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

(10) **DE 696 35 094 T2** 2006.04.20

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 260 891 B1**
(21) Deutsches Aktenzeichen: **696 35 094.7**
(96) Europäisches Aktenzeichen: **02 076 462.7**
(96) Europäischer Anmeldetag: **30.05.1996**
(97) Erstveröffentlichung durch das EPA: **27.11.2002**
(97) Veröffentlichungstag
der Patenterteilung beim EPA: **17.08.2005**
(47) Veröffentlichungstag im Patentblatt: **20.04.2006**

(51) Int Cl.[8]: ***G05B 19/416*** (2006.01)

(30) Unionspriorität:
454736 **30.05.1995** **US**

(84) Benannte Vertragsstaaten:
AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LI, LU, MC, NL, PT, SE

(73) Patentinhaber:
Roy-G-Biv Corp., Bingen, Wash., US

(74) Vertreter:
Weickmann & Weickmann, 81679 München

(72) Erfinder:
Brown, David W., Bingen, US; Clark, Jay S., Bingen, Washington 98605, US

(54) Bezeichnung: **Systeme zur Bewegungssteuerung**

Beschreibung

Technisches Gebiet

[0001] Die vorliegende Erfindung betrifft Bewegungssteuerungssysteme und insbesondere Schnittstellensoftware, die das Erstellen von hardware-unabhängiger Bewegungssteuerungs-Software ermöglicht.

Hintergrund der Erfindung

[0002] Der Zweck einer Bewegungssteuerungsvorrichtung ist die Bewegung eines Objekts in einer gewünschten Weise. Die Basiskomponenten einer Bewegungssteuerungsvorrichtung sind eine Steuereinheit und ein mechanisches System. Das mechanische System übersetzt Signale, die von der Steuereinheit erzeugt werden, in die Bewegung eines Objekts.

[0003] Während das mechanische System üblicherweise einen Antrieb und einen Elektromotor umfasst, kann eine Anzahl anderer Systeme wie ein hydraulisches System oder ein Vibrationssystem verwendet werden, um die Bewegung eines Objekts auf Grundlage eines Steuersignals zu bewirken. Außerdem kann eine Bewegungssteuerungsvorrichtung eine Mehrzahl von Antrieben und Motoren aufweisen, damit eine Mehrachsensteuerung der Bewegung des Objekts möglich ist.

[0004] Die vorliegende Erfindung ist von besonderer Bedeutung im Zusammenhang mit einem mechanischen System mit zumindest einem Antrieb und einem Elektromotor, dessen Rotationswelle auf eine Weise mit dem zu bewegenden Objekt verbunden ist, und diese Anwendung wird hier im Detail beschrieben. Doch sind die Grundgedanken der vorliegenden Erfindung allgemein auf jedes mechanische System anwendbar, das eine Bewegung auf Grundlage eines Steuersignals erzeugt. Der Rahmen der Erfindung sollte somit auf der Basis der anliegenden Ansprüche und nicht auf der Basis der folgenden Beschreibung bestimmt werden.

[0005] Bei einem mechanischen System, das eine Steuereinheit, einen Antrieb und einen Elektromotor aufweist, ist der Motor physisch mit dem zu bewegenden Objekt verbunden, so dass die Drehung der Motorwelle in eine Bewegung des Objekts übersetzt wird. Der Antrieb ist ein elektronischer Leistungsverstärker, der für die Versorgung eines Motors mit Energie ausgelegt ist, so dass sich die Motorwelle in kontrollierter/gesteuerter Weise dreht. Auf der Grundlage von Steuerbefehlen steuert die Steuereinheit den Antrieb auf vorhersagbare Weise, so dass das Objekt in der gewünschten Weise bewegt wird.

[0006] Diese Basiskomponenten werden normalerweise in einem größeren System angeordnet, damit eine spezielle Aufgabe erfüllt werden kann. Zum Beispiel kann eine Steuereinheit in Verbindung mit mehreren Antrieben und Motoren in einem Mehrachsensystem zur Bewegung eines Werkzeugs entlang einer vorgegebenen Bahn relativ zu dem Werkstück arbeiten.

[0007] Außerdem werden die vorstehend beschriebenen Basiskomponenten häufig in Verbindung mit einem Hauptrechner oder einer programmierbaren logischen Steuerung (PLC) verwendet. Der Hauptrechner bzw. die PLC erlauben die Anwendung einer höheren Programmiersprache, um Steuerbefehle zu generieren, die an die Steuereinheit übermittelt werden. Deshalb ist Software, die auf dem Hauptrechner läuft, so konzipiert, dass die Aufgabe des Programmierens der Steuerung vereinfacht wird.

[0008] Hersteller von Bewegungssteuerungsvorrichtungen sind traditionell hardware-orientierte Hersteller, die dedizierte Software für die von ihnen hergestellte Hardware produzieren. Diese Software-Produkte kann man als niedere Programme bezeichnen. Niedere Programme arbeiten gewöhnlich direkt mit einer für eine gegebene Bewegungssteuerungsvorrichtung spezifischen Befehlssprache für die Bewegungssteuerung. Solche niederen Programme bieten dem Programmierer zwar eine im wesentlichen vollständige Kontrolle über die Hardware, doch sind diese Programme in hohem Maße hardwareabhängig.

[0009] Im Gegensatz zu niederen Programmen ermöglichen höhere Softwareprogramme, die manchmal als Produktionsautomatisierungsanwendungen bezeichnet werden, einem Produktionssystemplaner die Entwicklung von Anwendungsprogrammen, die eine große Anzahl von Ein-/Ausgabe-Geräten (E/A-Geräten), Bewegungssteuerungsvorrichtungen eingeschlossen, zu einem komplexen System kombinieren, das zur Systemautomatisierung verwendet wird. Diese Systemautomatisierungsanwendungen erlauben die Verwendung einer beliebigen Anzahl von E/A-Geräten in einem gegebenen System, solange diese Geräte von dem höheren Programm unterstützt werden. Von anderen Softwareentwicklern entwickelte kundenspezifische Anwendungen erlauben keine Entwicklung dahingehend, dass sie die Funktionalität der einfachen Bewegungssteuerung

nutzen, die das Systemautomatisierungsprogramm bietet.

[0010] Hinzu kommt, dass diese Programme dem Programmierer keine weitreichende Kontrolle über die jeweilige Bewegungssteuerungsvorrichtung in dem System einräumen. Jedes Programm, das mit einer Systemautomatisierungsanwendung entwickelt wird, muss innerhalb des Kontexts dieser Anwendung laufen.

Stand der Technik

[0011] In der folgenden Diskussion wird eine Anzahl von Dokumenten zitiert, die ab dem Datum der Hinterlegung vorliegender Anmeldung öffentlich zugänglich waren. Bei vielen dieser Dokumente kennen die Anmelder das genaue Veröffentlichungsdatum nicht, weshalb diese Dokumente nicht als Stand der Technik gewertet werden sollten. Sofern notwendig, werden die Anmelder bemüht sein nachzuweisen, ob diese Dokumente dem Stand der Technik angehören.

[0012] Wie vorstehend bereits erwähnt, gibt es eine Anzahl von Softwareprogrammen für die Programmierung individueller Bewegungssteuerungsvorrichtungen und für die Unterstützung der Entwicklung von Systemen, die eine Anzahl von Bewegungssteuerungsvorrichtungen umfassen.

[0013] Die folgende Auflistung nennt Dokumente, die derzeit kommerziell verfügbare höhere Softwareprogramme beschreiben: (a) Software Products For Industrial Application; iconics 1993; (b) The complete, computer-based automation tool (IGSS), Seven Technologies A/S; (c) OpenBatch Product Brief, PID, Inc.; (d) FIX Product Brochure, Intellution (1994); (e) Paragon TNT Product Brochure, Intec Controls Corp.; (f) WEB 3.0 Product Brochure, Trihedral Engineering Ltd. (1994); und (g) AIMAX-WIN Product Brochure, TA Engineering Co., Inc. Die folgenden Dokumente beschreiben Simulationssoftware: (a) ExperTune PID Tuning Software, Gerry Engineering Software; und (b) XANALOG Model NL-SIM Product Brochure, XANALOG.

[0014] Die folgende Liste nennt Dokumente, die sich auf niedere Programme beziehen: (a) Compumotor Digiplan 1993–94 catalog, Seiten 10–11; (b) Aerotech Motion Control Product Guide, Seiten 233–34; (c) PMAC Product Catalog, Seite 43; (d) PC/DSP-Series Motion Controller C Programming Guide, Seiten 1–3; (e) Oregon Micro Systems Product Guide, Seite 17; (f) Precision Microcontrol Product Guide.

[0015] Die Anmelder haben auch Kenntnis von einem Software-Modell, das als WOSA bezeichnet wird und von Microsoft für die Verwendung in der Windows-Programmierumgebung definiert wurde. Das WOSA-Modell wird in dem Buch Inside Windows 95 auf den Seiten 348–351 erläutert. WOSA wird auch in dem Papier WOSA Backgrounder: Delivering Enterprise Services to the Windows-based Desktop besprochen. Das WOSA-Modell trennt Anwendungsprogrammierer von den Schwierigkeiten der Programmierung für verschiedene Dienstanbieter, indem eine API-Schicht (Anwendungsprogrammierschnittstelle) zur Verfügung gestellt wird, die von einer zugrundeliegenden Hardware oder einem Dienst unabhängig ist, und eine SPI-Schicht (Dienstanbieter-Schnittstelle), die hardware-unabhängig aber dienstabhängig ist. Das WOSA-Modell steht in keiner Beziehung zu Bewegungssteuerungsvorrichtungen.

[0016] Die Anmelder kennen auch die übliche Programmierpraxis, wonach Treiber für Hardware wie Drucker etc. vorgesehen werden; ein Anwendungsprogramm wie beispielsweise ein Textverarbeitungssystem ermöglicht einem Benutzer die Wahl eines Treibers, der einem gegebenen Drucker zugeordnet ist, damit das Anwendungsprogramm auf diesem gegebenen Drucker drucken kann.

[0017] Während dieser Ansatz den Anwendungsprogrammierer in der Tat mit den Problemen der Programmierung für jede existierende Hardeware-Konfiguration verschont, versetzt dieser Ansatz den Programmierer nicht in die Lage, die Hardware in inkrementalen Basisschritten zu steuern. Nimmt man als Beispiel den Drucker, so wird ein Anwendungsprogrammierer nicht in der Lage sein, jeden Schrittmotor im Drucker unter Verwendung des gelieferten Druckertreibers zu steuern; stattdessen wird der Druckertreiber eine Anzahl von Schrittmotoren in dem Drucker in einer vorbestimmten, für die Implementierung einer Gruppe höherer Befehle notwendigen Sequenz steuern.

[0018] Das derzeit für Drucker und dergleichen verwendete Softwaretreiber-Modell eignet sich damit nicht für die Anwendung bei der Entwicklung einer Sequenz von Steuerbefehlen für Bewegungssteuerungsvorrichtungen.

[0019] Das Dokument US5453933 beschreibt ein Steuersystem für Werkzeugmaschinen, das mit Software-Objekten arbeitet, die durch ein objektorientiertes Messaging-System (Nachrichtenübermittlungssystem)

mit Maschinen-Objekten kommunizieren, die physische Geräte an Maschinen darstellen. Das Dokument US5491813 beschreibt eine Systemprogrammierschnittstelle für Computergrafik, die eine Aufrüstung und Beschleunigung von mehreren physischen Bildschirmanschlüssen ermöglicht, ohne entsprechendes Aufrüsten oder Umschreiben von bildschirmspezifischen Gerätetreibercodes für jede separate Programmanwendung, die das Grafiksystem benutzt.

Aufgaben der Erfindung

[0020] Aus dem Vorgenannten ist zu erkennen, dass es eine primäre Aufgabe der Erfindung ist, verbesserte Vorrichtungen für die Bewegung von Objekten zur Verfügung zu stellen.

[0021] Eine weitere speziellere Aufgabe der vorliegenden Erfindung ist es, eine Vorrichtung für die Planung und den Einsatz von Bewegungssteuerungsvorrichtungen zu erhalten, bei denen diese Verfahren und diese Vorrichtungen eine vorteilhafte Mischung der folgenden Merkmale zeigen:

(a) die Ermöglichung der Erstellung höherer Bewegungssteuerungsprogramme, die hardware-unabhängig sind, aber die Programmierbarkeit von Basis-Bewegungsoperationen bieten;

(b) das Verbergen der Schwierigkeiten des Programmierens für multiple Hardwarekonfigurationen vor dem höheren Programmierer;

(c) das problemlose Erweitern auf die Unterstützung zusätzlicher Hardware-Konfigurationen; und

(d) das transparente Unterstützen höherer Programmierungsumgebungen auf Industriestandard.

Zusammenfassung der Erfindung

[0022] Die Erfindung sorgt für ein System zum Erzeugen von Steuerbefehlen zum Steuern einer Bewegungssteuerungsvorrichtung wie in Anspruch 1 angegeben. Ausführungsformen der Erfindung werden verwendet zum Implementieren eines die Schritte des Entwickelns eines höheren Bewegungssteuerungs-Anwendungsprogramms enthaltenden Verfahrens zur Bewegung eines Objekts, umfassend eine Sequenz von Komponentenfunktionen, die eine gewünschte Objektbahn beschreiben, Korrelieren dieser Komponentenfunktionen mit Treiberfunktionen, Wählen eines Softwaretreibers für die gesteuerte spezielle Hardwarekonfiguration, Generieren von Steuerbefehlen aus den Treiberfunktionen und dem zu der gesteuerten Hardwarekonfiguration gehörenden Softwaretreiber und Steuern einer Bewegungssteuerungsvorrichtung auf Grundlage der Steuerdaten, um das Objekt entlang der gewünschten Objektbahn zu bewegen.

