den zuletzt durch den Abwickler aktivierten angibt. In Schritt S10 wird „currentThread" dem „activeThread" zugewiesen. „activeThread" ist der Teilprozess, der durch den Abwickler als aktiv erkannt wird, während „currentThread" der Teilprozess ist, der aktiv war, unmittelbar bevor der Abwickler aufgerufen wurde. Somit ist nach Abschluss der in Schritt S10 ausgeführten Verarbeitung der durch den Abwickler als aktiv erkannte Teilprozess identisch zu dem Teilprozess, der aktiv war,

unmittelbar bevor der Abwickler aufgerufen wird.

**[0118]** Man erkennt, dass ein Überspringen der Schritte S3 bis S9 von Schritt S2 zu Schritt S10 nur stattfindet, wenn „currentThread" und „activeTread" von Beginn an gleich sind, d.h. wenn der Teilprozess, der den Abwickler aufgerufen hat, gleich dem Teilprozess ist, der durch den Abwickler zuletzt aufgerufen wurde.

**[0119]** Wenn kein Aufgabenumschalten, das ein Aufrufen des Abwicklers weglässt, in der Aufgabenumschalthistorie existiert, sind der Teilprozess, durch den der Abwickler nun aufgerufen wird, und der Teilprozess, der durch den Abwickler zuletzt aufgerufen wurde, gleich. Wenn dagegen die Aufgabenumschalthistorie ein Aufgabenumschalten enthält, das ein Aufrufen des Abwicklers weggelassen hat, sind der Teilprozess, durch den der Abwickler nun aufgerufen wird, und der Teilprozess, der durch den Abwickler zuletzt aufgerufen wurde, nicht gleich.

**[0120]** Daher sind „currentThread" und „activeThread" von Beginn des Prozesses an gleich, falls es kein Aufgabenumschalten gibt, das den Aufruf des Abwicklers weglässt. In einem solchen Fall überspringt der Prozess die Schritte S3 bis S9 von Schritt S2, der Schritt S1 folgt, zu Schritt S10. In diesem Fall wird in Schritt S10 in Wirklichkeit keine Verarbeitung ausgeführt, weil „activeThread" und „currentThread" bereits gleich gewesen sind. Wenn dagegen ein Aufgabenumschalten existiert, das ohne Aufrufen des Abwicklers ausgeführt wurde, sind „currentThread" und „activeThread" nicht gleich, sodass die Schritte S3 und folgende ausgeführt werden, um die Historie des Aufgabenumschaltvorgänge zurückzuverfolgen, um das Aufgabenumschalten zu lokalisieren, das ohne Aufrufen des Abwicklers ausgeführt wurde.

**[0121]** Eine Änderung und ein Aktualisieren des Teilprozesses und der Warteschlange, die beim vorherigen Aufgabenumschalten weggelassen wurden, werden deshalb abgeschlossen, wodurch ein Zusammenpassen zwischen dem tatsächlichen Objektzustand und dem durch den Abwickler verstandenen Objektzustand erzielt wird.

**[0122]** Es ist für den Fachmann klar, dass der oben beschriebene verzögerte Warteschlangenmechanismus auch in dem Fall verwendet werden kann, wenn das Rufen des Abwicklers wegen einer Notwenigkeit eines dringenden Prozesses wie beispielsweise einer Unterbrechung weggelassen worden ist, obwohl in dem beschriebenen Ausführungsbeispiel der verzögerte Warteschlangenmechanismus in dem Fall verwendet wird, wenn das Rufen des Abwicklers im Lauf der Mitteilung weggelassen wurde.

**[0123]** Wenn die Serialität des Aufgabenumschaltens zwischen Objekten des gleichen Metaraums als Abwickler durch die Veränderung des Zustands des Objekts, die durch den verzögerten Warteschlangenmechanismus durchgeführt wird, nicht beeinflusst wird, kann „currentThread" durch ein Verfahren „Scheduler::ChecklnvokeO", das einem Ablauf wie in **Fig. 9** dargestellt folgt, vor der durch „Scheduler::DelayedQueue()" durchgeführten Verarbeitung gesetzt werden.

**[0124]** Das Verfahren „Scheduler::ChecklnvokeO" beginnt mit Schritt S21, der dem „pThread" ein „SchedulerThread" zuweist. „SchedulerThread" bedeutet den Teilprozess des Abwicklers. Damit gibt „pThread" nach Abschluss von Schritt S21 den Teilprozess des Abwicklers an.

**[0125]** Schritt S22 bestimmt, ob der Ruf des durch „pThread" angegebenen Teilprozesses durch ein „Schnellsen den" bewirkt worden ist oder nicht. Falls die Antwort Y ist, d.h. falls ein „Schnellsen den" diesen Teilprozess aufgerufen hat, geht der Prozess weiter zu Schritt S23.

**[0126]** Die Tatsache, dass der durch „pThread" angegebene Teilprozess durch ein „Schnellsenden" aufgerufen worden ist, bedeutet, dass dieser Teilprozess und der Teilprozess, der diesem Teilprozess unmittelbar vorangeht, Teilprozesse von Metaobjekten des gleichen Metaraums als Abwickler sind. Daher wird Schritt S23 nur ausgeführt, wenn der durch „pThread" angegebene Teilprozess und der unmittelbar vorhergehende Teilprozess die Teilprozesse von Metaobjekten des gleichen Metaraums als Abwickler sind. In diesem Fall kann deshalb „currentThread", was den verzögerten Warteschlangenmechanismus wirken lässt, zu dem Teilprozess zurückverfolgen, der dem „currentThread" unmittelbar vorangeht. Deshalb wird in Schritt S23 „pThread" aktualisiert, um den Teilprozess anzugeben, der dem Teilprozess unmittelbar vorangeht, der durch „pThread" angegeben worden ist. Der Prozess kehrt dann zu Schritt S22 zurück, um die Schritte S22 und S23 zu wiederholen.

**[0127]** Wenn dagegen in Schritt S22 bestimmt wird, dass das Rufen des durch „pThread" angegebenen Teilprozesses nicht durch ein „Schnellsenden" erfolgte, springt der Prozess zu Schritt S24. Man erkennt, dass die-

ser Sprung durchgeführt wird, wenn der Teilprozess, der dem durch „ pThread" angegebenen Teilprozess unmittelbar vorangeht, nicht der Teilprozess eines Metaobjekts des gleichen Metaraums als Abwickler ist.

**[0128]** In Schritt S24 wird „pThread" „currentThread" zugewiesen, sodass der durch „pThread" angegebene Teilprozess auf den durch „currentThread" angegebenen Teilprozess gesetzt wird. Wenn die Verarbeitung von Schritt S24 beendet ist, ist deshalb der Teilprozess, der dem durch „currentThread" dargestellten Teilprozess unmittelbar vorangeht, nicht der Teilprozess eines Objekts des gleichen Metaraums als Abwickler.

**[0129]** Nach Abschluss der Einstellung von „currentThread" durch Ausführen von Schritt S24 ist der Prozess des Verfahrens „Scheduler::ChecklnvokeO" beendet und der Prozess geht weiter zur Verarbeitung des obigen Verfahrens „Scheduler::DelayedQueuing()".

**[0130]** So wird die Verarbeitung des Verfahrens „Scheduler::ChecklnvokeO" nach der Einstellung von „currentThread" ausgeführt. Diese Verarbeitungsabfolge reduziert die Verarbeitung zum Aktualisieren des Zustands des Abwicklers durch Rückverfolgen der Aufgabenumschalthistorie, was zur Verbesserung der Effizienz der Verarbeitung beträgt.

2-4-3 Asynchroner Abwickleraufrufmechanismus

**[0131]** Ein Warteschlangenaktualisierungsmerker wird gesetzt, wenn die Warteschlange durch den verzögerten Warteschlangenmechanismus verändert wird, und ein Umgruppieren wird durch asynchrones Rufen des Abwicklers basierend auf dem Merker durchgeführt. Insbesondere wird, wenn der verzögerte Warteschlangenmechanismus aktiviert wird, ein Warteschlangenaktualisierungsmerker für ein Ausgangsschalten gesetzt, das ein Aufgabenumschalten auslöst, das ein Umgruppieren erfordert. Wenn dieses Aufgabenumschalten aktiviert wird, wird der Abwickler aufgerufen, um das Umgruppieren durchzuführen. Dieser Mechanismus wird als ein „asynchroner Abwickleraufrufmechanismus" bezeichnet.

**[0132]** Der Prozess der durch den asynchronen Abwickleraufrufmechanismus durchgeführten Verarbeitung ist wie folgt.

(1) Wenn ein Aktualisieren der Warteschlange durch den verzögerten Warteschlangenmechanismus bewirkt wird, wird ein Warteschlangenaktualisierungsmerker für einen Ausgangsteilprozess gesetzt, der ein Aufgabenumschalten von einem Metaobjekt zu einem Basisobjekt auslöst. Das Setzen dieses Merkers wird in den Schritten S4 und S7 des zuvor unter Bezug auf **Fig. 8** beschriebenen Ablaufs durchgeführt.

(2) Wenn ein Aufgabenumschalten von einem Metaobjekt zu einem Basisobjekt stattgefunden hat, erfolgt eine Prüfung, ob ein Warteschlangenaktualisierungsmerker an dem Ausgangsteilprozess, d.h. dem Teilprozess des Metaobjekts, vor dem Aktivieren des Zielteilprozesses, d.h. des Teilprozesses des Basisobjekts, gesetzt worden ist oder nicht. Wenn ein solcher Merker an dem Ausgangsteilprozess bestätigt wird, wird anstelle des Zielteilprozesses der Abwickler aufgerufen. Der Abwickler führt ein Umgruppieren vor dem Aufgabenumschalten vom Metaobjekt zum Basisobjekt durch.

**[0133]** Das Setzen des Warteschlangenaktualisierungsmerkers wird in Abhängigkeit von den Umständen, unter denen die Warteschlange durch den Abwickler geändert wird, wie folgt durchgeführt.

(1) Wenn der verzögerte Warteschlangenmechanismus durch ein Objekt „mCore-Scheduler" ausgeführt worden ist, wird ein Warteschlangenaktualisierungsmerker an den Teilprozess gesetzt, der ein Aufgabenumschalten von einem den Metaraum mCore bildenden Objekt zu einem Objekt, das auf dem Metaraum mCore läuft, verursacht. Dies geschieht auch in dem Fall, wenn der Abwickler gemeinsam durch alle Metaräume besetzt wird, wenn der Abwickler als ein Metaobjekt des Metaraums mCore wirkt, um den verzögerten Warteschlangenmechanismus auszuführen.

(2) Wenn der verzögerte Warteschlangenmechanismus durch ein Objekt „mCOOP-Scheduler" ausgeführt worden ist, das durch ein Metaobjekt des Metaraums mCore aufgerufen wurde, wird ein Warteschlangenaktualisierungsmerker an den Teilprozess gesetzt, der ein Aufgabenumschalten von einem Metaobjekt, das den Metaraum mCOOP bildet, zu einem Objekt, das auf dem Metaraum mCOOP läuft, bewirkt. Dies geschieht auch in dem Fall, wenn der Abwickler gemeinsam von allen Metaräumen besessen wird, wenn der Abwickler als ein Metaobjekt des Metaraums mCOOP wirkt, um den verzögerten Warteschlangenmechanismus auszuführen. Das Setzen des Merkers wird durch Verändern des Teilprozesses, der den verzögerten Warteschlangenmechanismus initiiert, mittels des Verfahrens „Scheduler::ChecklnvokeO", das in **Fig. 9** dargestellt ist, ausgeführt.

**[0134]** Es folgt eine detailliert Beschreibung des asynchronen Abwickleraufrufmechanismus, der den oben beschriebenen Warteschlangenaktualisierungsmerker verwendet. Als Beispiel erfolgt die Beschreibung an ei-

nem Fall, wenn eine Mitteilung von einem Objekt A, das auf dem Metaraum mCOOP läuft, zu einem Objekt C, das auf dem Metaraum mDrive läuft, gesendet wird, es wird auch angenommen, dass der Abwickler von allen Metaräumen gemeinsam besessen wird. Das heißt, die Ablaufverwaltung wird auf allen Metaräumen mittels des gemeinsamen Objekts „Scheduler" ausgeführt.

**[0135]** Fig. 10 und Fig. 11 zeigen die Prozedur des Mitteilens vom Objekt A zum Objekt C. Insbesondere zeigt Fig. 10 wie in den Fällen von Fig. 3 und Fig. 5 die Prozedur zusammen mit der hierarchischen Struktur des Betriebssystems. Fig. 11 ist ein Zeitdiagramm, das wie in den Fällen von Fig. 4 und Fig. 6 die während des Weitergebens der Mitteilung, welche in diesem Fall vom Objekt A zum Objekt C geschickt wird, durchgeführte Objektwechselwirkung zeigt.

**[0136]** Die Prozedur ist wie folgt.

(1) Metaruf an mCOOP-Mailer

Das Objekt A ruft ein Objekt „mCOOP-Mailer" des Metaraums mCOOP durch einen Metaruf auf. So wird das Objekt „mCOOP-Mailer", das das durch ein auf den Metaraum mCOOP laufendes Objekt durchgeführte Mitteilen unternimmt, aufgerufen, um das Mitteilen vom Objekt A zum Objekt B zu ermöglichen.

In diesem Fall wird das Objekt „mCOOP-Mailer" direkt durch den Metaruf vom Objekt A aufgerufen, wie durch den Pfeil C1 angegeben. Somit wird das Objekt „mCOOP-Mailer" ohne die Notwendigkeit eines Aufrufs des Objekts „Scheduler" zum Zweck der Ablaufverwaltung direkt aufgerufen, weil eine Serialität in dem Ausführungsübergang vom Objekt A zum Objekt „mCOOP-Mailer" existiert.

(2) Senden an mDrive-Mailer

Das Objekt C als Mitteilungszielobjekt ist auf dem Metaraum mDrive. Deshalb wird eine Mitteilung, die das Mitteilen an das Objekt C anfordert, zum Objekt „mDrive-Mailer" geschickt, das ein Metaobjekt verantwortlich für das Mitteilen durch Objekte, die auf dem Metaraum mDrive laufen, ist.

Insbesondere wird, um ein Mitteilen vom Objekt „mCOOP-Mailer" zum Objekt „mDrive-Mailer" zu ermöglichen, ein Ruf ausgeführt, wie durch den Pfeil C2 angedeutet, um das Objekt „mCore-Mailer" aufzurufen, das das Metaobjekt verantwortlich zum Mitteilen zwischen auf dem Metaraum mCore laufenden Objekten ist. Anschließend wird, wie durch den Pfeil C3 angegeben, das Objekt „mDrive-Scheduler" direkt aufgerufen und die Mitteilung zu diesem Objekt „mDrive-Scheduler" geschickt. Es ist zu beachten, dass das Objekt „mDrive-Scheduler" ohne die Notwendigkeit des Rufs des „Scheduler" zum Zweck der Ablaufverwaltung direkt aufgerufen wird, weil eine Serialität in dem Ausführungsübergang vom Objekt „mCOOP-Mailer" zum Objekt „mDriveScheduler" existiert. Mit anderen Worten erreicht ein „Schnellsenden" das Mitteilen vom Objekt „mCOOP-Mailer" zum Objekt „mDrive-Mailer".

(3) Senden an Scheduler

Dann sendet das Objekt „mDrive-Mailer" eine Mitteilung an das Objekt „Scheduler", um das Objekt C zu aktivieren. Die Aktivierung des Objekts C erfordert eine Ablaufverwaltung. Diesbezüglich wird die zum Aktivieren des Objekts C notwendige Mitteilung zum Objekt „Scheduler" geschickt, dass ein Metaobjekt verantwortlich für die Ablaufverwaltung ist.

Insbesondere wird zum Zweck des Ermöglichens des Mitteilens vom Objekt „mDrive-Mailer" zum Objekt „Scheduler" ein Ruf durchgeführt, wie durch den Pfeil C4 angegeben, um das Objekt „mCore-Mailer" aufzurufen, das das Metaobjekt ist, das das Mitteilen zwischen auf dem Metaraum mCore laufenden Objekten unternimmt. Anschließend wird das Objekt „Scheduler" direkt aufgerufen, wie durch den Pfeil C5 angegeben, und die zum Aktivieren des Objekts C notwendige Mitteilung wird zu diesem Objekt „Scheduler" geschickt. Es ist zu beachten, dass das Objekt „Scheduler" ohne den Schritt der Ablaufsteuerung direkt aufgerufen wird, weil eine Serialität in dem Ausführungsübergang vom Objekt „mCOOP-Mailer" zum Objekt „Scheduler" existiert. Daher wird das Mitteilen vom Objekt „mDrive-Mailer" zum Objekt „Scheduler" durch „Schnellsenden" durchgeführt.

(4) Aktualisieren der Ablaufwarteschlange

Dann ändert das Objekt „Scheduler" den Zustand des Objekts C basierend auf der empfangenen Mitteilung und lässt den verzögerten Warteschlangenmechanismus die Warteschlange ändern.