[0023] Ausführungsformen der Erfindung werden ebenfalls verwendet zum Implementieren eines Verfahrens zur Erzeugung einer Sequenz von Steuerbefehlen zum Steuern einer Bewegungssteuerungsvorrichtung für die Bewegung eines Objekts entlang einer gewünschten Bahn bereitgestellt. Ein Anwendungsprogramm, das eine Reihe von Komponentenfunktionen umfasst, definiert eine Sequenz von Bewegungsschritten, die von der Bewegungssteuerungsvorrichtung ausgeführt werden müssen, um das Objekt entlang der gewünschten Bahn zu bewegen. Die Komponentenfunktionen enthalten einen Code, der die Komponentenfunktionen mit den Treiberfunktionen in Beziehung setzt. Die Treiberfunktionen sind Softwaretreibern zugeordnet oder enthalten Softwaretreiber, die einen Treibercode für die Implementierung der Bewegungsschritte an einer gegebenen Bewegungssteuerungsvorrichtung aufweisen. Die Steuerbefehle werden auf der Grundlage des Anwendungsprogramms und eines einer gegebenen Bewegungssteuerungsvorrichtung zugeordneten Treibercodes erzeugt.

[0024] Die Verwendung von Komponentenfunktionen, die von den Treiberfunktionen getrennt sind, trennt den Programmierer von den Schwierigkeiten der Programmierung für eine spezielle Bewegungssteuerungsvorrichtung. Diese Anordnung ermöglicht auch die Benutzung eines gegebenen Anwendungsprogramms ohne Modifikation für jede Bewegungssteuerungsvorrichtung, der ein Softwaretreiber zugeordnet ist.

[0025] Die Treiberfunktionen können in Kerntreiberfunktionen und erweiterte Treiberfunktionen gruppiert werden. Alle Softwaretreiber müssen die Kerntreiberfunktionen unterstützen; die Softwaretreiber können auch eine oder mehrere der erweiterten Treiberfunktionen unterstützen, wenngleich dies nicht notwendig ist.

[0026] Wo die Softwaretreiber die erweiterten Treiberfunktionen nicht unterstützen, kann die mit den erweiterten Treiberfunktionen verbundene Funktionalität normalerweise simuliert werden, indem man eine Kombination von Kerntreiberfunktionen verwendet. In diesem Fall kann das Ermitteln, welche der erweiterten Treiberfunktionen nicht durch Softwaretreiber unterstützt werden, und, wo dies möglich ist, das Substituieren einer Kombination von Kerntreiberfunktionen durch Ausführungsformen der vorliegenden Erfindung implementiert werden. In manchen Fällen kann die Funktionalität einer erweiterten Treiberfunktion nicht durch die Verwendung von Kerntreiberfunktionen emuliert werden, und diese Funktionalität ist für den Programmierer einfach

nicht verfügbar.

[0027] Die Verwendung von Kerntreiberfunktionen zum Emulieren von erweiterten Treiberfunktionen schafft Funktionalität dort, wo sie ansonsten nicht vorhanden wäre, doch der bevorzugte Ansatz ist die Bereitstellung eines Softwaretreibers, der jede der erweiterten Treiberfunktionen unterstützt. Wenn eine erweiterte Treiberfunktion unterstützt und nicht emuliert wird, wird die zu erfüllende Aufgabe normalerweise schneller und mit größerer Genauigkeit erfüllt.

[0028] Zur Vereinfachung der Anwendung von emulierten erweiterten Treiberfunktionen kann das System außerdem die Schritte des Ermittelns, ob und gegebenenfalls welche erweiterten Treiberfunktionen durch den Softwaretreiber für eine gegebene Hardwarekonfiguration nicht unterstützt werden, den Schritt des Entwickelns einer Funktionszeigertabelle sowohl der nichtunterstützten erweiterten Treiberfunktionen als auch der unterstützten erweiterten Treiberfunktionen und den Schritt des Konsultierens der Tabelle bei jedem Aufruf einer erweiterten Treiberfunktion, um festzustellen, ob diese erweiterte Treiberfunktion emuliert werden muss, implementieren. Auf diese Weise wird der Prozess des Aufrufens der Sequenz von Kerntreiberfunktionen, die zur Emulation der nichtunterstützten erweiterten Treiberfunktionen verwendet werden, optimiert.

[0029] Wenn die Steuerbefehle wie oben beschrieben generiert werden, können sie zur Steuerung einer Bewegungssteuerungsvorrichtung in Echtzeit angewendet oder aber für eine spätere Verwendung in einer Datei gespeichert werden. Vorzugsweise können Ausführungsformen der vorliegenden Erfindung ferner einer Anzahl von Streams (Datenströmen), die einen Stream-Code (Datenstromcode) enthalten, vorsehen. Jeder Stream, der einem Bestimmungsort von Steuerbefehlen zugeordnet ist, und der Stream-Code eines gegebenen Stream bestimmen, wie die Steuerbefehle zu dem Bestimmungsort zu transferieren sind, der diesem Stream zugeordnet ist. Der Benutzer erhält dadurch die Möglichkeit, einen oder mehrere Streams zu wählen, die den Bestimmungsort der Steuerbefehle diktieren.

[0030] Als Hilfe bei der Isolierung des Programmierers von hardware-spezifischen Schwierigkeiten können Ausführungsformen auch die zusätzlichen administrativen Schritte wie die Wahl eines einer besonderen Bewegungssteuerungsvorrichtung zugeordneten Treibers und/oder die Wahl von Übersetzungseinheiten durchführen, die benötigt werden zum Definieren des Bewegungssteuerungssystems als das besondere System von Einheiten, die von einer gegebenen Bewegungssteuerungsvorrichtung verwendet werden.

Figurenkurzbeschreibung

[0031] **Fig.** 1 ist ein System-Interaktions-Abbild eines Bewegungssteuerungssystems, das gemäß den Grundgedanken der vorliegenden Erfindung konstruiert ist und diese verkörpert;

[0032] Fig. 2 ist ein Modul-Interaktions-Abbild einer Bewegungssteuerungs-Komponente des in **Fig.** 1 dargestellten Systems;

[0033] Fig. 3 ist ein Objekt-Interaktions-Abbild der in Fig. 2 gezeigten Komponente;

[0034] Fig. 4 bis Fig. 8 sind Abbilder von Szenarien der in Fig. 2 gezeigten Komponente;

[0035] Fig. 9 ist ein Abbild der Schnittstelle der in Fig. 2 gezeigten Komponente;

[0036] Fig. 10 ist ein Datenabbild, das eine beispielhafte Methode für den Zugriff auf die Daten darstellt, die für die Emulation erweiterter Treiberfunktionen unter Verwendung von Kerntreiberfunktion notwendig sind;

[0037] Fig. 11 ist ein Modul-Interaktions-Abbild des Treiberabschnitts des in **Fig.** 1 gezeigten Systems;

[0038] Fig. 12 ist ein Objekt-Interaktions-Abbild des in Fig. 11 dargestellten Treiberabschnitts;

[0039] Fig. 13 bis Fig. 20 sind Abbilder von Szenarien betreffend den in Fig. 11 gezeigten Treiber;

[0040] Fig. 21 ist ein Abbild der Schnittstelle für den in Fig. 11 dargestellten Treiber,

[0041] Fig. 22 ist ein Modul-Interaktions-Abbild der von dem in **Fig.** 1 dargestellten System benutzten Streams;

[0042] Fig. 23 ist ein Objekt-Interaktions-Abbild der in Fig. 22 gezeigten Streams;

[0043] Fig. 24 bis Fig. 32 sind Abbilder von Szenarien der in Fig. 22 gezeigten Streams;

[0044] Fig. 33 ist ein Schnittstellen-Abbild der Objekte, die die in Fig. 22 gezeigten Streams umfassen;

[0045] Fig. 34 ist ein Modul-Interaktions-Abbild des Abschnitts der Treiber-Stubroutine des in **Fig.** 1 gezeigten Systems;

[0046] Fig. 35 ist ein Objekt-Interaktions-Abbild der in Fig. 34 gezeigten Treiber-Stubroutine;

[0047] Fig. 36 bis Fig. 38 sind Abbilder von Szenarien der in Fig. 34 gezeigten Treiber-Stubroutine;

[0048] Fig. 39 ist ein Schnittstellen-Abbild des in Fig. 34 gezeigten Abschnitts der Treiber-Stubroutine;

[0049] Fig. 40 ist ein Modul-Interaktions-Abbild des Treiber-Administratorbereichs des in **Fig.** 1 gezeigten Systems;

[0050] Fig. 41 ist ein Objekt-Interaktions-Abbild des in Fig. 40 gezeigten Treiber-Administrators;

[0051] Fig. 42 bis Fig. 49 sind Abbilder von Szenarien, die sich auf den in Fig. 40 dargestellten Treiber-Administrator beziehen;

[0052] Fig. 50 ist ein Schnittstellen-Abbild der Objekte, die den in Fig. 40 gezeigten Treiber-Administrator aufweisen;

[0053] Fig. 51 ist ein Modul-Interaktions-Abbild des Abschnitts des in **Fig.** 1 gezeigten Systems, der das CPL-Applet des Treiber-Administrators betrifft;

[0054] Fig. 52 ist ein Objekt-Interaktions-Abbild des in Fig. 51 gezeigten CPL-Applet des Treiber-Administrators;

[0055] Fig. 53 bis Fig. 57 sind Abbilder von Szenarien betreffend das in Fig. 51 gezeigte CPL-Applet des Treiber-Administrators;

[0056] Fig. 58 ist ein Modul-Interaktions-Abbild, das alle mit dem Treiber interagierenden binären Module zeigt sowie die Art und Weise, wie die Module interagieren;

[0057] Fig. 59 ist ein Objekt-Interaktions-Abbild, das dem Modul-Interaktions-Abbild von Fig. 58 entspricht, das jedoch darüber hinaus die den Sprachtreiber **44** bildenden internen C++-Objekte sowie die Art und Weise darstellt, auf welche die Objekte interagieren;

[0058] Fig. 60 bis Fig. 65 zeigen eine Anzahl von Szenarien-Abbildern, in denen die stattfindenden Interaktionen zwischen den an bestimmten Prozessen beteiligten C++-Objekten dargestellt sind;

[0059] Fig. 66 ist ein Schnittstellen-Abbild, das die von der Sprachtreiber-Komponente **44** aufgedeckten Schnittstellen, alle verwendeten Datenstrukturen und die Definitionen jeder angewendeten C++-Klasse beschreibt; und

[0060] Fig. 67 zeigt ein Tabellenblatt, das darstellt, wie eine von dem Sprachtreiber **44** benutzte typische Datenbank aufgebaut sein kann.

Detailbeschreibung der Erfindung

[0061] Bezugnehmend auf die Zeichnungen ist bei Pos. **10** in **Fig.** 1 ein Bewegungssteuerungssystem dargestellt, das erfindungsgemäß ausgebildet ist und das die Grundgedanken der vorliegenden Erfindung verkörpert. Dieses System **10** hat einen Personalcomputer-Abschnitt **12** mit einem Hardware-Bus **14**, einer Mehrzahl von Hardware-Steuereinheiten **16a**, **16b** und **16c** und mechanischen Systemen **18a**, **18b** und **18c**, die mit einem oder mehreren zu bewegenden Objekten (nicht dargestellt) interagieren.

[0062] Der Personalcomputer-Abschnitt **12** des Systems **10** kann ein beliebiges System sein, das in der hier beschriebenen Weise programmiert werden kann, doch in der bevorzugten Ausführungsform ist es ein System, das in einer Microsoft Windows-Umgebung laufen kann. Ein solches System hat normalerweise zusätzlich zu dem in **Fig.** 1 gezeigten Hardware-Bus **14** einen seriellen Port (Anschlussstelle).

[0063] Der Hardware-Bus **14** sorgt für die physischen Verbindungen, die notwendig sind, damit der Computer **12** mit den Hardware-Steuereinheiten **16** kommunizieren kann. Die Hardware-Steuereinheiten **16** steuern das mechanische System **18**, so dass dieses sich auf vorhersagbare Weise bewegt. Das mechanische System **18** hat einen Motor oder dergleichen, dessen Ausgangswelle mit dem zu bewegenden Objekt verbunden ist. Die Kombination der Hardware-Steuereinheiten **16a**, **16b** und **16c** und der mechanischen Systeme **18a**, **18b** und **18c** bildet jeweils Bewegungssteuerungsvorrichtungen **20a**, **20b** und **20c**.

[0064] Der Hardware-Bus **14**, die Hardware-Steuereinheiten **16** und die mechanischen Systeme **18** sind in der Fachwelt hinlänglich bekannt, weshalb sie nur so weit beschrieben werden, wie es für das Verständnis der Erfindung notwendig ist.

[0065] Der Personalcomputer-Abschnitt **12** enthält ein Softwaresystem **22**, das es einem Anwendungsbenutzer **24** erlaubt, Softwareanwendungen **26** zu erstellen, die die Bewegungssteuerungsvorrichtungen **20** steuern.

[0066] Insbesondere generiert das Softwaresystem **22** auf Grundlage von Daten, die von dem Benutzer **24** eingegeben werden, und auf Grundlage des Inhalts des Anwendungsprogramms **26** Steuerbefehle, die durch einen oder mehrere Streams (Datenströme) wie jene, die bei Pos. **28a**, **28b** und **28d** gezeigt sind, übertragen werden. Die Streams **28** übertragen Steuerbefehle, in denen die hardware-spezifische Sprache enthalten ist, die notwendig ist, um eine gegebene Bewegungssteuerungsvorrichtung derart zu steuern, dass sie in der gewünschten Weise arbeitet. Wie nachfolgend näher erläutert wird, implementieren die Streams **28** das Kommunikationsprotokoll, das es den Steuerbefehlen ermöglicht, über einen geeigneten Kanal (d.h. PC-Bus, serieller Port) die geeignete Bewegungssteuerungsvorrichtung zu erreichen.

[0067] Unter Verwendung des Systems **22** wird das Anwendungsprogramm **26** so entwickelt, dass es keinen Code enthält, der für irgendeine der exemplarischen Hardware-Steuereinheiten **16** spezifisch ist. Im Normalfall ist das Anwendungsprogramm **26** und daher der Benutzer **24**, der das Programm **26** erstellt hat, vollständig von den Bewegungssteuerungsvorrichtungen **20** isoliert. Der Benutzer **24** muss deshalb nichts über die hardware-spezifische Befehlssprache oder das mit jeder dieser Vorrichtungen **20** verbundene Kommunikationsprotokoll wissen; es kann sogar möglich sein, dass die Befehlssprache einer oder mehrerer der Hardware-Steuereinheiten **16** zur Zeit der Erstellung des Anwendungsprogramms **26** nicht definiert war.