Das Objekt „Scheduler" ist zum Zweck des Änderns des Zustands des Objekt C aufgerufen worden und wirkt deshalb als ein Metaobjekt des Metaraums mDrive. Deshalb setzt das Objekt „Scheduler" einen Warteschlangenaktualisierungsmerker an einen Teilprozess, der ein Aufgabenumschalten zum Wiederaufnehmen einer Ausführung eines auf dem Metaraum mDrive laufenden Objekts oder ein Aufgabenumschalten zum Wiederaufnehmen einer Ausführung eines auf dem Metaraum mCOOP, der vom gleichen hierarchischen Niveau wie der Metaraum mDrive ist, laufenden Objekts aktiviert. Insbesondere setzt das Objekt „Scheduler" einen Warteschlangenaktualisierungsmerker an den Teilprozess des Objekts „mCOOP-Mailer".

Dieser Vorgang kann praktisch unter Verwendung des Algorithmus des Verfahrens „Scheduler::ChecInvoke()" durch Verfolgen des Objektausführungsübergangs in der folgenden Abfolge und anschließendes Aus-

führen des verzögerten Warteschlangenmechanismus, beginnend vom Objekt „mCore-Mailer(2)" implementiert werden: Objekt „Scheduler" – Objekt „mCore-Mailer(1)" – Objekt „mDrive-Mailer" – Objekt „mCore-Mailer(2)" – Objekt „mCOOP-Mailer" – Objekt A.

Das Objekt „mCore-Mailer(1)" und das Objekt „mCore-Mailer(2)" sind das gleiche Objekt. Die Anhänge (1) und (2) sind angefügt, um den Unterschied in der Zeit, zu welcher dieses Objekt aufgerufen wird, zu zeigen.

(5) Anschließend sendet das Objekt „Scheduler" eine Antwort an das Objekt „mDrive-Mailer" als Reaktion auf die Mitteilung, die durch „Schnellsenden" vom Objekt „mDrive-Mailer" zum Objekt „Scheduler" geschickt worden ist. Insbesondere wird, um das Schicken der Antwort vom Objekt „Scheduler" zum Objekt „ mDrive" zu ermöglichen, ein Ruf durchgeführt, wie durch den Pfeil C6 angedeutet, um das Objekt „mCore-Mailer" aufzurufen, das ein Metaobjekt ist, das ein Mitteilen zwischen auf dem Metaraum mCore laufenden Objekten unternimmt. Dann wird das Objekt „mDrive-Mailer" direkt aufgerufen, wie durch den Pfeil C7 angedeutet, und die Antwort vom Objekt „Scheduler" wird zu diesem Objekt „mDrive" geschickt. Die Antwort wird ohne den Schritt der Ablaufsteuerung direkt zum Objekt „mDrive-Mailer" geschickt, wegen der Serialität des Ausführungsübergangs vom Objekt „Scheduler" zum Objekt „mDrive-Mailer". Die Ausführung des Objekts „mDrive-Mailer" wird somit wieder aufgenommen.

(6) Antwort an mCOOP-Mailer

Dann sendet das Objekt „mDrive-Mailer" eine Antwort an das Objekt „mCOOP-Mailer" als Reaktion auf die Mitteilung, die durch „Schnellsenden" vom Objekt „mCOOP-Mailer" geschickt worden ist. Insbesondere wird, um das Senden der Antwort vom Objekt „mDrive" zum Objekt „mCOOP-Mailer" zu ermöglichen, ein Ruf durchgeführt, wie durch den Pfeil C8 angegeben, um das Objekt „ mCore-Mailer" aufzurufen, das ein Metaobjekt ist, das ein Mitteilen zwischen auf dem Metaraum mCore laufenden Objekten unternimmt. Dann wird das Objekt „mCOOP-Mailer" direkt aufgerufen, wie durch den Pfeil C9 angegeben, und die Antwort vom Objekt „mDrive-Mailer" wird zu diesem Objekt „mCOOP-Mailer" geschickt. Die Antwort wird wegen der Serialität des Ausführungsübergangs vom Objekt „mDrive-Mailer" zum Objekt „mCOOP-Mailer" ohne den Schritt der Ablaufsteuerung direkt zum Objekt „mCOOP-Mailer" geschickt. Die Ausführung des Objekts „mCOOP-Mailer" wird so wieder aufgenommen.

(7) Wiederaufnehmen des Basisobjekts

Nach Abschluss der durch das Objekt „mCOOP-Mailer" durchgeführten Verarbeitung muss eine Aufgabenumschaltverarbeitung durchgeführt werden, um die Aufgabe zum Objekt A umzuschalten, um das Objekt A wieder aufzunehmen. Es ist jedoch zu erkennen, dass ein Warteschlangenaktualisierungsmerker an den Teilprozess des Objekts „mCOOP-Mailer", das der Ausgangsteilprozess dieses Aufgabenumschaltens ist, gesetzt worden ist. Deshalb wird anstelle des Aufgabenumschaltens zum Objekt A tatsächlich eine Aktivierung des asynchronen Rufmechanismus durchgeführt. Das heißt, die Wiederaufnahme des Objekts A wird nicht durchgeführt, sondern das Objekt „Scheduler" wird aufgerufen, wie durch den Pfeil C10 angegeben.

(8) Umgruppieren

Ein Umgruppieren wird deshalb durch das Objekt „Scheduler" durchgeführt. Es wird hierbei angenommen, dass der Ablauf der Warteschlange aktualisiert worden ist, um das Objekt C vor der Ausführung des Objekts A ausführen zu lassen. Ein Umgruppieren wird daher durch das Objekt „Scheduler" derart durchgeführt, dass das Objekt C als nächstes ausgeführt wird. Das Objekt C wird deshalb aufgerufen, wie durch den Pfeil C11 angegeben, und dann ausgeführt.

**[0137]** Man erkennt, dass die Anzahl der Aufrufe des Abwicklers, d.h. des Objekts „Scheduler" beträchtlich reduziert werden kann. Dies liegt an der Verwendung des asynchronen Abwickleraufrufmechanismus beim Mitteilungsvorgang zum Senden der Mitteilung vom Objekt A zum Objekt C. Insbesondere wird in dem dargestellten Ausführungsbeispiel die Ablaufsteuerung für die Objekte, die auf dem Metaraum „mCOOP" oder „mDrive" laufe n, verzögert und der Abwickler wird in einer asynchronen Weise aufgerufen, wenn ein Aufgabenumschalten stattgefunden hat, um den Teilprozess des Objekts A wieder aufzunehmen. Es ist zu erkennen, dass die Anzahl der Aufrufe des Abwicklers im Vergleich zu dem Fall, wenn der Abwickler jedes Mal gerufen wird, wenn das Umschalten zum Antworten auf eine Mitteilung des „Schnellsendens" durchgeführt wird, um zwei reduziert ist.

**[0138]** In dem oben beschriebenen Ausführungsbeispiel wird ein Aktualisieren der Warteschlange unter der Steuerung des Abwicklers nur für die Objekte bewirkt, die auf dem Metaraum „mCOOP" und dem Metaraum „mDrive" laufen. Daher beeinflusst das Aktualisieren das Objekt, das auf dem Metaraum „mCore" läuft, überhaupt nicht. Die Verringerung der Anzahl der Aufrufe des Abwicklers, geleistet durch das Verzögern der Ablaufsteuerung für die auf den Metaräumen „mCOOP" und „mDrive" laufende n Objekte, erzeugt keine unerwünschte Wirkung auf den Betrieb des gesamten Systems.

**[0139]** Es ist aus der obigen Beschreibung verständlich, dass Ausführungsbeispiele der vorliegenden Erfindung die Anzahl von Aufrufen des Abwicklers reduzieren, während ein korrekter Betrieb des gesamten Sys-

tems gewährleistet ist, wodurch eine Verbesserung der Gesamtleistung des gesamten Systems geboten wird, während die Benutzung einer Nano-Kernroutine erlaubt ist.

**[0140]** Obwohl die Erfindung durch ihre bevorzugten Formen beschrieben worden ist, ist es selbstverständlich, dass das beschriebene Ausführungsbeispiel nur beispielhaft ist und verschiedene Änderungen und Modifikationen daran ohne Verlassen des Schutzumfangs der Erfindung vorgenommen werden können.

## Patentansprüche

1. Datenverarbeitungsverfahren, mit den Schritten:
wenn ein Aufgabenumschalten stattgefunden hat, um den auszuführenden Teilprozess umzuschalten, Aufzeichnen der Historie des Aufgabenumschaltens; und
wenn ein Abwickler zum Steuern der Ausführungsabfolge von Teilprozessen aufgerufen wird, Aktualisieren einer durch den Abwickler verwalteten Warteschlange durch Zurückverfolgen der aufgezeichneten Historie des Aufgabenumschaltens.

2. Datenverarbeitungsverfahren nach Anspruch 1, bei welchem der Schritt des Aktualisierens der Warteschlange enthält:
Setzen eines vorbestimmten Merkers auf einen Teilprozess, der ein Aufgabenumschalten bewirkt, das eine Veränderung der Reihenfolge in der Warteschlange erfordert; und
Aufrufen des Abwicklers als Reaktion auf ein Auftreten eines neuen Aufgabenumschaltens, um die Reihenfolge in der Warteschlange zu verändern, wenn der Teilprozess, der das neue Aufgabenumschalten bewirkt hat, diesen Merker trägt.

3. Datenverarbeitungsverfahren nach Anspruch 2, bei welchem sich der Teilprozess, auf den der Merker gesetzt ist, entsprechend der Art des Aufrufs des Abwicklers unterscheidet.

4. Computerlesbares Aufzeichnungsmedium mit einem Betriebssystem, das einen Abwickler zum Steuern der Ausführungsreihenfolge von Teilprozessen besitzt, wobei das Betriebssystem ein Datenverarbeitungsverfahren implementiert, das aufweist:
wenn ein Aufgabenumschalten stattgefunden hat, um den auszuführenden Teilprozess umzuschalten, Aufzeichnen der Historie des Aufgabenumschaltens; und
wenn ein Abwickler zum Steuern der Ausführungsreihenfolge von Teilprozessen aufgerufen wird, Aktualisieren einer durch den Abwickler verwalteten Warteschlange durch Zurückverfolgen der aufgezeichneten Historie des Aufgabenumschaltens.

5. Computerlesbares Aufzeichnungsmedium nach Anspruch 4, bei welchem der Schritt des Aktualisierens der Warteschlange enthält:
Setzen eines vorbestimmten Merkers auf einen Teilprozess, der ein Aufgabenumschalten bewirkt, welches eine Veränderung der Reihenfolge in der Warteschlange erfordert; und
Aufrufen des Abwicklers als Reaktion auf ein Auftreten eines neuen Aufgabenumschaltens, um die Reihenfolge in der Warteschlange zu verändern, wenn der Teilprozess, der das neue Aufgabenumschalten bewirkt hat, diesen Merker trägt.

6. Computerlesbares Aufzeichnungsmedium nach Anspruch 4, bei welchem sich der Teilprozess, auf den der Merker gesetzt ist, entsprechend der Art des Aufrufs des Abwicklers unterscheidet.

7. Datenverarbeitungsvorrichtung, mit
einer Einrichtung zum Aufzeichnen der Historie eines Aufgabenumschaltens, wenn ein Aufgabenumschalten stattgefunden hat, um den auszuführenden Teilprozess umzuschalten; und
einer Einrichtung zum Aktualisieren einer durch einen Abwickler verwalteten Warteschlange, wenn der Abwickler zum Steuern der Ausführungsreihenfolge von Teilprozessen aufgerufen wird, durch Zurückverfolgen der aufgezeichneten Historie des Aufgabenumschalters.

8. Datenverarbeitungsvorrichtung nach Anspruch 7, bei welchem die Einrichtung zum Aktualisieren der Warteschlange enthält:
eine Einrichtung zum Setzen eines vorbestimmten Merkers auf einen Teilprozess, der ein Aufgabenumschalten bewirkt, das eine Veränderung der Reihenfolge in der Warteschlange erfordert; und
eine Einrichtung zum Aufrufen des Abwicklers als Reaktion auf ein Auftreten eines neuen Aufgabenumschaltens, um die Reihenfolge in der Warteschlange zu verändern, wenn der Teilprozess, der das neue Aufgabe-

numschalten bewirkt hat, diesen Merker trägt.

9. Datenverarbeitungsvorrichtung nach Anspruch 8, bei welchem der Teilprozess, auf den der Merker gesetzt ist, sich entsprechend der Art des Aufrufs des Abwicklers unterscheidet.

Es folgen 14 Blatt Zeichnungen

**Anhängende Zeichnungen**

FIG. 1

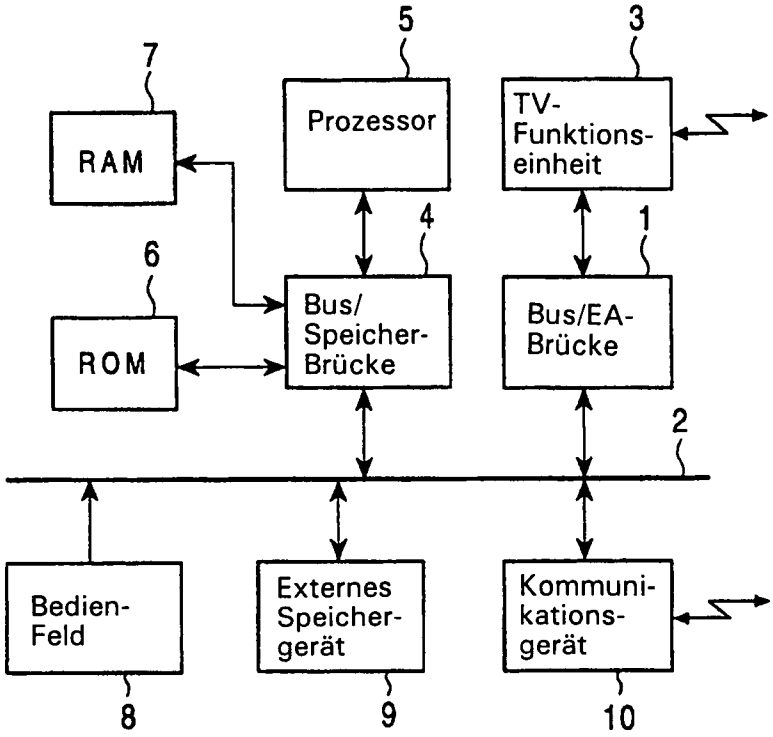


FIG. 2

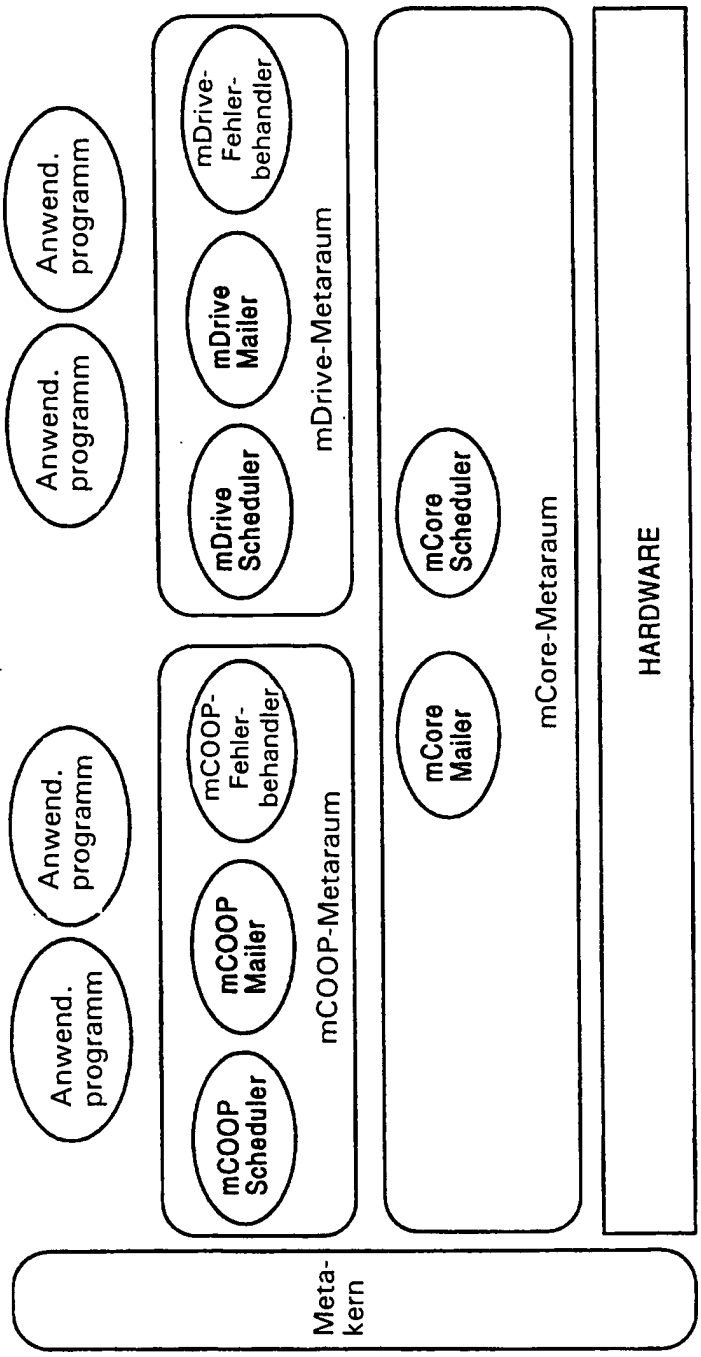

Anwend. programm
Anwend. programm
Anwend. programm
Anwend. programm
mCOOP Scheduler
mCOOP Mailer
mCOOP-Fehler-behandler
mCOOP-Metaraum
mDrive Scheduler
mDrive Mailer
mDrive-Fehler-behandler
mDrive-Metaraum
mCore Mailer
mCore Scheduler
mCore-Metaraum
Meta-kern
HARDWARE