[0068] Das Softwaresystem **22** umfasst eine Kombination von Elementen, die eine komplette Trennung des Anwendungsprogramms **26** von den Hardware-Steuereinheiten **16** erlauben. In der folgenden Erläuterung wird das Rahmenwerk des Softwaresystems **22** als eine Methode zur Bewegung eines Objekts und/oder eine Methode zur Generierung von Steuerbefehlen beschrieben. Nach dieser allgemeinen Erläuterung wird jede Komponente des Systems **22** in einer speziellen Arbeitsumgebung im Detail beschrieben.

I. Methode zur Generierung von Steuerbefehlen für die Steuerung einer Bewegungssteuerungsvorrichtung zum Bewegen eines Objekts

[0069] Zunächst sei bemerkt, dass an der in diesem Abschnitt beschriebenen Methode in den meisten Situationen normalerweise, wenn auch nicht unbedingt, wenigstens zwei und vielleicht sogar drei separate Softwareprogrammierer mit ihrer Arbeit beteiligt sind: ein Softwaresystemplaner; ein mit den Verwicklungen der Bewegungssteuerungsvorrichtung vertrauter Hardwareplaner; und ein Planer für Bewegungssteuerungssysteme. Der vorgenannte Anwendungsbenutzer **24** wird in der Regel der Planer für das Bewegungssteuerungssystem sein, wobei sich die jeweilige Rolle des Softwaresystemplaners und des Hardwareplaners aus der nachstehenden Erläuterung ergibt.

[0070] Der Softwaresystemplaner entwickelt das Softwaresystem **22**. Der Softwaresystemplaner definiert zu Beginn eine Gruppe von Bewegungssteuerungsoperationen, die für die Durchführung der Bewegungssteuerung angewendet werden. Die Bewegungssteuerungsoperationen sind nicht speziell auf eine bestimmte Hardware-Konfiguration der Bewegungssteuerungsvorrichtung bezogen. Vielmehr handelt es sich um abstrakte Operationen, die alle Hardware-Konfigurationen der Bewegungssteuerungsvorrichtungen ausführen müssen, damit sie funktionieren.

[0071] Bewegungssteuerungsoperationen können entweder Grundoperationen oder Nicht-Grundoperationen sein. Grundoperationen sind Operationen, die für die Bewegungssteuerung notwendig sind und die nicht durch die Verwendung einer Kombination von anderen Bewegungssteuerungsoperationen simuliert werden können. Beispiele für Grundoperationen sind unter anderem GET POSITION und MOVE RELATIVE, die für die Bewegungssteuerung notwendig sind und nicht durch die Verwendung anderer Bewegungssteuerungsoperationen emuliert werden können. Nicht-Grundoperationen sind Bewegungssteuerungsoperationen, die die Definition einer Grundoperation nicht erfüllen. Beispiele für Nicht-Grundoperationen sind unter anderem CONTOUR MOVE, die unter Verwendung einer Kombination von Bewegungssteuerungs-Grundoperationen emuliert werden können.

[0072] Nimmt man eine Gruppe von Bewegungssteuerungsoperationen wie oben definiert, so definiert der Softwaresystemplaner als nächstes eine Dienstanbieterschnittstelle (SPI), die eine Anzahl von Treiberfunktionen umfasst. Treiberfunktionen können entweder Kerntreiberfunktionen oder erweiterte Treiberfunktionen sein. Kerntreiberfunktionen sind Grundoperationen zugeordnet, während erweiterte Treiberfunktionen Nicht-Grundoperationen zugeordnet sind. Wie bei Bewegungssteuerungsoperationen sind die Treiberfunktionen nicht auf eine bestimmte Hardware-Konfiguration bezogen; grundsätzlich definieren Treiberfunktionen Parameter, die für die Implementierung von Bewegungssteuerungsvorgängen im typischen Sinne notwendig sind, die aber diesen Parametern keine spezifischen Werte oder dergleichen beiordnen.

[0073] Als nächstes definiert der Softwaresystemplaner eine Anwendungsprogrammierschnittstelle (API), die eine Gruppe von Komponentenfunktionen enthält. Für diese Komponentenfunktionen schreibt der Softwaresystemplaner einen Komponentencode, der zumindest einige der Komponentenfunktionen mit zumindest einigen der Treiberfunktionen assoziiert. Das Verhältnis zwischen Komponentenfunktionen und Treiberfunktionen muss nicht 1:1 betragen: zum Beispiel sind bestimmte Komponentenfunktionen für administrative Zwecke vorgesehen und haben keine entsprechende Treiberfunktion. Jedoch haben die meisten Komponentenfunktionen eine zugehörige Treiberfunktion.

[0074] Das durch das Softwareprogramm **22** implementierte Gesamtsoftware-Modell enthält somit eine API (Anwendungsprogrammierschnittstelle) mit Komponentenfunktionen und eine SPI (Dienstanbieterschnittstelle) mit Treiberfunktionen, wobei die API durch den Komponentencode, der den Komponentenfunktionen zugeordnet ist, mit der SPI in Beziehung steht.

[0075] Damit das System **22** Steuerbefehle generieren kann, werden wenigsten zwei weitere Komponenten benötigt: das Anwendungsprogramm **26** und zumindest ein Softwaretreiber wie z.B. die bei Pos. **30a**, **30b** und **30c** in **Fig.** 1 gezeigten Treiber.

[0076] Die Softwaretreiber **30** werden normalerweise von einem Hardwareplaner entwickelt und werden jeweils einer einzelnen Bewegungssteuerungsvorrichtung zugeordnet. Der Hardwareplaner schreibt den Treibercode, der diktiert, wie Steuerbefehle zum Steuern der zugeordneten Bewegungssteuerungsvorrichtungen zu generieren sind, damit die mit zumindest einigen der Treiberfunktionen verbundenen Bewegungssteuerungsoperationen durchgeführt werden.

[0077] In dem exemplarischen Softwaresystem **22** sind die Softwaretreiber **30a**, **30b** und **30c** den jeweiligen Bewegungsteuerungsvorrichtungen **20a**, **20b** und **20c** zugeordnet. Da für jede der Bewegungssteuerungsvorrichtungen **20a**, **20b** und **20c** ein Softwaretreiber vorhanden ist, bilden diese Vorrichtungen **20a**, **20b** und **20c** eine Gruppe von unterstützten Bewegungssteuerungsvorrichtungen.

[0078] Eine sorgfältige Durchsicht des Rahmenwerks des Softwaresystems **22** wie oben beschrieben wird zeigen, dass von allen Komponenten dieses Systems **22** nur die Softwaretreiber **30** hardwareabhängig sind.

[0079] Der Planer für das Bewegungssteuerungssystem, normalerweise auch der Benutzer **24**, entwickelt das Anwendungsprogramm **26**. Das Anwendungsprogramm **26** umfasst eine Sequenz von Komponentenfunktionen, die die Bewegungssteuerungsoperationen definieren, die für die Steuerung einer Bewegungssteuerungsvorrichtung zum Bewegen eines Objekts in einer gewünschten Weise notwendig sind. Das Anwendungsprogramm **26** ist jede Anwendung, die das System **22** durch Programmieren der Bewegungssteuerungs-Komponente **35** verwendet. Anwendungen können das System **22** entweder durch OLE-Automatisierung oder durch die Verwendung von irgendwelchen kundenspezifischen OLE-Schnittstellen, die die API bilden, programmieren.

[0080] Wie vorstehend erwähnt, assoziiert der Komponentencode viele der Komponentenfunktionen mit den

Treiberfunktionen, und die Treiberfunktionen definieren die Parameter, die für die Durchführung der Bewegungssteuerungsoperationen notwendig sind. Mit geeignet geordneten Komponentenfunktionen enthält das Anwendungsprogramm **26** somit auch die Logik, die für die Bewegung des Objekts in der gewünschten Weise notwendig ist.

[0081] Ist das Anwendungsprogramm **26** einmal geschrieben und sind die Softwaretreiber **30** bereitgestellt, wählt der Benutzer **24** zumindest eine Bewegungssteuerungsvorrichtung aus der Gruppe unterstützter Bewegungssteuerungsvorrichtungen **20a**, **20b** und **20c** aus. Unter Verwendung eines Treiber-Administrator-Moduls **32** wählt der Benutzer **24** dann den der gewählten Bewegungssteuerungsvorrichtung zugeordneten Softwaretreiber. Dieses Treiber-Administrator-Modul **32** wird zum Installieren, Deinstallieren, Registrieren und für das Setup eines jeden Stream (Datenstroms) verwendet.

[0082] Mit der gegenwärtigen Implementierung erlaubt der Treiber-Administrator **32** die Wahl nur eines Softwaretreibers. In künftigen Versionen des Softwaresystems **22** wird der Treiber-Administrator dem Benutzer die Wahl eines oder mehrerer Softwaretreiber ermöglichen.

[0083] Das Softwaresystem **22** generiert dadurch Steuerbefehle auf Grundlage der in dem Anwendungsprogramm **26** enthaltenen Komponentenfunktionen, des den Komponentenfunktionen zugeordneten Komponentencodes und des dem gewählten Softwaretreiber **28** zugeordneten Treibercodes.

[0084] Da die Steuerbefehle wie oben beschrieben generiert werden, können sie direkt zu einer Bewegungssteuerungsvorrichtung übertragen werden, um diese Vorrichtung in Echtzeit zu steuern, oder sie können für eine spätere Verwendung in einer Ausgabedatei gespeichert werden. Das Softwaresystem **22** benutzt die Streams **28**, um die Übertragung der Steuerbefehle zu ihrem gewünschten Ziel abzuwickeln.

[0085] In dem als Beispiel dienenden System **22** können die Zielorte der Steuerbefehle eine oder mehrere Ausgabedateien **34** und/oder die Steuereinheiten **16** sein. Andere mögliche Zielorte sind unter anderem ein Debug-Monitor (Fehlersuchmonitor) oder -Fenster oder ein anderer kundenspezifischer Ausgabemechanismus, der für eine bestimmte Situation definiert ist. Der Softwaresystemplaner oder in manchen Fällen der Hardwaresystemplaner schreiben einen Sende-Streamcode für jeden Stream **28**, der bestimmt, wie die Steuerbefehle zu einem gegebenen der Zielorte **16** und **34** der Steuerbefehle zu übertragen sind. Indem er den Treiber-Administrator **32** verwendet, wählt der Benutzer **24** einen oder mehrere der Zielorte **16** und **34** der Steuerbefehle, und beim späteren Betrieb überträgt das Steuersystem **22** die Steuerbefehle zu dem für die Steuerbefehle gewählten Zielort **16** und/oder **34** auf Grundlage des Sende-Streamcodes in dem Stream **28**, der dem für den Steuerbefehl gewählten Zielort **16** und/oder **34** zugeordnet ist.

[0086] Viele Steuerbefehl-Zielorte wie **16** und **34** sind zur Rückübertragung von Daten zu dem System **22** in der Lage. Daten, die von einem Steuerbefehl-Zielort an das System **22** zurückgesandt werden, werden Antwortdaten genannt. Der Softwaresystemplaner schreibt deshalb einen Daten-Antwortstreamcode für jeden der Streams **28a**, **28b** und **28c**, der bestimmt, wie die Antwortdaten von den Steuereinheiten **16** an das System **22** zu senden sind. Das System **22** verarbeitet die von den Steuereinheiten **16** gesandten Antwortdaten basierend auf dem in den Streams **28** enthaltenen Daten-Antwortstreamcode.

[0087] Es wird erneut auf **Fig.** 1 Bezug genommen. Dann ist gezeigt, dass das System **22** ferner eine Bewegungssteuerungs-Komponente **35** und ein Treiber-Stubroutine-Modul **36** aufweist. Das Bewegungssteuerungs-Komponenten-Modul **35** ist der Bereich des Softwaresystems **22**, der die Komponentenfunktionen mit den Treiberfunktionen in Beziehung setzt. Das Bewegungssteuerungs-Komponenten-Modul **35** enthält somit den Komponentencode, der die Assoziation zwischen den in dem Anwendungsprogramm **26** enthaltenen Komponentenfunktionen und den Treiberfunktionen herstellt.

[0088] Das Treiber-Stubroutine-Modul **36** wird für die Implementierung des durch das System **22** implementierten Basis-Softwaremodells nicht benötigt, aber es verleiht dem System **22** eine bedeutend größere Flexibilität für die Unterbringung von verschiedenen Bewegungssteuerungs-Hardewarekonfigurationen bei minimalem Aufwand.

[0089] Insbesondere wenn das Treiber-Stubroutine-Modul **26** verwendet wird, braucht der Hardwareplaner keinen Treibercode für die Implementierung aller Treiberfunktionen zu entwickeln; im Gegenteil, der Hardewareplaner muss einen Treibercode für die Implementierung der Kerntreiberfunktionen schreiben, aber nicht für die Implementieung der erweiterten Treiberfunktionen. Der Softwaresystemplaner versieht die Treiber-Stubroutine **36** für die Bewegungssteuerung mit dem Stubcode, der diejenigen Kombinationen von Kerntreiber-

funktionen identifiziert, die zur Emulation der Funktionalität der erweiterten Treiberfunktionen verwendet werden.

[0090] Die Bewegungssteuerungs-Komponente **24** bestimmt für den gewählten Softwaretreiber **30**, ob und gegebenenfalls welche erweiterten Funktionen der gewählte Treiber **30** unterstützt. Für erweiterte Funktionen, die nicht unterstützt werden (in der vorliegenden Beschreibung als nichtunterstützte erweiterte Treiberfunktion bezeichnet), wendet sich die Bewegungssteuerungs-Komponente **35** an das Treiber-Stubroutine-Modul **36**, um die geeignete Kombination von Kerntreiberfunktionen zum Emulieren der Funktionsvielfalt der nichtunterstützten erweiterten Treiberfunktionen zu bestimmen. Dadurch generiert das System **22** unter Verwendung der geeigneten Kombination von Kerntreiberfunktionen die Steuerbefehle, die für die Implementierung der nichtunterstützten erweiterten Treiberfunktionen notwendig sind.