FIG. 3

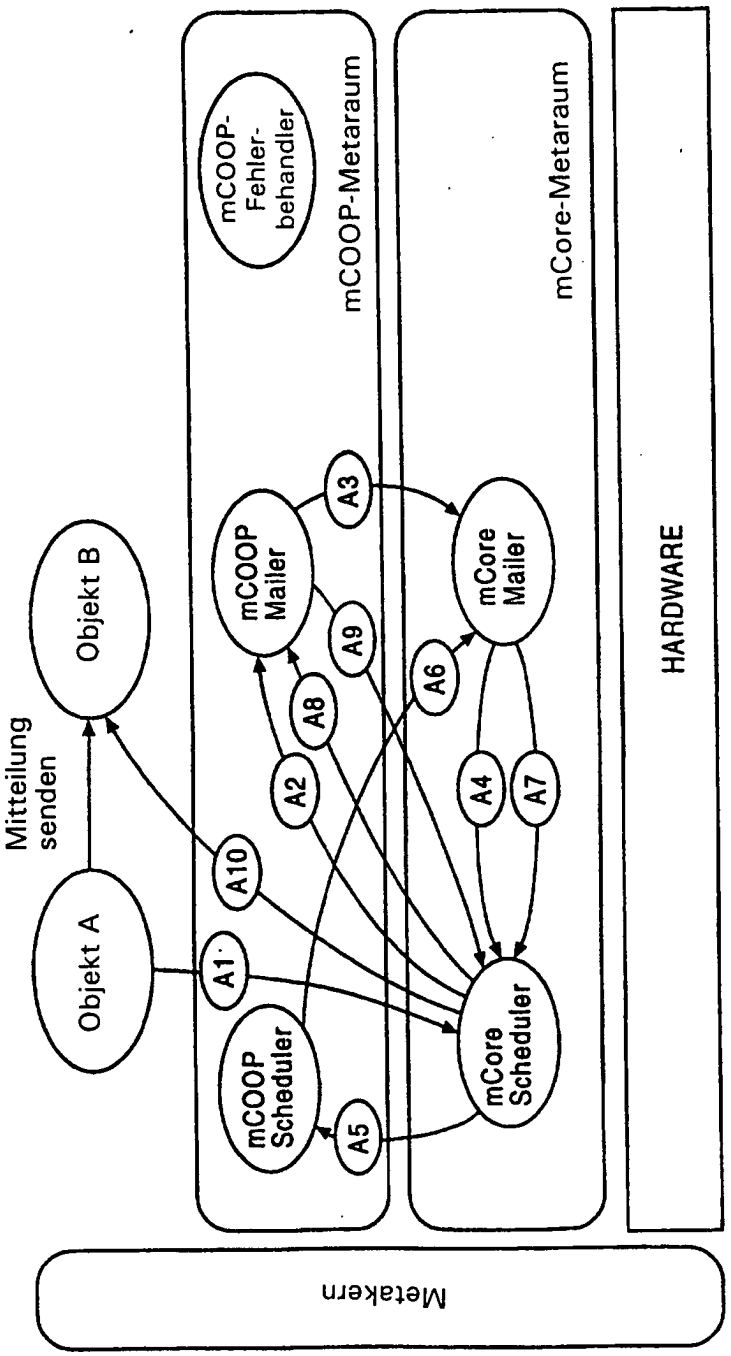

Metakern
mCOOP-Fehler-behandler
mCOOP-Metaraum
mCore-Metaraum
HARDWARE
Objekt B
Objekt A
Mitteilung senden
mCOOP Mailer
mCore Mailer
mCOOP Scheduler
mCore Scheduler
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10

FIG. 4

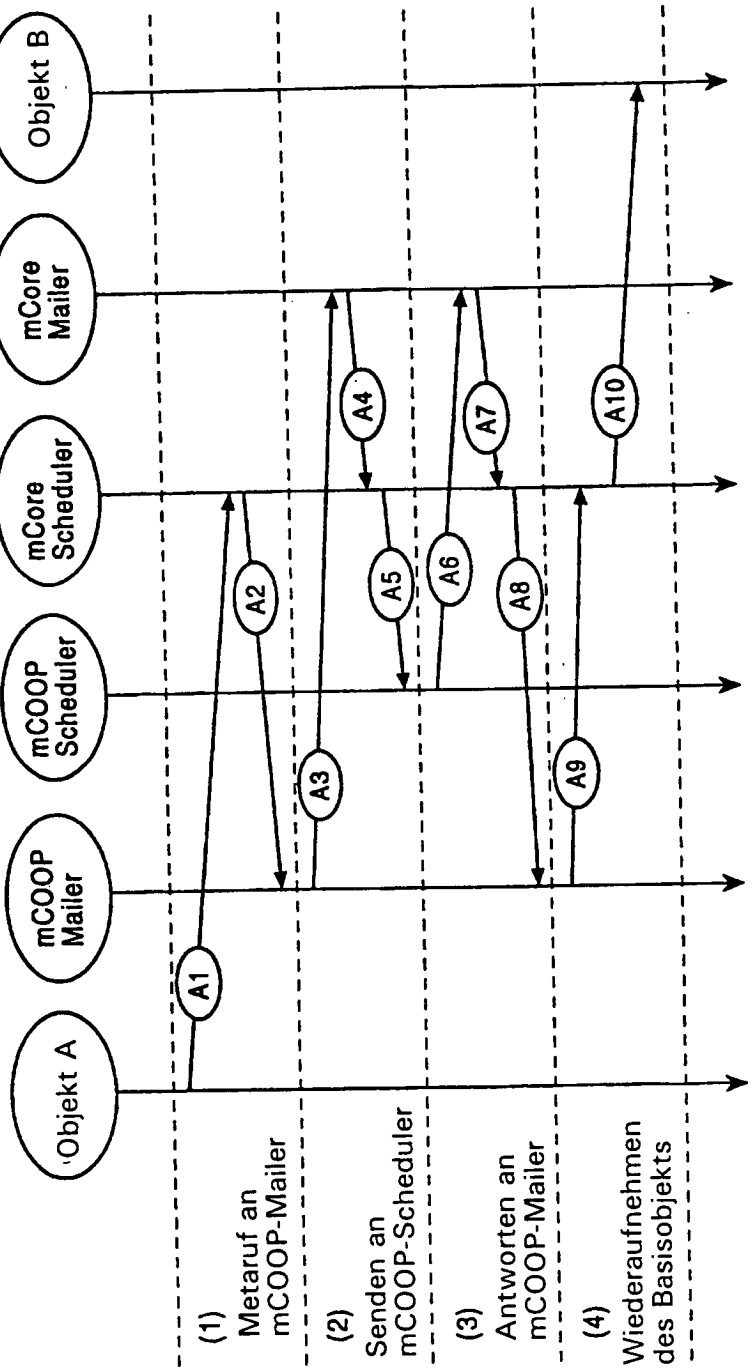

Objekt A
mCOOP Mailer
mCOOP Scheduler
mCore Scheduler
mCore Mailer
Objekt B
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
(1) Metaruf an mCOOP-Mailer
(2) Senden an mCOOP-Scheduler
(3) Antworten an mCOOP-Mailer
(4) Wiederaufnehmen des Basisobjekts