[0091] Der Prozess zur Bestimmung, wann die erweiterten Treiberfunktionen emuliert werden müssen, lässt sich optimieren, indem die Bewegungssteuerungs-Komponente **35** mit einer Funktionszeigertabelle versehen wird, die einen Zeiger auf jede der erweiterten Funktionen enthält. Beim Erstellen der Funktionszeigertabelle prüft die Bewegungssteuerungs-Komponente **35** das gewählte Treibermodul **30**, um zu sehen, ob dieses jede erweiterte Funktion unterstützt. Wenn das gewählte Treibermodul **30** die erweiterte Funktion unterstützt, speichert das Bewegungssteuerungs-Komponenten-Modul **35** einen Zeiger (eine Hinweisadresse) für die durch das gewählte Treibermodul **30** implementierte Funktion an der der erweiterten Funktion entsprechenden Speicherstelle der Tabelle. Falls das gewählte Treibermodul **30** die erweiterte Funktion nicht unterstützt, speichert das Bewegungssteuerungs-Komponenten-Modul **35** einen Zeiger auf die sich in dem Treiber-Stubroutine-Modul **36** befindende Version der erweiterten Treiberfunktion. Die Implementierung der erweiterten Funktion mit dem Treiber-Stubroutine-Modul **36** enthält Aufrufe einer Vielzahl von Kernfunktionen, die durch das gewählte Treibermodul **30** implementiert werden.

[0092] Deshalb erlaubt das Treiber-Stubroutine-Modul **36** dem Planer des Bewegungssteuerungssystems bei minimalen Zeit- und Arbeitsaufwand seitens des Hardwareplaners die Verwendung eines Arbeitssoftwaretreibers **28**, der den Treibercode zum Implementieren nur der Kernfunktionen enthält. Der für die Implementierung der Kerntreiberfunktionen entwickelte Softwaretreiber **28** kann dann durch die Entwicklung eines Treibercodes verbessert werden, um je nach Wunsch erweiterte Treiberfunktionen zu implementieren.

[0093] Allgemein bevorzugt man die Verwendung eines Treibercodes, der speziell für die Implementierung erweiterter Treiberfunktionen ausgelegt ist, statt sich auf das Treiber-Stubroutine-Modul **36** zu verlassen, um die erweiterten Treiberfunktionen zu emulieren; ein Treibercode, der speziell für die Implementierung einer erweiterten Treiberfunktion geschrieben ist, erreicht fast immer eine Implementierung, die besser optimiert ist als die Emulation dieser Treiberfunktion mit einer Kombination von Kerntreiberfunktionen.

[0094] Es wird noch einmal kurz auf **Fig.** 1 Bezug genommen. Diese Figur zeigt, dass das System **22** außerdem ein Treiber-Administrator-CPL-Applet **38** (Hilfsprogramm bzw. kleines Anwendungsprogramm im Dialogfeld (control panel) des Treiber-Administrators) und einen DDE-Server **40** (Server für dynamischen Datenaustausch) aufweist. Das Treiber-Administrator-CPL-Applet **38** generiert die Benutzerschnittstelle, über welche der Benutzer **24** mit dem Treiber-Administrator-Modul **32** kommuniziert. Der DDE-Server **40** stellt die Softwareschnittstelle zur Verfügung, über welche das Anwendungsprogramm **26** mit dem Bewegungssteuerungs-Komponenten-Modul **35** kommuniziert.

II. Bewegungssteuerungs-Komponente

[0095] Die Bewegungssteuerungs-Komponente **35** wird nunmehr mit Bezug auf die Fig. 2 bis Fig. 10 näher erläutert. Die Bewegungssteuerungs-Komponente **35** wird von jeder Anwendung, die das System **22** für die Durchführung von Bewegungssteuerungsoperationen programmiert, benutzt. Die Hauptgruppe der API wird durch diese Komponente implementiert. Im Betrieb interagiert die Bewegungssteuerungs-Komponente **35** mit dem Treiber-Administrator **32**, damit der gegenwärtige Treiber und der Treiber **30** und die Treiber-Stubroutine **36** Bewegungssteuerungsoperationen durchführen. Anwendungen, die das System **22** benutzen, interagieren nur mit der Bewegungssteuerungs-Komponente **35**.

[0096] Dieser Abschnitt beschreibt die Planung der Bewegungssteuerungs-Komponente **35** in drei Hauptteilen. Zuerst werden alle binären Module, die die Komponente **35** betreffen, zusammen mit ihren Interaktionen mit der Komponente **35** beschrieben. Danach wird das Abbild der Interaktionen detailliert erläutert, um die Interaktionen zwischen allen C++-Objekten zu zeigen, die zur Implementierung der Bewegungssteuerungs-Komponente **35** verwendet werden. Dann wird das Abbild der Interaktionen getestet, indem die speziel-

len Interaktionen angezeigt werden, die während bestimmter Schlüsselprozesse stattfinden, zu deren Durchführung die Komponente **35** aufgefordert wird.

[0097] Das in Fig. 2 gezeigte Modul-Interaktions-Abbild zeigt alle binären Module und ihre Interaktionen mit der Bewegungssteuerungs-Komponente **35**. Wie aus dem Modul-Interaktions-Abbild ersichtlich ist, kommunizieren nur Anwendungen mit der Bewegungssteuerungs-Komponente **35**. Vom diesem Punkt aus koordiniert die Komponente **35** alle Interaktionen zwischen den Komponenten: Treiber-Administrator **32**, Treiber **30** und Treiber-Stubroutine **36**.

[0098] Bricht man das Modul-Interaktions-Abbild ab und fügt die zwischen allen C++-Objekten, die zur Implementierung der Bewegungssteuerungs-Komponente **35** verwendet werden, stattfindenden Interaktionen hinzu, entsteht das Objekt-Interaktions-Abbild, das in Fig. 3 gezeigt ist.

[0099] Jedes Objekt in dem Diagramm wird wie folgt beschrieben. Das Objekt CCmpntDisp ist das Dispatch-Objekt (Objekt Abfertigen/Senden), das benutzt wird, um die aufgedeckten Schnittstellen-Methoden zu senden. Während des Dispatch-Prozesses werden sämtliche Rohdaten in die geeignete C++-Form konvertiert. Zum Beispiel werden Sammlungen von Daten, die zwischen OLE-Komponenten übermittelt werden, gewöhnlich in einen Rohdatenblock des Speichers gepackt. Das Objekt CCmpntDisp sorgt dafür, dass ausgehende Daten komprimiert und eingehende Daten dekomprimiert werden. Das Komprimieren bzw. Packen von Daten beinhaltet das Konvertieren der Daten zwischen einem Rohformat und einem systemeigenen C++-Format.

[0100] Das Objekt CDriverAdmin wird benutzt, um direkt mit der Treiber-Administrator-Komponente zu kommunizieren. Alle OLE-bezogenen Details werden in dieser Klasse verkapselt.

[0101] Das Objekt CDriverMgr wird benutzt, um die gesamte Einheiten-Abbildung zu steuern, bevor die geeignete Treiberfunktion aufgerufen wird. Das Objekt CUnitMapper wird benutzt, um die tatsächliche Abbildung zwischen Einheiten vorzunehmen.

[0102] Das Objekt CUnitMapper wird benutzt, um Einheiten zwischen dem Teilekoordinatensystem (PCS = Part Coordinate System) und dem Maschinenkoordinatensystem (MCS = Machine Coordinate System) abzubilden. Dieses Objekt nimmt die Einheiten-Abbildung in beiden Richtungen vor.

[0103] Das Objekt CDriver wird verwendet, um die SPI-Tabelle zu erstellen, die sowohl Kerntreiberfunktionen als auch erweiterte Treiberfunktionen enthält. Je nach Ebene der Treiberunterstützung können die erweiterten Funktionen in der SPI-Tabelle auf Funktionen hinweisen, die entweder in der Treiber-Stubroutine **36** oder in dem Treiber **30** implementiert sind.

[0104] Die folgenden Erläuterungen der Fig. 4 bis Fig. 8 beschreiben alle Haupt-Szenarien oder Operationen, die an der Bewegungssteuerungs-Komponente **35** stattfinden. Jedes Szenario-Abbild zeigt alle Objekte, die beteiligt sind, und die Interaktionen zwischen den Objekten in der Reihenfolge, in der sie stattfinden.

[0105] Wie Fig. 4 zeigt, muss eine Anwendung, bevor sie die Bewegungssteuerungs-Komponente **35** benutzen kann, unter Verwendung der OLE-Funktion CoCreateInstance eine Instanz bzw. ein Exemplar des Objekts erstellen und dann das Exemplar aufrufen, indem sie die durch die Komponente **35** implementierte aufgedeckte kundenspezifische Schnittstellen-Methode Initialize aufruft. Fig. 4 zeigt die Sequenz von Ereignissen, die bei Aufruf der Methode Initialize stattfinden.

[0106] Während der Initialisierung finden die folgenden Schritte statt: Zuerst muss die Anwendung ein Exemplar der Bewegungssteuerungs-Komponente **35** erstellen, indem sie die Standard-OLE-Funktion CoCreateInstance aufruft Ist diese einmal geladen, muss die Anwendung die aufgedeckte Initialize-Methode der Komponente **35** aufrufen. Wenn diese erstmalig geladen wird, lädt die Komponente **35** sämtliche vorher gespeicherten Registrierungsdaten. Danach weist die Komponente **35** das CCmpntDisp an, das System zu initialisieren. Das CCmpntDisp weist den CDriverAdmin an, den(die) zu verwendenden aktuellen Treiber zu holen. Der CDriverAdmin lädt unter Anwendung der Standard-OLE-Funktion CoCreateInstance den Treiber-Administrator **32** und initialisiert als nächstes den Treiber-Administrator. Anschließend fragt es den Treiber-Administrator nach einem zu benutzenden Treiber (zu benutzenden Treibern) und deren SPI-Unterstützungsinformation. Schließlich gibt der Treiber-Administrator den(die) Treiber und die Unterstützungsinformation zurück an die Komponente **35** und gibt alle von der Treiber-Administrator-Komponente **32** benutzte Schnittstellen frei.

[0107] Sobald der(die) aktive(n) Treiber **30** und ihre Unterstützungsinformation empfangen wurde(n), übergibt die Bewegungssteuerungs-Komponente **35** den(die) Treiber **30** an den CDriverMgr mit der Anweisung, das System zu initialisieren. Während seiner Initialisierung initialisiert der CDriverMgr den CUnitMapper. Ebenfalls während der Initialisierung initialisiert der CDriverMgr einen CDriver für jeden verwendeten Treiber. Nach dem Initialisieren eines jeden CDriver wird die Unterstützungsinformation verwendet, um jede SPI-Tabelle in jedem Objekt CDriver zu erstellen. Beim Erstellen der SPI-Tabelle werden alle Kern- und unterstützten erweiterten SPI-Schnittstellen von dem Treiber abgefragt. Beim Erstellen der SPI-Tabelle fragt der CDriver auch alle von dem Treiber **30** nicht unterstützten Schnittstellen aus der Treiber-Stubroutine **36** ab.

[0108] Wenn nun bezugnehmend auf Fig. 5 die Bewegungssteuerungs-Komponente **35** einmal initialisiert ist, kann die Anwendung **26** Operationen an ihr ausführen. Es gibt zwei Arten von Operationen, die an der Komponente **35** stattfinden können: Operationen, die Kerntreiberfunktionen benutzen, und Operationen, die erweiterte Treiberfunktionen benutzen. Wenngleich der Unterschied zwischen den beiden für die die Komponente **35** benutzende Anwendung völlig unsichtbar ist, sind die internen Interaktionen zwischen den beiden verschieden. In der folgenden Abhandlung werden diese Unterschiede erläutert.

[0109] Die folgenden Interaktionen finden statt, wenn die Komponente **35** eine Operation durchführt, die nur Kerntreiberfunktionen benutzt. Zunächst muss die Anwendung die Operation anfordern und alle relevanten Parameter an die Komponente **35** übermitteln. Als nächstes wird das CCmpntDisp von der Komponente **35** angewiesen, die Operation durchzuführen. Das CCmpntDisp weist dann den CDriverMgr an, die Operation durchzuführen und übermittelt alle relevanten Parameter an den CDriverMgr. Vor Durchführung der Operation benutzt der CDriverMgr den CUnitMapper, um alle Einheiten in das Maschinenkoordinatensystem (MCS) zu konvertieren. Danach weist der CDriverMgr das Objekt CDriver an, die Operation durchzuführen und übergibt diesem sämtliche relevanten Parameter. Das Objekt CDriver benutzt seine interne SPI-Tabelle, um direkt mit der durch die Treiber-Komponente implementierten Kerntreiberfunktion zu kommunizieren.

[0110] Fig. 6 zeigt die Sequenz von Ereignissen, die stattfinden, wenn die Komponente **35** angewiesen wird, eine Operation durchzuführen, die zufällig eine erweiterte SPI benutzt, die von dem Treiber **30** nicht unterstützt wird. Die folgenden Schritte finden statt, wenn die Operation angefordert wird.

[0111] Zunächst muss die Anwendung die Operation anfordern und alle relevanten Parameter an die Komponente **35** übermitteln. Dann weist die Komponente **35** das CCmpntDisp an, die Operation durchzuführen. Das CCmpntDisp weist dann den CDriverMgr an, die Operation durchzuführen und übergibt alle relevanten Parameter an den CDriverMgr. Vor Durchführung der Operation benutzt der CDriverMgr den CUnit-Mapper, um alle Einheiten in das Maschinenkoordinatensystem (MCS) zu konvertieren. Danach weist der CDriverMgr das Objekt CDriver an, die Operation durchzuführen und übergibt diesem die neu abgebildeten Parameter. Das Objekt CDriver benutzt seine interne SPI-Tabelle, um direkt mit der durch die Treiber-Komponente implementierten Kerntreiberfunktion zu kommunizieren.