FIG. 5

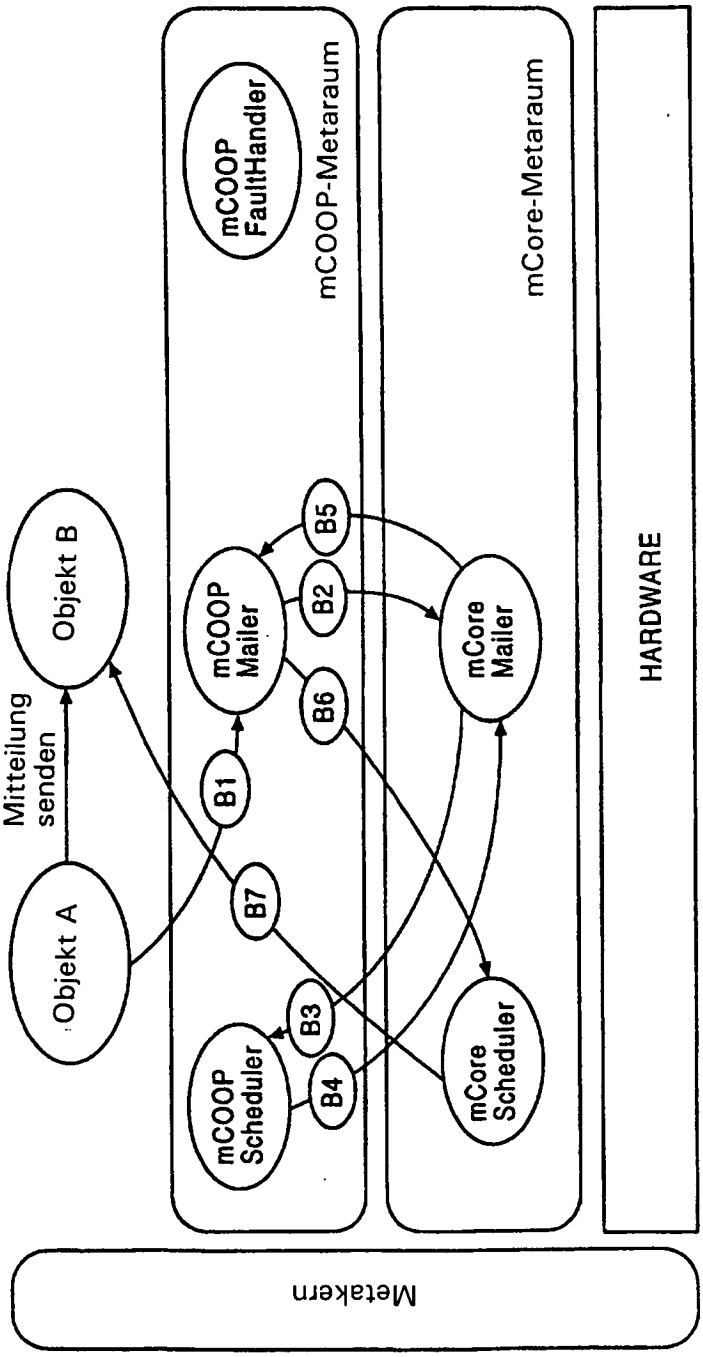

Metakern
mCOOP-Metaraum
mCore-Metaraum
HARDWARE
mCOOP FaultHandler
mCOOP Mailer
mCOOP Scheduler
mCore Mailer
mCore Scheduler
Objekt A
Objekt B
Mitteilung senden
B1
B2
B3
B4
B5
B6
B7

FIG. 6

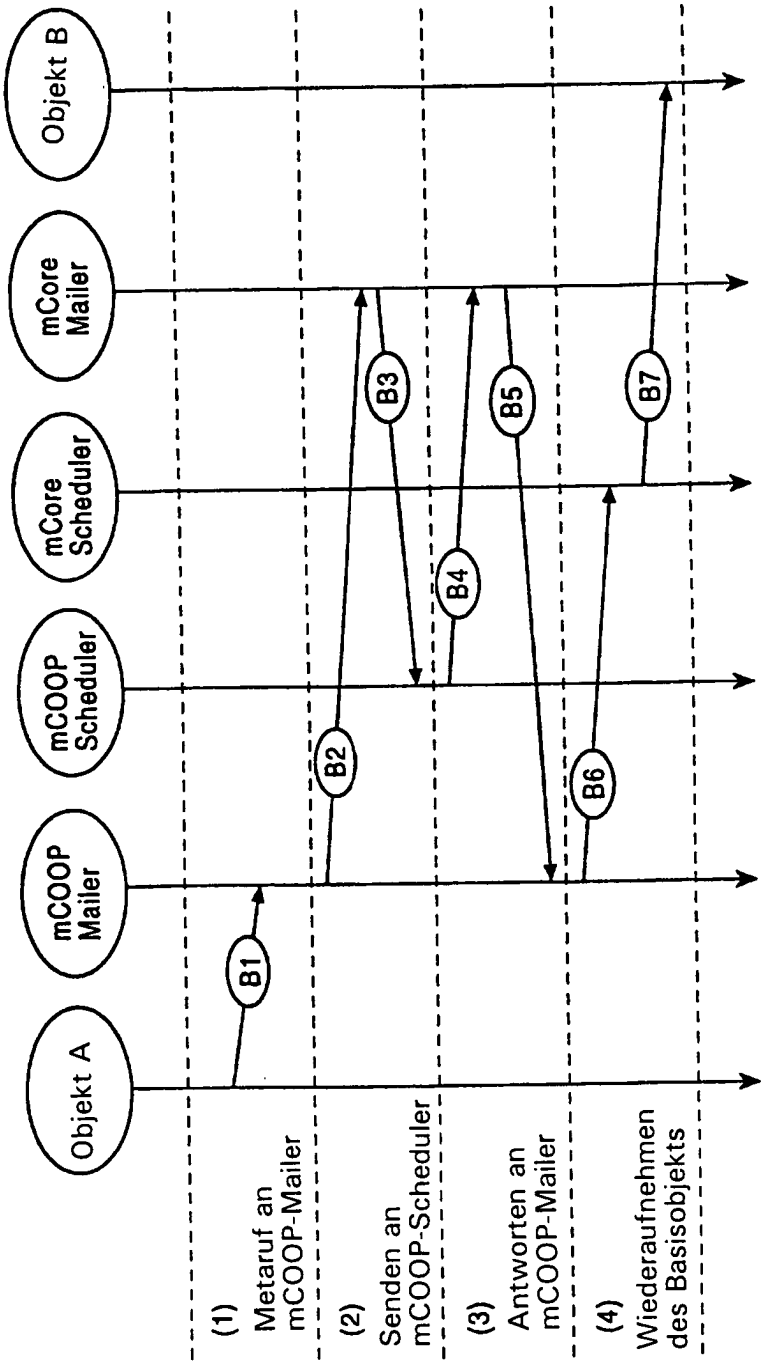

Objekt A
mCOOP Mailer
mCOOP Scheduler
mCore Scheduler
mCore Mailer
Objekt B
B1
B2
B3
B4
B5
B6
B7
(1) Metaruf an mCOOP-Mailer
(2) Senden an mCOOP-Scheduler
(3) Antworten an mCOOP-Mailer
(4) Wiederaufnehmen des Basisobjekts