[0112] Wie vorstehend kurz erläutert, werden, wenn das System **22** benutzt wird, verschiedene Arten von Einheiten und zwei verschiedene Koordinatensysteme verwendet. Der Prozess der Einheiten-Abbildung beinhaltet das Konvertieren von Messwerten zwischen dem Teile- und dem Maschinenkoordinatensystem. Fig. 7 zeigt diesen Prozess, und die folgenden Schritte finden statt, wenn die Operation angefordert wird.

[0113] Zuerst muss die Anwendung die Operation anfordern und alle relevanten Parameter an die Komponente **35** übergeben. Es ist anzumerken, dass alle Parameter in dem PCS (Parts Coordinate System = Teilekoordinatensystem) sind. Die Komponente **35** weist das CCmpntDisp dann an, die Operation durchzuführen. Das CCmpntDisp weist den CDriverMgr an, die Operation durchzuführen und übergibt diesem die PCS-Parameter. Der CDriverMgr übernimmt alle Messwerte und benutzt den CUnitMapper für deren Konvertierung in das MCS (Machine Coordinate System = Maschinenkoordinatensystem). Die neu abgebildeten Parameter werden dann an den CDriver übergeben. Der CDriver weist entweder den Treiber oder die Treiber-Stubroutine-Komponente an, die Operation durchzuführen.

[0114] Wenn die Anwendung die Benutzung der Bewegungssteuerungs-Komponente **35** zu Ende geführt hat, weist sie die Komponente **35** an, alle ihre Betriebsmittel freizugeben, indem sie ihre aufgedeckte Methode Release aufruft. Dieser Prozess ist in Fig. 8 dargestellt. Während des Reinigungsprozesses finden die folgenden Schritte statt:
Zunächst muss die Anwendung die Komponente **35** anweisen, durch Aufrufen ihrer Methode Release alle ihre Betriebsmittel freizugeben. Ist der Aufruf erfolgt, übergibt die Komponente **35** den Aufruf an das Objekt CCmpntDisp. Das Objekt CCmpntDisp weist den CDriverMgr an, sämtliche von ihm benutzten Betriebsmittel

freizugeben. Der CDriverMgr weist jedes Objekt CDriver an, jedes seiner Betriebsmittel freizugeben und löscht dann die CDriver-Objekte. Zuerst befreit das Objekt CDriver sämtliche von ihm benutzten Schnittstellen von der Treiber-Komponente. Dann befreit das Objekt CDriver sämtliche von ihm benutzten Schnittstellen von der Treiber-Stubroutine-Komponente.

[0115] Fig. 9 ist ein Schnittstellen-Abbild, das die Bewegungssteuerungs-Komponente **35** betrifft. Fig. 10 ist ein Daten-Abbild, das zeigt, wie die Daten gespeichert werden, die sich auf die Frage beziehen, ob erweiterte Treiberfunktionen emuliert werden müssen.

III. Software-Treiber

[0116] Der Treiber **30** wird sowohl von dem Treiber-Administrator **32** als auch von der Komponente **35** benutzt. Seine Hauptaufgabe ist die Implementierung einer Funktionalität, die Bewegungssteuerungsbefehle für spezielle unterstützte Hardware generiert. Zum Beispiel generiert der Treiber AT6400, der für die Steuerung der Bewegungssteuerungs-Hardware des Compumotor AT6400 verwendet wird, AT6400-Befehlscodes. Während der Initialisierungsphase des Systems **22** kommuniziert der Treiber-Administrator **32** mit jedem Treiber **30**, wodurch es dem Benutzer möglich ist, die Konfiguration des Treibers hinzuzufügen, zu entfernen oder zu ändern. Wenn eine das System **22** benutzende Anwendung läuft, kommuniziert die Komponente **35** mit dem Treiber **30** und weist diesen an, die geeigneten Bewegungssteuerungsoperation durchzuführen.

[0117] Dieser Abschnitt beschreibt die vollständige Planung eines auswählbaren Treibers **30**. Alle Treiber sind aus dem in diesem Handbuch beschriebenen Basisentwurf entwickelt.

[0118] Dieser Abschnitt ist in drei Teile unterteilt. Als erstes wird ein Modul-Interaktions-Abbild diskutiert, das alle binären Module beschreibt, die mit dem Treiber **30** interagieren. Dann wird das Modul-Interaktions-Abbild als ein Objekt-Interaktions-Abbild gezeichnet, in dem alle Interna des Treibers aufgedeckt werden. In diesem Abbild sind alle C++-Objekte, die den Treiber bilden, und ihre Interaktionen dargestellt. Dann werden verschiedene Szenarien-Abbilder gezeichnet. Jedes Szenario-Abbild zeigt die Interaktionen, die zwischen den an bestimmten Prozessen beteiligten C++-Objekten stattfinden. Schließlich beschreibt dieser Abschnitt die von der Treiber-Komponente aufgedeckten Schnittstellen, alle verwendeten Datenstrukturen und die Definitionen einer jeden verwendeten C++-Klasse.

[0119] Bezugnehmend auf Fig. 11 zeigt das Modul-Interaktions-Abbild alle binären Module und deren Interaktionen bzw. Dialoge mit dem Treiber **30**. Es gibt zwei Module, die in direktem Dialog mit dem Treiber stehen: die Bewegungssteuerungs-Komponente **35** und der Treiber-Administrator **32**. Der Treiber Administrator **32** fragt die Treiber-Einstellungen ab und ändert sie, und die Komponente **35** weist den Treiber an, die Bewegungssteuerungsoperation durchzuführen, wie zum Beispiel die Bewegung zu einer bestimmten Stelle in dem System. Bei Pos. **42** in Fig. 11 ist die Standard-Registrierungsdatenbank von Windows gezeigt, die in der vorliegenden Beschreibung als Registratur bezeichnet wird.

[0120] Wenn man das Modul-Interaktions-Abbild stärker aufschließt, indem man die Interaktionen einfügt, die zwischen allen für die Implementierung des Treibers verwendeten C++-Objekten stattfindet, erhält man das Objekt-Interaktions-Abbild. Das Objekt-Interaktions-Abbild für den Treiber **30** ist in Fig. 12 gezeigt.

[0121] Jedes Objekt in dem Diagramm wird folgendermaßen beschrieben.

[0122] Das CDriverDisp ist das Objekt Dispatch (Senden), das benutzt wird, um die aufgedeckten Schnittstellen-Methoden zu senden. Während des Dispatch-Prozesses werden alle Rohdaten in die geeignete C++-Form konvertiert. Zum Beispiel werden Sammlungen von Daten, die zwischen OLE-Komponenten übermittelt werden, normalerweise zu Rohdatenblöcken des Speichers komprimiert. Das Objekt CDriverDisp sorgt für das Packen ausgehender Daten und das Auspacken eingehender Daten. Das Packen bzw. Komprimieren schließt die Konvertierung von Daten zwischen einem Rohformat und einem systemeigenen C++-Format ein.

[0123] Das Objekt CStreamMgr ist verantwortlich für das Management der Gruppe von Streams (Strömen), die bei dem Treiber registriert sind. Streams können hinzugefügt, entfernt und aktiviert werden. Nur aktivierten Streams werden Daten zugesandt. Der CLSID- und Betriebsbereitschafts-Status eines jeden registrierten Stream wird in der Registrierungsdatenbank gespeichert. Beim Kommunizieren mit Streams wird der CStreamMgr benutzt, um die Command Strings (Befehlszeichenfolgen) an alle aktivierten Streams zu senden.

[0124] Das Objekt CCommandMgr wird verwendet, um zu dem Stream gesandte Befehle zu bilden und um

Antworten zu extrahieren, die von dem Stream empfangen werden. Der CCommandMgr ist das Kontroll-Objekt, das die Objekte CResponse, CCommandList und CStream verwaltet.

[0125] Das Objekt CCommandList speichert die komplette Liste von Befehlen, die die Befehlssprache für die Bewegungssteuerung bilden. Solche Befehle können als Text-Betriebsmittel oder in einer Textdatei gespeichert werden.

[0126] Das Objekt CCcomand erstellt Command Strings, die dann an den CStream gesandt werden. Jeder gebildete Befehl ist ein kompletter Bewegungssteuerungs-String.

[0127] Das Objekt CResponseList erstellt CResponse-Objekte, die mit dem Parsingformat (Syntaxanalyseformat) für die erwartete Antwort initialisiert werden.

[0128] Das Objekt CResponse konvertiert Raw Response Strings (Rohantwortzeichenketten), die durch den CStream rückübermittelt werden, und konvertiert sie in C++-Datentypen. Zum Beispiel kann ein Antwort-String, der Positionsdaten enthält, in eine Gruppe von Doppelwerten konvertiert werden.

[0129] Das Objekt CStream wird benutzt, um direkt mit der zugrundeliegenden Stream-Komponente zu kommunizieren.

[0130] Die Fig. 14 bis Fig. 20 enthalten Szenarien-Abbilder, die alle Hauptszenarien oder Operationen beschreiben, die in dem Treiber **30** stattfinden. Jedes Szenario-Abbild zeigt alle beteiligten Objekte und die Interaktionen bzw. Dialoge zwischen diesen Objekten in der Reihenfolge, in der sie stattfinden.

[0131] Es gibt zwei Arten von Operationen, die in dem Treiber **30** stattfinden. Zuerst kann der Treiber-Administrator **32** Operationen initiieren, wie zum Beispiel das Hinzufügen von Streams oder das Konfigurieren des Treibers. Als nächstes kann die Bewegungssteuerungs-Komponente **35** Operationen an dem Treiber initiieren, wenn eine Anmeldung tatsächlich läuft. Die folgende Erläuterung beschreibt jede Perspektive, beginnend mit Operationen, die von dem Treiber-Administrator **32** angewiesen werden; sämtliche Operationen, die von dem Treiber-Administrator an dem Treiber vorgenommen werden, werden in der Reihenfolge beschrieben, in der sie bei der Benutzung des Treibers vorkommen können.

[0132] Bevor ein Treiber benutzt werden kann, muss er in dem OLE-System registriert werden. Um einen Treiber zu registrieren, verifiziert der Treiber-Administrator zuerst, dass das zu registrierende Modul tatsächlich ein Treiber **30** ist, dann ruft er die exportierte Funktion des DDLRegisterServer auf, um den Treiber zu registrieren. Jedes Modul des Systems **22** exportiert eine Funktion, die DLLGetModuleType genannt wird. Diese Funktion wird benutzt, um zu bestätigen, dass das Modul eine Treiber-Komponente **30** ist. Fig. 13 zeigt die Interaktionen, die im Zuge der Registrierung eines Treibers stattfinden.

[0133] Während des in Fig. 13 dargestellten Registrierungsprozesses finden folgende Schritte statt: Erst muss der Treiber-Administrator die DLL (DYNAMIC LINK LIBRARY), die die Stream-Komponente enthält, laden und bestätigen, dass das Modul ein Treiber **30** ist. Wenn die Funktion einen Wert rückübermittelt, der den Wert XMC_DRIVER_MT im höherwertigen Byte enthält, fährt der Treiber-Administrator fort und registriert den Treiber, indem er seine exportierte Funktion, DLLRegisterServer, aufruft. Nach deren Aufruf schreibt die Implementierung von DLLRegisterServer sämtliche OLE 2.0-Registrierungsinformationen in die Windows Registrierungsdatenbank.

[0134] Nachdem, bezugnehmend auf Fig. 14, der Treiber registriert ist, kann der Treiber-Administrator unter Verwendung der OLE-Funktion CoCreateInstance die Komponente **35** laden. Während des Initialisierungsprozesses lädt der Treiber sämtliche Registrierungsdaten und initialisiert sowohl das Objekt CDriverDisp als auch die C++-Objekte CStreamMgr.

[0135] Während der Initialisierung finden die folgenden Schritte statt.

[0136] Vor dem Laden der Treiber-Komponente muss der Treiber-Administrator das Treiber-Modul nach seiner CLSID fragen. Der Aufruf der exportierten Funktion des Treibers, DLLGetCLSID, reicht die CLSID zurück. Wenn der Treiber-Administrator die CLSID einmal hat, kann er ein Exemplar des Treibers erstellen, indem er die Standard-OLE-Funktion CoCreateInstance aufruft. Beim erstmaligen Laden lädt der Treiber sämtliche vorher gespeicherten Registrierungsdaten. Als nächstes weist der Treiber das Objekt CDriverDisp an, das System zu initialisieren. Sobald benachrichtigt, initialisiert sich das Objekt CDriverDisp und weist dann den CStreamM-

gr an, sich zu initialisieren. Während seiner Initialisierung lädt der CStreamMgr sämtliche Stream-Einstellungen aus der Registrierungsdatenbank. Zum Beispiel werden die CLSID und der Freigabezustand sämtlicher vorher beim Treiber registrierten Streams geladen.

[0137] Nach dem Initialisieren des Treibers kann der Treiber-Administrator Operationen an dem Treiber durchführen. Zum Beispiel kann der Treiber-Administrator den Treiber auffordern, einen Stream hinzuzufügen oder zu löschen. Fig. 15 zeigt die Sequenz von Ereignissen, die stattfinden, wenn der Treiber aufgefordert wird, einen neuen Stream hinzuzufügen. Wenn ein Stream hinzugefügt wird, finden die folgenden Schritte statt.

[0138] Zuerst weist der Treiber-Administrator den Stream an, einen neuen Stream hinzuzufügen und übergibt die CLSID des hinzuzufügenden Stream an den Treiber. Der Treiber übergibt die CLSID an das Objekt CDriverDisp und weist dieses an, den Stream hinzuzufügen. Im letzten Schritt nimmt der CStreamMgr an, dass das Modul eine gültige Stream-Komponente **28** ist und fügt die CLSID dem Informationssatz des Treibers in der Registrierungsdatenbank hinzu.

[0139] Eine weitere nach der Initialisierung von dem Treiber angeforderte Operation ist die der Abfrage seiner gegenwärtigen Einstellungen. Vor dem Anzeigen von Informationen über den Treiber, wie beispielsweise der Name der von ihm unterstützten Hardware, muss der Treiber-Administrator den Treiber auf Informationen abfragen. Zum Beispiel zeigt Fig. 16 den Prozess der Abfrage des Treibers auf eine Nummerierung der bei ihm registrierten Streams. Beim Abfragen des Treibers auf Informationen finden folgende Schritte statt:
Zuerst ruft der Treiber-Administrator die Schnittstellen-Methode auf, die zur Abfrage des Treibers auf die Nummerierung der Streams verwendet wird. Das Objekt CDriverDisp weist das CStreamDisp dann an, eine Stream-Nummerierung zu erstellen. Das Objekt CDriverDisp weist schließlich den CStreamMgr an, die Stream-Nummerierung vorzubereiten. Der CStreamMgr überprüft die Registrierungsdatenbank und stellt sicher, dass ihr interner Zustand mit den in der Registratur gespeicherten Daten synchronisiert ist. Danach setzt er eine Sperre, die bewirkt, dass sämtliche Stream-Managementoperationen wie das Hinzufügen oder Entfernen von Streams fehlschlagen. Der CStreamMgr bereitet eine Liste von Streams vor und lädt sie unter Benutzung des Objekts CStream in den Speicher. Das Objekt CStream lädt die Stream-Komponente unter Verwendung der OLE-API CoCreateInstance.

[0140] Nachdem der Treiber-Administrator die Benutzung des Treibers abgeschlossen hat, muss er den Treiber freigeben, indem er seine aufgedeckte Release-Methode aufruft. Das Aufrufen dieser Methode weist den Treiber an, alle verwendeten Betriebsmittel freizugeben. Fig. 17 zeigt den Prozess der Freigabe der Treiber-Komponente. Während des Reinigungsprozesses finden die folgenden Schritte statt.

[0141] Zuerst muss der Treiber-Administrator die Treiber-Komponente anweisen, sich selbst zu reinigen, indem er deren Release-Methode aufruft. Nachdem sie aufgerufen ist, übergibt die Treiber-Komponente den Aufruf an das Objekt CDriverDisp. Das Objekt CDriverDisp weist den CStreamMgr danach an, alle Daten zu sichern. Der CStreamMgr sichert alle Daten, den Zustand eines jeden Stream eingeschlossen, in der Registrierungsdatenbank. Schließlich sichert der Treiber alle internen Daten in der Registrierungsdatenbank.

[0142] Nachdem ein Treiber erfolgreich in dem System **22** installiert und unter Verwendung des Treiber-Administrators konfiguriert wurde, ist er bereit für die Benutzung durch die Bewegungssteuerungs-Komponente **35**. Die Komponente **35** benutzt den Treiber **30**, wenn sie Bewegungssteuerungsoperationen durchführt, die von der die Komponente **35** benutzenden Anwendung angefordert wurden. Der folgende Beitrag beschreibt die von der Komponente **35** dirigierten Operationen, die an dem Treiber stattfinden können.

[0143] Ehe der Treiber benutzt werden kann, muss er von der Komponente **35** initialisiert werden. Diese Operation unterscheidet sich von der Treiber-Initialisierung, die bei Benutzung durch den Treiber-Administrator an dem Treiber stattfindet, weil das System für das Senden und den Empfang von Befehlen vorbereitet werden muss. Zur Vorbereitung auf die Datenkommunikation muss der Stream initialisiert und dann geöffnet werden. Fig. 18 beschreibt den Initialisierungsprozess. Die folgenden Schritte finden während des Initialisierungsprozesses statt.

[0144] Zuerst muss die Komponente **35** den Treiber anweisen, sich selbst zu initialisieren. Dies ist gewöhnlich ein Vorgang mit zwei Schritten. Im ersten Schritt erstellt die Komponente **35** unter Verwendung der OLE-Funktion CoCreateInstance ein Exemplar des Treibers. Als nächstes wird die von dem Treiber aufgedeckte Methode Initialize aufgerufen, um den Treiber für die Datenübertragungen vorzubereiten. Wenn die Methode Initialize aufgerufen wird, lädt der Treiber zuerst sämtliche in der Registrierungsdatenbank **42** gespeicherten internen Daten. Dann weist der Treiber das CDriverDisp an, das interne System zu initialisieren. Das CDriverDisp

schließlich weist den CStreamMgr an, die Streams zu initialisieren. Dann lädt der CStreamMgr alle Daten aus der Registrierungsdatenbank, und zwar einschließlich der Gruppe aller CLSIDs und aktivierten Zustände für alle bei dem Treiber registrierten Streams. Anschließend lädt der CStreamMgr jeden aktivierten Stream durch Erstellen eines neuen Objekts CStream für jeden aktivierten Stream. Beim Erstellen eines jeden Objekts CStream wird die CLSID für den zugrundeliegenden Stream an das Objekt CStream übergeben. Wenn jedes Objekt CStream erstellt und bei einer Stream-Komponente angemeldet ist, lädt es die Komponente **35** durch Aufrufen der Standard-OLE-Funktion CoCreateInstance. Sobald der CStreamMgr fertig ist, weist das CDriverDisp den CCommandMgr an, sich selbst zu initialisieren. Während seines Initialisierungsprozesses initialisiert und lädt der CCommandMgr die CCommandList. Wenn sich CCommandMgr initialisiert lädt er die CResponseList entsprechend der CCommandList.

[0145] Ist das System einmal initialisiert, kann die Bewegungssteuerungs-Komponente **35** den Treiber anweisen, bestimmte Befehlsoperationen durchzuführen. Befehlsoperationen sind Standard-Bewegungssteuerungsoperationen wie das Bewegen zu einer bestimmten Stelle in dem System oder die Abfrage des Systems nach seiner momentanen Position. Fig. 19 beschreibt den Prozess der Befehlserteilung an den Treiber, eine bestimmte Operation durchzuführen. Wenn dem Treiber befohlen wird, eine bestimmte Operation durchzuführen, finden die folgenden Schritte statt.

[0146] Zuerst weist die Komponente **35** den Treiber an, die Operation durchzuführen, zum Beispiel die Bewegung in eine Position oder die Abfrage des Systems nach der momentanen Position. Dann weist der Treiber das Objekt CDriverDisp an, eine Operation durchzuführen. Danach weist das Objekt CDriverDisp den CCommandMgr an, einen geeigneten Befehl zu erstellen. Sämtliche für den Befehl relevanten Parameter werden dem CCommandMgr übergeben. Wenn der Treiber zum Beispiel angewiesen wird, sich in eine bestimmte Position zu bewegen, wird die Positionsinformation an den CCommandMgr geleitet. Als nächstes fordert der CCommandMgr die CResponseList auf, ein Objekt CResponse zu erstellen. CResponseList fragt das Antwortformat ab und benutzt es, um ein neues Objekt CResponse zu schaffen, das an den CCommandMgr zurückgereicht wird. Dann weist der CCommandMgr die CCommandList an, den Befehl zu erstellen. Sämtliche für den Befehl relevanten Parameter werden an die CCommandList übergeben. Die CCommandList erstellt ein neues Objekt CCommand, fragt den Rohbefehl-String ab und übergibt diesen und die Befehlsparameter an das Objekt CCommand, das dann den Befehls-String aufbaut.

[0147] Der CCommandMgr übergibt danach das Objekt CCommand, das von der CCommandList zurückgereicht wurde, und das vorher erstellte Objekt CResponse an das Objekt CStreamMgr. Das Objekt CStreamMgr wird angewiesen, die Objekte zu verarbeiten. Der CStreamMgr reicht die Objekte CCommand und CResponse weiter an alle aktivierten Objekte CStream. Das Objekt CStream fragt das Objekt CCommand nach dem vollen Befehls-String in Rohtextform ab. Der Rohtext-Befehl wird an die Stream-Komponente übergeben. Als nächstes wartet das Objekt CStream auf die Antwort und liest die Rohtextantwort dann in einen Zwischenspeicher ein. Die Rohtext-Antwort wird dann an das Objekt CResponse übergeben. Danach wird das Objekt CRETONNE an den CStreamMgr zurückgegeben, der es an den CCommandMgr zurückgibt, der es an das Objekt CDriver zurückgibt. CResponse kehrt schließlich zurück zu dem Objekt CDriverDisp, das CResponse anweist, die Antwort in einen auswählbaren C++-Typ zu konvertieren. Der auswählbare Typ wird an die Bewegungssteuerungs-Komponente **35** zurückgegeben.

[0148] Sobald die Komponente **35** mit der Benutzung des Treibers fertig ist, muss der Treiber durch Aufrufen seiner Methode Release freigegeben werden. Die Freigabe des Treibers löscht alle von dem Treiber benutzten Betriebsmittel. Fig. 20 beschreibt den Prozess der Freigabe des Treibers. Die folgenden Schritte finden beim Reinigen und Löschen sämtlicher von dem Treiber benutzter Betriebsmittel statt.

[0149] Zuerst muss die Komponente **35** die Methode Release des Treibers aufrufen. Ist sie aufgerufen, weist der Treiber das Objekt CDriverDisp an, jegliche benutzten Betriebsmittel freizugeben. Das CDriverDisp weist danach den CStreamMgr an, jegliche benutzen Betriebsmittel zu löschen. CStreamMgr löscht daraufhin alle aktiven Objekte CStream. Jedes Objekt CStream gibt alle benutzten Stream-Komponenten-Schnittstellen frei. Als nächstes weist das CDriverDisp den CCommandMgr an, alle seine Betriebsmittel zu löschen. Während dieser Reinigung löscht der CCommandMgr das Objekt CCommandList. Um seine Reinigung zu vollenden, löscht der CCommandMgr das Objekt CResponseList.

[0150] Anhang D ist ein Dokument, das die tatsächlichen aufgedeckten OLE-Schnittstellen, die während des Umlaufs von Daten verwendeten Definitionen der Datenstrukturen und die Definitionen jeder von dem Treiber intern benutzten Klasse beschreibt.

IV. Streams

[0151] Dieser Abschnitt beschreibt die Stream-(Strom)-Komponente **28**, die als Schicht für den Datentransport zwischen der Treiber-Komponente **30** und dem Ziel-Ausgabeort wie die Bewegungssteuerungsvorrichtung **20** und/oder die Ausgabedatei **34** beschreibt. Wenn zum Beispiel Bewegungssteuerungs-Hardware verwendet wird, die mit dem PC-Bus verbunden ist, kommuniziert die Treiber-Komponente **30** mit der PC-Bus-Stream-Komponente **28**.

[0152] Die Planung einer Stream-Komponente **28** wird in drei Teilen erläutert. Zuerst beschreibt ein Modul-Interaktions-Abbild die hinsichtlich des Stream (Stroms) beteiligten Module und wie sie interagieren bzw. aufeinander einwirken. Dann schließt das Objekt-Interaktions-Abbild das Modul-Interaktions-Abbild in eine detailliertere Ansicht auf, die nicht nur die zwischen den Modulen stattfindenden Interaktionen darstellt, sondern auch die Interaktionen, die zwischen den C++-Objekten innerhalb der Stream-Komponente **28** stattfinden. Dann wird das Objekt-Interaktions-Abbild "getestet", indem man es durch verschiedene Szenarien-Abbilder laufen lässt. Jede Szenario-Abbildung zeigt die Objekt-Interaktionen, die während einer bestimmten Operation stattfinden.

[0153] Das in Fig. 22 gezeigte Modul-Interaktions-Abbild zeigt alle Module, die mit der Stream-Komponente **28** interagieren. Interaktionen beginnen aus zwei verschiedenen Perspektiven. Zuerst interagiert der Treiber-Administrator **32** mit der Stream-Komponente **28**, wenn der Stream installiert, entfernt und konfiguriert wird. Danach interagiert jeder Treiber **30**, wenn er benutzt wird, mit dem Stream, während er Daten zu dem Bestimmungsort sendet und von dem Bestimmungsort holt. Wenn zum Beispiel ein Treiber Daten in einen Textdatei-Stream schreibt, sorgt der Stream dafür, dass die Daten an die Datei ausgeschrieben werden. Oder wenn der Treiber Daten aus einem PC-Bus-Stream liest, besorgt der Stream das eigentliche Auslesen aus der Hardware und reicht die Daten zurück an den Treiber.

[0154] Treiber kommunizieren nur mit Datenströmen, die speziell mit dem Treiber verbunden wurden. Ist die Verbindung einmal hergestellt, wird der Stream benutzt, um mit dem Bestimmungsobjekt wie dem PC-Bus, der seriellen E/A-Anschlussstelle, der Textdatei oder dem Debug-Monitor zu kommunizieren.

[0155] Die in Fig. 22 gezeigte Stream-Komponente **28** ist das Objekt, das als Datentransportschicht für jeden Treiber arbeitet. Jeder Stream hat ein anderes Ziel, das den Typ des Stream definiert. Die folgenden sind die gegenwärtigen Stream-Ziele.

[0156] PC-Bus/WinNT – Dieser Windows NT-Stream arbeitet mit einem Windows NT. SYS Gerätetreiber, um direkt mit der Bewegungssteuerungs-Hardware zu kommunizieren, die mit dem PC-Bus verbunden ist.

[0157] PC-Bus/Win95 – Dieser Windows 95-Stream arbeitet mit einem Windows 95 VxD, um direkt mit der Bewegungssteuerungs-Hardware zu kommunizieren, die mit dem PC-Bus verbunden ist.

[0158] PC-Bus/Win 3.1 – Dieser Windows 3.1-Stream kommuniziert direkt mit der mit dem PC-Bus verbundenen Bewegungssteuerungs-Hardware.

[0159] Seriell – Dieser Stream arbeitet mit der COMM API, um mit der an den seriellen Anschluss angeschlossenen Bewegungssteuerungs-Hardware zu kommunizieren.

[0160] Textdatei – Dieser Stream ist write-only (nur Schreiben) und sendet alle Daten an eine Textdatei.

[0161] Debug-Monitor – Dieser Stream ist write-only (nur Schreiben) und sendet alle Daten zum Debug-Monitor.

[0162] Custom – Dies ist ein kundenspezifischer Stream, der Daten zu einem unbekannten Ort sendet.