FIG. 7

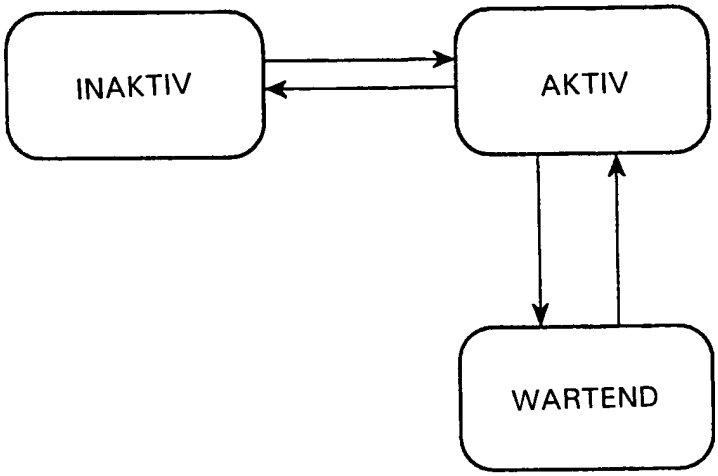


FIG. 8

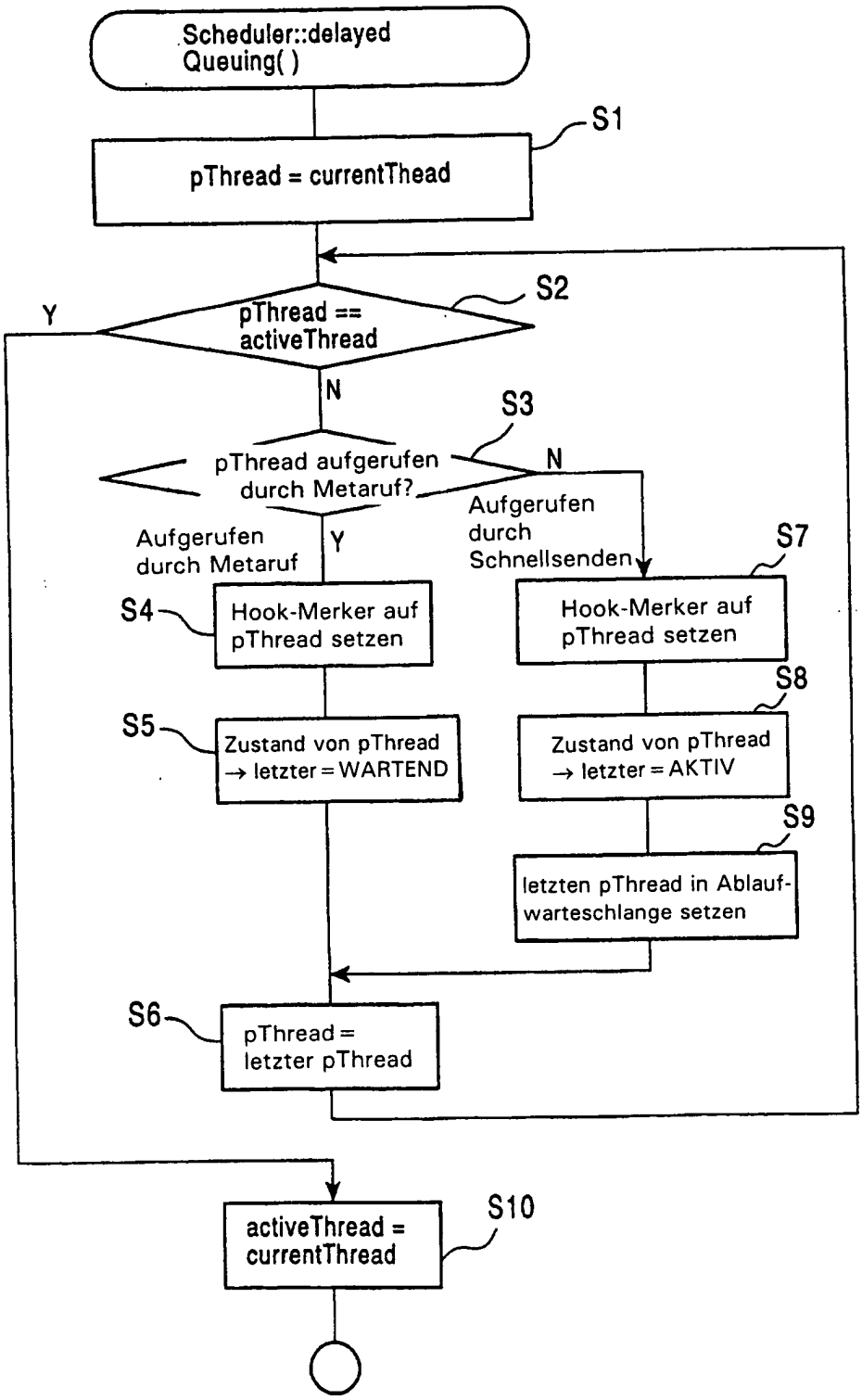

Scheduler::delayed Queuing( )
S1
pThread = currentThead
S2
pThread == activeThread
Y
N
S3
pThread aufgerufen durch Metaruf?
N
Y
Aufgerufen durch Metaruf
Aufgerufen durch Schnellsenden
S4
Hook-Merker auf pThread setzen
S5
Zustand von pThread → letzter = WARTEND
S7
Hook-Merker auf pThread setzen
S8
Zustand von pThread → letzter = AKTIV
S9
letzten pThread in Ablauf-warteschlange setzen
S6
pThread = letzter pThread
S10
activeThread = currentThread

FIG. 9

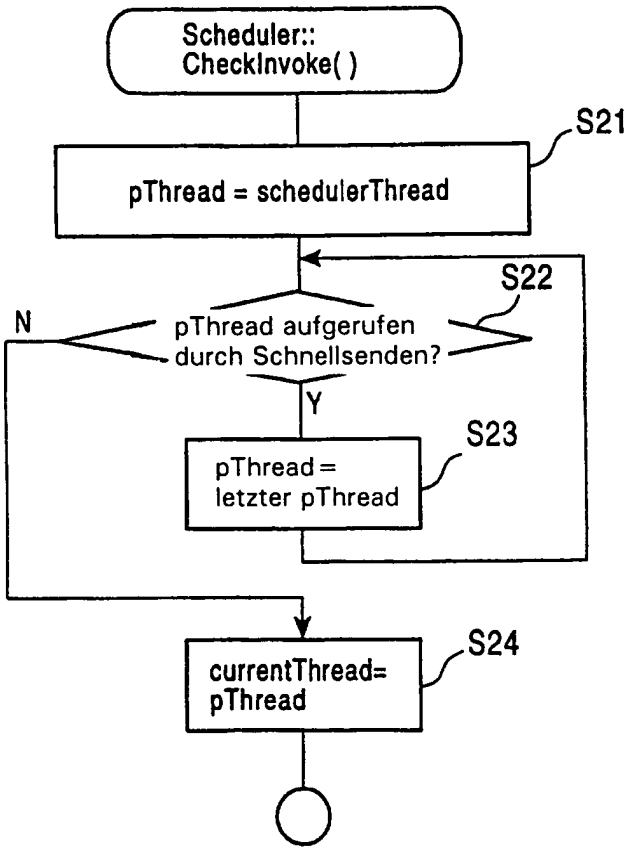

Scheduler::
CheckInvoke( )
S21
pThread = schedulerThread
S22
N
pThread aufgerufen
durch Schnellsenden?
Y
S23
pThread =
letzter pThread
S24
currentThread=
pThread

FIG. 10

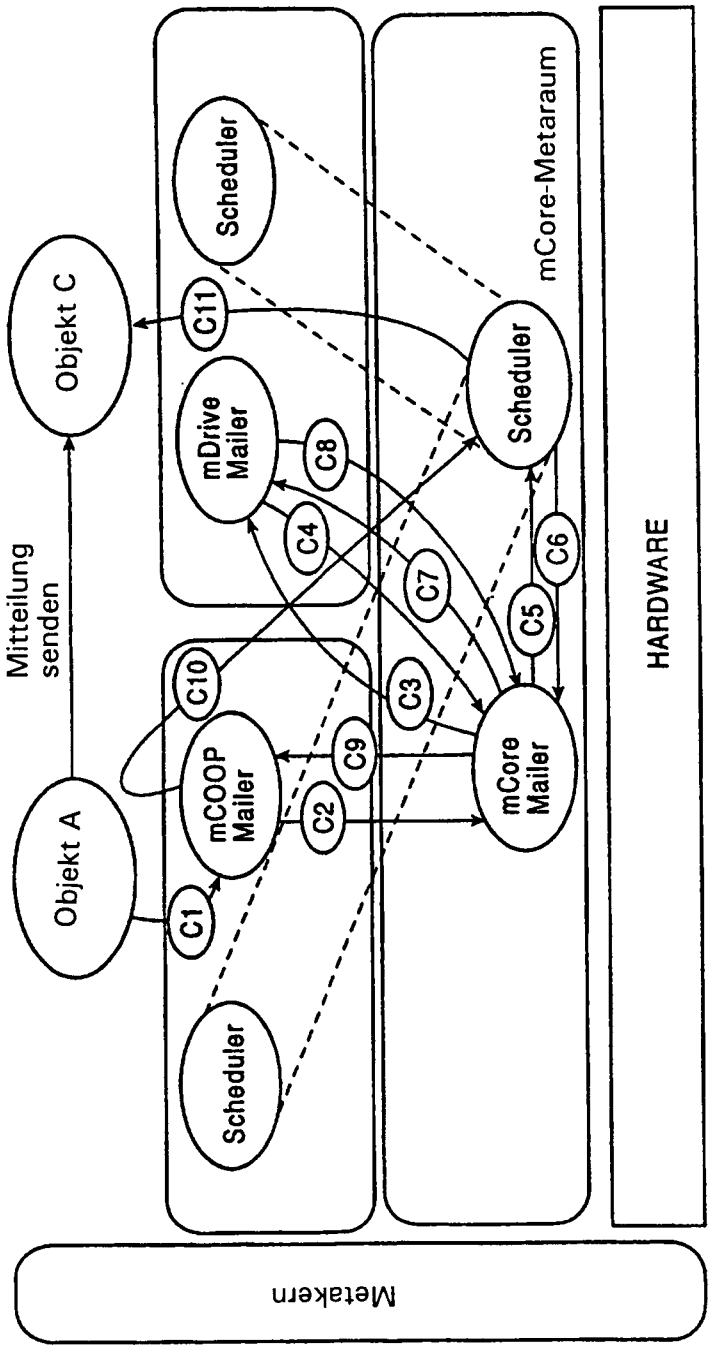

Metakern
HARDWARE
mCore-Metaraum
Scheduler
Objekt C
C11
mDrive Mailer
C8
C4
C7
C6
C5
Scheduler
Mitteilung senden
C10
C3
C9
mCOOP Mailer
C2
mCore Mailer
Objekt A
C1
Scheduler