[0163] Ähnlich wie das Modul-Interaktions-Abbild zeigt das Objekt-Interaktions-Abbild Interaktionen zwischen den Modulen. Darüber hinaus zeigt dieses Abbild alle Interaktionen, die zwischen jedem C++-Objekt innerhalb der Stream-Komponente **28** stattfinden. Fig. 23 ist das Objekt-Interaktions-Abbild für die Stream-Komponente **28**.

[0164] Jedes Objekt in dem Diagramm wird folgendermaßen dargestellt. Das Objekt CStream ist das Objekt Dispatch (Senden), das verwendet wird, um aufgedeckte Schnittstellen-Methoden zu senden. Während des

Dispatch-Prozesses werden alle Daten in die geeignete C++-Form konvertiert. Zum Beispiel werden Sammlungen von Daten, die zwischen OLE-Komponenten übermittelt werden, gewöhnlich in einen Rohdatenblock des Speichers gepackt. Das Objekt CStreamDisp sorgt für das Packen ausgehender Daten und das Auspacken eingehender Daten. Das Packen von Daten beinhaltet das Konvertieren der Daten zwischen einem Rohformat und einem systemeigenen C++-Format.

[0165] Der CRegistryMgr sorgt für die Verwaltung aller in der Registrierungsdatenbank gespeicherten Daten. Da viele Streams des gleichen Typs gleichzeitig existieren können, wird jedem Stream ein Handle zugeordnet. Der zugeordnete Handle wird von dem Stream benutzt, um den Ort aufzufinden, den er zum Laden und Speichern von Daten in der Registrierungsdatenbank verwendet, genauso wie ein Bücherei-Index verwendet wird, um ein Buch in einer Bücherei aufzufinden.

[0166] Alle Eingaben und Ausgaben werden durch den CIOMgr-Manager zusammengeführt. Die Verwaltung von Eingabe- und Ausgabeoperationen besteht aus der Zwischenspeicherung von Daten und der Steuerung von Grundroutinen, die für den Transport von Daten zu und von dem Zielort verwendet werden.

[0167] Das Objekt CIOHAL ist die Abstraktionsschicht der Eingabe/Ausgabe-Hardware. Innerhalb dieses Objekts liegen all hardwareabhängigen Codes wie Aufrufe von inp und outp. Jeder unterschiedliche Streamtyp enthält eine unterschiedliche Implementierung dieses Objekts.

[0168] Szenario-Abbilder sind spezialisierte Objekt-Interaktions-Abbilder, die zeigen, wie jedes Modul und die Objekte in der Stream-Komponente während der durch das Abbild beschriebenen Operationen interagieren bzw. aufeinander einwirken. Die Szenario-Abbilder in den Fig. 24–Fig. 32 sind in zwei verschiedene Kategorien unterteilt; solche, die durch den Treiber-Administrator **32** initiiert werden, und solche, die durch den Treiber **30** initiiert werden.

[0169] Durch den Treiber-Administrator dirigierte Operationen beziehen sich normalerweise auf das Initialisieren, De-Initialisieren und Konfigurieren des Stream. Die folgenden Abschnitte beschreiben alle an dem Stream stattfindenden Operationen, die von dem Treiber-Administrator angewiesen werden.

[0170] Bevor eine Stream-Komponente von jemandem benutzt werden kann, muss sie in der Windows Registrierungsdatenbank registriert werden. Die Registrierung ist eine OLE 2.0-Standardoperation, die notwendig ist, damit eine OLE 2.0-Komponente wie beispielsweise die Stream-Komponente benutzt werden kann. Fig. 24 beschreibt diesen Prozess. Während des Registrierungsprozesses finden die folgenden Schritte statt.

[0171] Zuerst muss der Treiber-Administrator die die Stream-Komponente enthaltende DLL laden und bestätigen, dass das Modul eine Stream-Komponente **28** ist. Dazu ruft der Treiber-Administrator die durch den Stream exportierte Funktion DLLGetModuleType auf. Wenn das höherwertige Byte im Rückgabewert den Wert XMC_STREAM_MT enthält, fährt der Treiber-Administrator fort und registriert den Stream durch Aufrufen seiner exportierten Funktion DLLRegisterServer. Wenn sie aufgerufen ist, schreibt die Implementierung von DLLRegisterServer alle OLE 2.0-Registrierungsinformationen in die Windows Registrierungsdatenbank.

[0172] Nach erfolgreicher Registrierung der Stream-Komponente ist sie bereit für die Initialisierung. Während der Initialisierung initialisiert sich die Stream-Komponente nicht nur selbst, sondern sie initialisiert sämtliche benutzten Gerätetreiber durch Registrierung des Treibers bei dem Betriebssystem. Zum Beispiel registriert die Windows NT Stream-Komponente den Windows NT .SYS Treiber bei Windows NT und startet den Dienst. Fig. 25 beschreibt diesen Prozess. Während der Initialisierung finden die folgenden Schritte statt.

[0173] Zuerst muss der Treiber-Administrator den Stream anweisen, sich selbst zu initialisieren. Wenn er diesen Aufruf macht, werden der Name und der Ort des benutzten Treibers und der Handle des Stream als Argumente in die Methode eingeführt. Sobald sie angewiesen wurde, sich selbst zu initialisieren, ruft die Stream-Komponente das CStreamDisp auf und weist dieses an, das System zu initialisieren. Das CStreamDisp gibt dann Anweisung an den CRegistryMgr, alle relevanten Daten für den Stream unter Verwendung des ihm verliehenen Handle zu laden. Der CRegistryMgr lädt alle Daten aus der Registrierungsdatenbank. Nachdem sämtliche Informationen aus der Registratur geladen wurden, weist das CStreamDisp den CIOMgr an, den geeigneten Treiber bei dem Betriebssystem zu registrieren. Der CIOMgr weist die CIOHAL an, den Treiber, sofern geeignet, zu registrieren. Wenn sie in Windows NT läuft, registriert die CIOHAL den .SYS-Treiber bei dem Windows NT Betriebssystem und startet den Treiber. Bei Ausführung in Windows 95 wird die VxD-Integrität mit einem schnellen dynamischen Laden und Entladen verifiziert.

[0174] Nach dem Initialisieren der Stream-Komponente kann diese auf ihre aktuellen Einstellungen abgefragt werden oder angewiesen werden, neue Einstellungen vorzunehmen. Da beide Operationen sehr ähnlich sind, wird nur die Änderung der Einstellungen beschrieben. Stream-Einstellungen umfassen Daten wie: Port-Adressen, IRQ-Level, Dateinamen etc. Daten, die für die Kommunikation mit dem Eingabe/Ausgabe-Ziel benötigt werden, sind in den Stream-Einstellungen enthalten. Fig. 26 beschreibt den Prozess der Änderung der Stream-Einstellungen. Während des Setup-Prozesses finden folgende Schritte statt.

[0175] Zunächst weist der Treiber-Administrator den Stream an, die überreichten Daten für die Änderung seiner internen Daten zu benutzen. Sobald diese Anweisung erfolgt ist, übergibt die Stream-Komponente den Aufruf der Schnittstellen-Methode an das Objekt CStreamDisp. Das Objekt CStreamDisp weist dann den CRegistryMgr an, die neuen Einstellungen zu speichern. Der CRegistryMgr speichert die neuen Einstellungen in der Registrierungsdatenbank.

[0176] Wenn der Treiber-Administrator mit der Benutzung einer Stream-Komponente fertig ist, muss er die verwendeten Betriebsmittel löschen. Fig. 27 zeigt diesen Prozess. Während des Reinigungsprozesses finden die folgenden Schritte statt. Zuerst muss der Treiber-Administrator die Stream-Komponente anweisen, sich durch den Aufruf ihrer Release-Methode selbst zu reinigen. Wenn der Aufruf erfolgt ist, gibt der Stream den Aufruf weiter an das Objekt CStreamDisp. Das Objekt CStreamDisp weist den CRegistryMgr dann an, alle Daten zu sichern. Alle relevanten Daten werden durch den CRegistryMgr in der Registrierungsdatenbank gesichert.

[0177] Treiber-dirigierte Operationen finden statt, wenn jeder Treiber **30** die mit ihm verbundene Stream-Komponente **28** benutzt. An dieser Stelle sei daran erinnert, dass jede Stream-Komponente als Datentransportschicht verwendet wird. Jeder Treiber benutzt den Stream, um die Bewegungssteuerungs-Befehlsdaten, die er generiert, zum Ausgabeziel zu übertragen. Streams werden auch zur Rückübertragung von Daten zu dem Speicher verwendet, wenn Leseoperationen stattfinden. Nur bestimmte Streams sind lesbar.

[0178] Ehe der Treiber Operationen an dem Stream durchführen kann, muss der Stream initialisiert werden. Die Initialisierung findet in zwei Schritten statt Erst muss die OLE-Stream-Komponente geladen werden, und wenn dies getan ist, muss der Stream ausdrücklich initialisiert werden. Fig. 28 beschreibt den zweiten Abschnitt des Initialisierungsprozesses. Während des Initialisierungsprozesses finden folgende Schritte statt.

[0179] Erst muss der Treiber die durch eine der Stream-Schnittstellen exportierten Initialize-Methoden aufrufen. Wenn Initialize aufgerufen wird, übergibt der Treiber den Stream-Handle an den Stream. Als nächstes gibt der Stream die Anweisung weiter an das Objekt CStreamDisp zum Senden. Das Objekt CStreamDisp weist zuerst den CRegistryMgr an, alle an dem durch den Stream-Handle definierten Ort gespeicherten Einstellungen zu laden. Der CRegistryMgr liest die Daten ein, die bei dem Handle in der Registratur gespeichert sind. Sind die Daten geladen, weist das CStreamDisp den CIOMgr an, sich selbst zu initialisieren. Als Teil seiner Intitialisierung initialisiert der CIOMgr das Objekt CIOHAL, das er benutzt.

[0180] Ist der Stream erst initialisiert, muss er geöffnet werden. Das Öffnen eines Stream versetzt den Stream in einen Zustand, in dem er Daten zwischen dem Treiber und dem Ziel übermitteln kann. Fig. 29 beschreibt den Prozess des Öffnens eines Stream. Beim Öffnen eines Stream finden folgende Schritte statt.

[0181] Als erstes weist der Treiber den Stream an, sich selbst zu öffnen, indem er die aufgedeckte Schnittstellen-Methode Open (Öffnen) aufruft. Ist er geöffnet, leitet der Stream den Aufruf weiter zu dem Objekt CStreamDisp. Danach weist das Objekt CStreamDisp den CIOMgr an, den Stream zu öffnen. Zu diesem Zeitpunkt bereitet der CIOMgr sämtliche Zwischenspeicher vor, die später beim Übertragen von Daten durch den Stream benutzt werden. Sind die Zwischenspeicher bereit, weist der CIOMgr das Objekt CIOHAL an, mit dem Ziel zu interagieren und es zu öffnen. CIOHAL kommuniziert direkt mit dem Ziel oder mit einem Gerätetreiber und öffnet den Stream. Beim Arbeiten mit Hardware-Streams kommunizieren der Treiber oder der serielle E/A direkt mit der Hardware und bereiten sie für den Betrieb vor.

[0182] Nach dem Öffnen eines Stream ist dieser bereit für die Durchführung von Datentransport-Operationen. Es sind zwei Hauptoperationen für den Datentransport verfügbar: das Lesen von Daten und das Einlesen von Daten. Fig. 30 beschreibt den Prozess des Einlesens von Daten in den Stream. Beim Einlesen in den Stream finden die folgenden Schritte statt. Zuerst weist der Treiber den Stream an, Daten in das Ziel zu schreiben und übermittelt die Daten an den Stream. Dann übermittelt der Stream die Daten an den CIOMgr und weist diesen an, die Daten in das Ziel zu schreiben. Der CIOMgr übermittelt entweder den gesamten Datenblock an das Objekt CIOHAL oder speichert den Block in einem internen Zwischenspeicher und übermittelt Teile des Zwi-

schenspeichers an das Objekt CIOHAL, bis der komplette Zwischenspeicher gesendet ist. Das Objekt CIOHAL nimmt die ihm übermittelten Daten und sendet sie direkt an das Ziel oder übermittelt sie an einen Gerätetreiber oder ruft COMM API auf, um die Daten an die serielle E/A-Anschlussstelle zu senden. Der Gerätetreiber oder COMM API sendet die Daten direkt an die gesteuerte Hardware.

[0183] Bestimmte Streams wie der PC-Bus und serielle E/A-Streams (Übertragungsströme) geben die Daten zurück, nachdem Schreibvorgänge an ihnen stattgefunden haben. Die zurückgegebenen Daten können spezifisch sein für eine vorherige Datenanforderung oder für einen Status, der den Erfolg oder das Fehlschlagen der vorhergehenden Schreiboperation beschreibt. Fig. 31 beschreibt den Prozess des Lesens von Daten aus dem Stream. Hier ist anzumerken, dass nicht alle Streams lesbar sind. Derzeit sind die einzigen lesbaren Streams der PC-Bus und serielle Streams. Während der Operation des Lesens von Daten aus dem Ziel finden folgende Schritte statt.

[0184] Zunächst weist der Treiber den Stream an, Daten aus dem Ziel zu lesen. Der Stream übermittelt den Aufruf an das Objekt CStreamDisp. Das Objekt CStreamDisp weist den CIOMgr an, den Lesevorgang durchzuführen. Abhängig davon, wie der Stream implementiert ist, kann der CIOMgr entweder einen Aufruf oder mehrere Aufrufe an das Objekt CIOHAL senden. Das Objekt CIOHAL kommuniziert entweder direkt mit der Hardware, benutzt COMM API oder einen Gerätetreiber, um die Daten zu lesen. Werden ein Gerätetreiber oder die COMM API verwendet, kommunizieren diese direkt mit der Hardware, um die Daten zu lesen.

[0185] Ist der Treiber fertig mit der Benutzung des Stream, muss er den Stream anweisen, sämtliche von ihm benutzten Betriebsmittel zu reinigen. Dazu ruft der Treiber die standardmäßige Release-Methode auf. Fig. 32 zeigt die Sequenz von Ereignissen, die nach dem Aufruf der Release-Methode stattfinden. Die folgenden Schritte finden beim Reinigen und Löschen aller von dem Stream benutzten Betriebsmittel statt.