FIG. 11

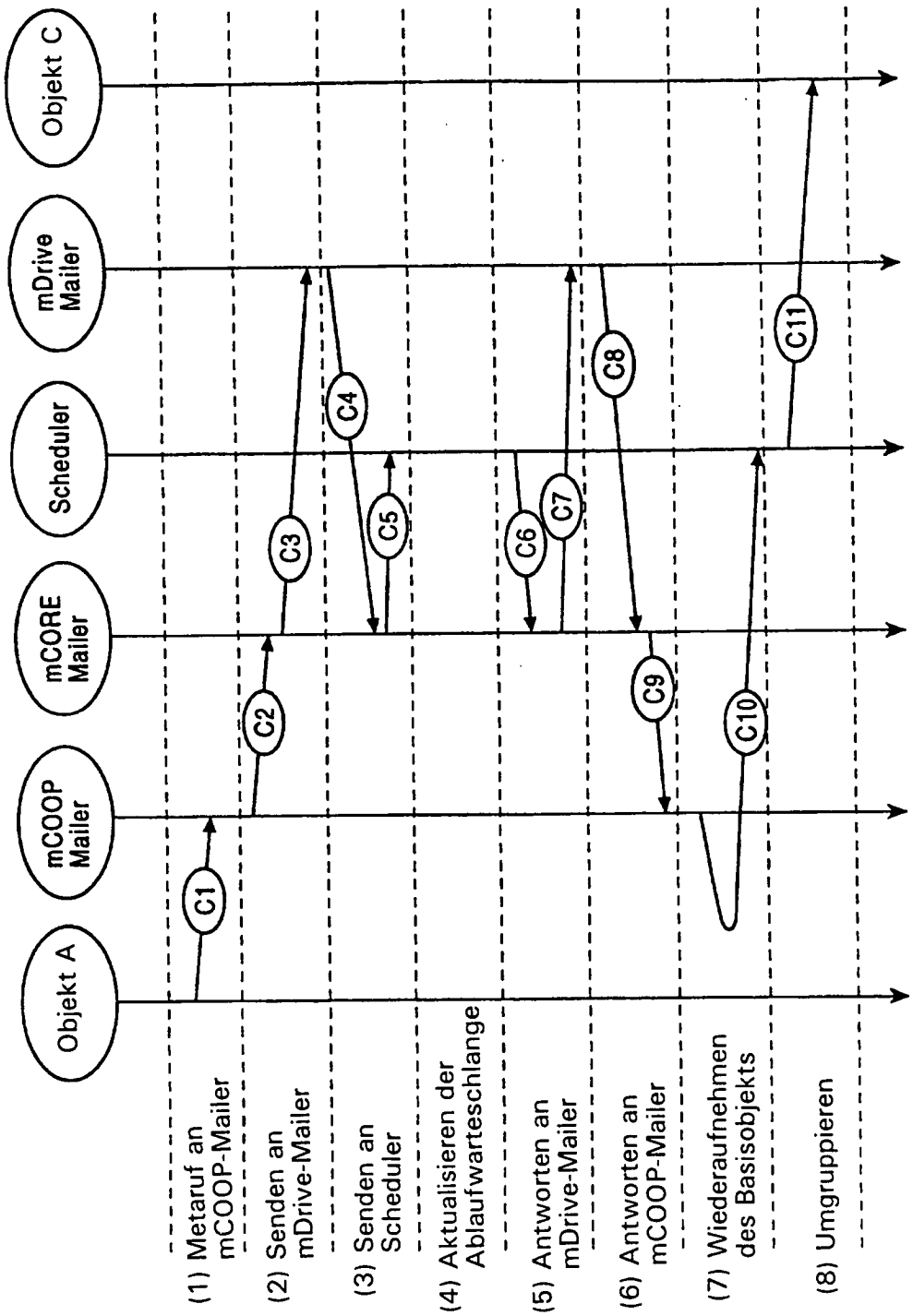

Objekt A
mCOOP Mailer
mCORE Mailer
Scheduler
mDrive Mailer
Objekt C
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
(1) Metaruf an mCOOP-Mailer
(2) Senden an mDrive-Mailer
(3) Senden an Scheduler
(4) Aktualisieren der Ablaufwarteschlange
(5) Antworten an mDrive-Mailer
(6) Antworten an mCOOP-Mailer
(7) Wiederaufnehmen des Basisobjekts
(8) Umgruppieren

FIG. 12

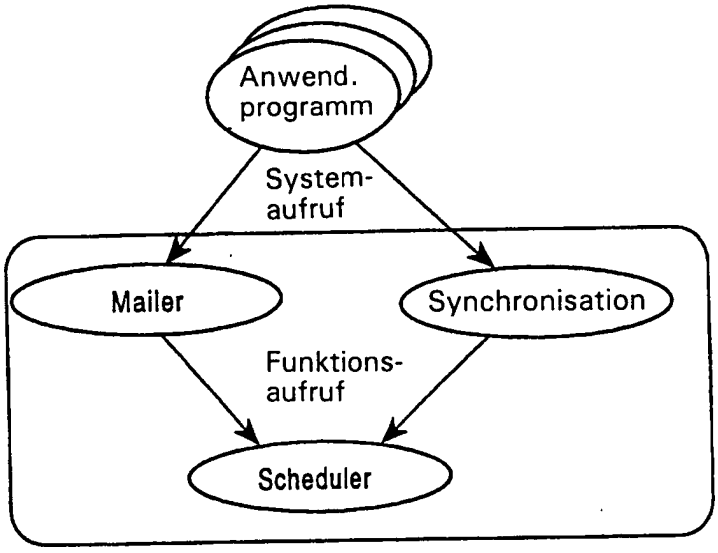

Anwend.
programm
System-
aufruf
Mailer
Synchronisation
Funktions-
aufruf
Scheduler

FIG. 13

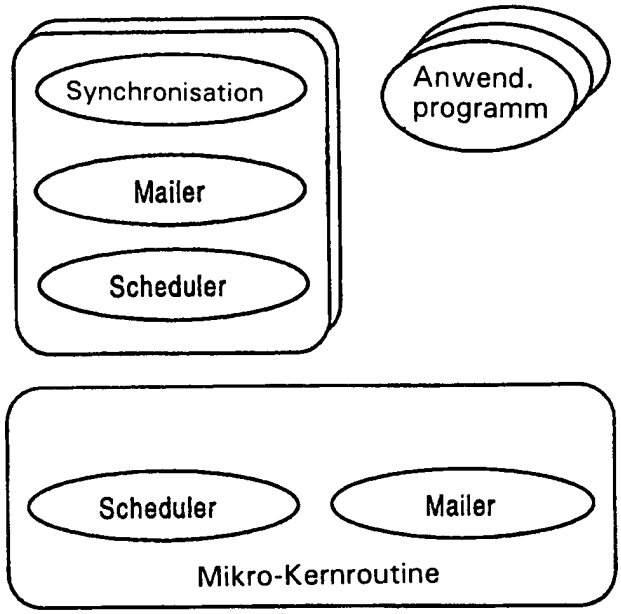
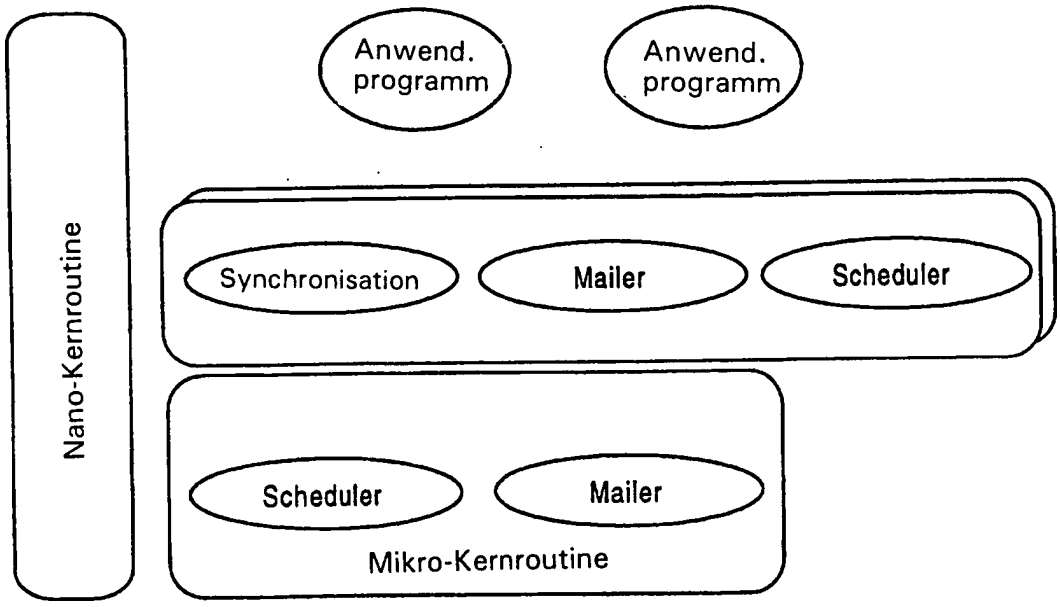

Synchronisation
Mailer
Scheduler
Anwend. programm
Scheduler
Mailer
Mikro-Kernroutine

FIG. 14


Anwend. programm
Anwend. programm
Nano-Kernroutine
Synchronisation
Mailer
Scheduler
Scheduler
Mailer
Mikro-Kernroutine