[0186] Zuerst muss der Treiber die Release-Methode des Stream aufrufen. Dann weist der Stream das Objekt CStreamDisp an, alle benutzten Betriebsmittel freizugeben. Das Objekt CStreamDisp gibt danach die Anweisung an den CIOMgr, sämtliche in den Zwischenspeichern etc. benutzte Betriebsmittel zu löschen. Als nächstes weist der CIOMgr das CIOHAL an, sämtliche benutzten Betriebsmittel zu löschen. Während dieser Reinigung und abhängig von dem Streamtyp wird das CIOHAL benutzte Textdateien löschen, den Debug-Monitor schließen, die Hardware abschalten oder Hardware-Treiber anweisen, die Hardware abzuschalten. Wenn Gerätetreiber oder die COMM API benutzt werden, weisen sie die Hardware an abzuschalten.

[0187] Fig. 33 zeigt ein Schnittstellen-Abbild für den Stream **28**.

V. Treiber-Stubroutine-Modul

[0188] Das Treiber-Stubroutine-Modul **36** wird verwendet, um die erweiterten Treiberfunktionen auszufüllen, die der Treiber **30** nicht unterstützen oder implementieren kann. Durch Simulieren der erweiterten SPI, können Anwendungen ein größere Gruppe von Bewegungssteuerungs-Funktionalitäten benutzen, als diese verfügbar wäre, wenn die Anwendung die Bewegungssteuerungs-Software direkt programmieren würde. Um die erweiterte SPI zu implementieren, verwendet die Treiber-Stubroutine Algorithmen, die Kern-SPI-Schnittstellen-Methoden aufrufen, die durch den Treiber **30** implementiert werden. Während der Initialisierung der Treiber-Stubroutine wird der zu benutzende Treiber **30** bei der Treiber-Stubroutine registriert.

[0189] Dieser Abschnitt beschreibt alle Aspekte der Treiber-Stubroutine in drei grundlegenden Teilen. Der erste Teil dieses Abschnitts beschreibt alle binären Module und ihre Interaktionen mit der Treiber-Stubroutine **36**. Wie aus Fig. 34 zu ersehen ist, wird die Treiber-Stubroutine von der Komponente **35** benutzt. Die Treiber-Stubroutine wirkt mehr oder weniger als Helfer für die Komponente **35**, indem sie alle möglichen erweiterten Treiberfunktionen ausfüllt.

[0190] Indem man das Modul-Interaktions-Abbild in Fig. 34 nimmt und alle Interaktionen darstellt, die mit allen die Treiber-Stubroutine implementierenden C++-Objekten stattfinden, produziert man ein sogenanntes Objekt-Interaktions-Abbild. Fig. 35 ist das Objekt-Interaktions-Abbild für die Treiber-Stubroutine-Komponente **36**.

[0191] Jedes Objekt in dem Diagramm wird wie folgt beschrieben.

[0192] Das Objekt CDriverStubDisp ist das Objekt Dispatch (Objekt Senden), das zum Senden aufgedeckter Schnittstellenmethoden verwendet wird. Während des Sendeprozesses werden alle Rohdaten in die geeignete C++-Form konvertiert. Zum Beispiel werden Sammlungen von Daten, die zwischen OLE-Komponenten über-

mittelt werden, gewöhnlich in einen Rohdatenblock des Speichers gepackt Das Objekt CDriverStubDisp sorgt für das Packen ausgehender Daten und für das Auspacken eingehender Daten. Das Packen der Daten beinhaltet das Konvertieren der Daten zwischen einem Rohformat und einem systemeigenen C++-Format.

[0193] Das Objekt CSPIMgr ist verantwortlich für die Verwaltung sämtlicher SPI-Angelegenheiten wie zum Beispiel die Verwaltung des CSimpleDriver, indem es diesen anweist, sich mit den geeigneten, von dem Treiber aufgedeckten SPI-Kernschnittstellen zu verbinden.

[0194] Das Objekt CSimpleDriver wird verwendet, um direkt mit dem die SPI-Kernschnittstellen implementierenden Treiber zu kommunizieren. Der CSimpleDriver kommuniziert nur mit den von dem Treiber implementierten SPI-Kernschnittstellen.

[0195] Die folgende Erläuterung behandelt alle Hauptszenarien oder Operationen, die an der Treiber-Stubroutine **36** stattfinden. Jedes Szenario-Abbild zeigt alle beteiligten Objekte und die Interaktionen zwischen diesen Objekten in der Reihenfolge, in der sie stattfinden. Alle Operationen an der Treiber-Stubroutine stammen aus der Bewegungssteuerungs-Komponente **35**. Zusätzlich zu der Bewegungssteuerungs-Komponente **35** interagiert die XMCSetup-Komponente mit der Treiber-Stubroutine, wenn das System **22** installiert wird. Hier sei bemerkt, dass alle unten dargestellten Szenarien annehmen, dass die Treiber-Stubroutine **36** bereits im OLE-System registriert wurde. Die Registrierung dieser Komponente ist die Verantwortlichkeit der Setup-Anwendung und Setup-Komponente.

[0196] Dieser Beitrag beschreibt alle Operationen, die von der Bewegungssteuerungs-Komponente **35** an der Treiber-Stubroutine durchgeführt werden. Die jeweiligen Abschnitte sind in der Reihenfolge ihres möglichen Erscheinens bei Benutzung des Treibers beschrieben.

[0197] Wie Fig. 36 zeigt, muss die Bewegungssteuerungs-Komponente **35**, ehe sie die Treiber-Stubroutine **36** benutzt, diese initialisieren, indem sie ein Exemplar bzw. eine Instanz der Treiber-Stubroutine erstellt und dann das erstellte Exemplar initialisiert. Das Aufrufen der Standard-OLE-Funktion CoCreateInstance beendet den ersten Schritt. Nachdem ein Exemplar erstellt wurde, muss die Komponente **35** die aufgedeckte Schnittstellen-Methode Initialize der Treiber-Stubroutine aufrufen. Während der Initialisierung finden folgende Schritte statt.

[0198] Die Komponente erstellt ein Exemplar bzw. eine Instanz der Treiber-Stubroutine durch Aufrufen der standardmäßigen OLE-Funktion CoCreateInstance. Einmal geladen, wird die CLSID des zu benutzenden Treibers an die Treiber-Stubroutine übermittelt, wenn deren aufgedeckte Schnittstellen-Methode Initialize aufgerufen wird. Beim erstmaligen Laden lädt der Treiber sämtliche vorher gespeicherten Registrierungsdaten. Dann übermittelt die Komponente **35** die CLSID des zu benutzenden Treibers an das Objekt CDriverStubDisp und weist diese an, das System zu initialisieren. Das Objekt CDriverStubDisp erteilt dann die Anweisung an den CSPIMgr, sich selbst zu initialisieren, und übermittelt ihm die Treiber-CLSID. Der CSPIMgr übermittelt die CLSID an das CSimpleDriver und weist diesen an, nur die von dem Treiber freigelegten Kern-SPI-Schnittstellen abzufragen. Das CSimpleDriver lädt ein Exemplar des Treibers und fragt alle von dem Treiber aufgedeckten Kernschnittstellen ab.

[0199] Ist die Treiber-Stubroutine einmal initialisiert, ist sie bereit für die Durchführung von Operationen wie beispielsweise die Durchführung erweiterter Treiberfunktionen. Fig. 37 beschreibt die Schritte, die stattfinden, wenn die Komponente **35** die Treiber-Stubroutine anweist, eine erweiterte SPI-Operation durchzuführen. Die folgenden Schritte finden statt, wenn die Operation angefordert wird.

[0200] Zunächst muss die Komponente **35** die Operation anfordern und alle relevanten Parameter an die Treiber-Stubroutine übermitteln. Danach weist die Treiber-Stubroutine das CDriverStubDisp an, diese Operation zu bewerkstelligen. Das CDriverStubDisp erteilt Weisung an den CSPIMgr, die erweiterte SPI-Funktion durchzuführen und übermittelt den geeigneten XMC_EXT_SPI-Identifier (Bezeichner) als einen Parameter. Der CSPIMgr ruft die geeignete Funktion entsprechend dem XMC_EXT_SPI-Identifier auf. Die Funktion simuliert die erweiterte Treiberfunktion und ruft das CSimpleDriver für Kernfunktionen auf. Wenn angewiesen, führt das CSimpleDriver SPI-Kernfunktionen durch, indem es die durch den Treiber implementierten aufgedeckten Schnittstellen direkt aufruft.

[0201] Wenn die Bewegungssteuerungs-Komponente **35** mit der Benutzung der Treiber-Stubroutine **36** fertig ist, muss sie diese freigeben, indem sie die aufgedeckte Release-Methode aufruft. Der Aufruf der Release-Methode veranlasst die Treiber-Stubroutine zum Löschen aller von ihr verwendeten Betriebsmittel. Fig. 38 zeigt

die Sequenz von Ereignissen. Während des Reinigungsprozesses finden folgende Schritte statt.

[0202] Zunächst muss die Komponente **35** die Treiber-Stubroutine anweisen, alle ihre Betriebsmittel freizugeben, indem sie ihre Release-Methode aufruft. Ist der Aufruf erfolgt, übermittelt die Treiberkomponente den Aufruf an das Objekt CDriverStubDisp. Das Objekt CDriverStubDisp weist dann den CSPIMgr an, jegliche von ihm benutzten Betriebsmittel freizugeben. Der CSPIMgr gibt alle benutzten Betriebsmittel frei, einschließlich des Objekts CSimpleDriver. Wenn gelöscht, gibt das CSimpleDriver sämtliche von dem Treiber benutzten Schnittstellen frei.

[0203] Fig. 39 ist ein Schnittstellen-Abbild des Treiber-Stubroutine-Moduls **36**.

VI. Treiber-Administrator-Modul

[0204] Der Treiber-Administrator **32** wird aus zwei unterschiedlichen Perspektiven benutzt. Wenn das Treiber-Administrator-CPL-Applet **38** (Applet = kleiner Programmabschnitt ohne Hauptfunktion) zum Konfigurieren des Systems verwendet wird, weist das Applet den Treiber-Administrator **32** an, die Operationen durchzuführen. Das Applet **38** liefert ganz einfach die Schnittstelle, und die Komponente **35** macht die echte Arbeit für die Verwaltung der in dem System **22** benutzten Treiber und Streams. Das Benutzen der Treiber-Administrator-Komponente mit dem Applet ist die erste Perspektive bei der Verwendung der Komponente **35**.

[0205] Aus der zweiten Perspektive benutzt die Bewegungssteuerungs-Komponente **35** die Treiber-Administrator-Komponente, um den Treiber **30** nach dem gegenwärtig freigegebenen Satz abzufragen. Hier ist zu bemerken, dass gegenwärtig nur eine Operation mit einem Treiber zugelassen wird. Das System **22** kann eindeutig mehrfache Treiber unterstützen, die virtualisiert sind. Wenn zum Beispiel zwei Vierachsen-Treiber installiert werden, könnten Anwendungen, die das System benutzen, so agieren, als würden sie ein Achtachsensystem benutzen.

[0206] Dieser Abschnitt beschreibt den Treiber-Administrator **32** in drei Hauptteilen. Zuerst werden alle mit der Treiber-Administrator-Komponente interagierenden Module zusammen mit ihren Interaktionen beschrieben. Als nächstes wird das Modul-Interaktions-Abbild erweitert, um alle Interaktionen darzustellen, die zwischen den C++-Objekten stattfinden, die zur Implementierung der Treiber-Administrator-Komponente **32** verwendet werden. Diese Beschreibung wird Objekt-Interaktions-Abbild genannt. Dann wird das Objekt-Interaktions-Abbild getestet, indem man es durch verschiedene Szenarien oder Szenario-Abbilder laufen lässt. Jedes Szenario-Abbild zeigt die Ereignisse und deren Reihenfolge in einem bestimmten Prozess, der an der Treiber-Administrator-Komponente stattfindet.

[0207] Das in Fig. 40 gezeigte Modul-Interaktions-Abbild zeigt alle binären Module und ihre Interaktionen mit der Treiber-Administrator-Komponente **32**. Sowohl das Treiber-Administrator-CPL **38** als auch die Bewegungssteuerungs-Komponente **35** sind die Hauptmodule, die mit der Treiber-Administrator-Komponente **32** interagieren.

[0208] Das Treiber-Administrator-CPL-Modul **38** liefert die Benutzerschnittstelle, die es dem Benutzer erlaubt, Treiber und Streams in dem System **22** hinzuzufügen, zu konfigurieren und zu entfernen. Der Treiber-Administrator **32** ist für das Management aller Treiber und Streams zuständig. Wenngleich das Applet die Schnittstelle bereitstellt, macht dieses Modul **32** die eigentliche Managementarbeit.

[0209] Außerdem wird der Treiber-Administrator von der Komponente **35** für den Zugriff auf den(die) zu benutzenden gegenwärtigen Treiber benutzt, wenn Bewegungssteuerungsoperationen durchgeführt werden. Wenn zum Beispiel der AT6400-Treiber als gegenwärtiger Treiber gewählt wird, wenn die Komponente **35** den Treiber-Administrator abfragt, schickt der Treiber-Administrator die CLSID des AT600-Treibers zurück.

[0210] Nimmt man den in dem Modul-Interaktions-Abbild dargestellten Treiber-Adminstrator **32** und zeigt alle zwischen den C++-Objekten, die zur Implementierung des Administrators **34** verwendet werden, vorkommenden Interaktionen an, so entsteht das Objekt-Interaktions-Abbild dafür. Das Objekt-Interaktions-Abbild für den Treiber-Administrator **32** ist in Fig. 41 gezeigt.

[0211] Jedes Objekt in dem Programm wird wie folgt beschrieben. Das Objekt CDriverAdminDisp ist das Objekt Dispatch (Senden), das zum Senden aufgedeckter Schnittstellen-Methoden verwendet wird. Während des Sendeprozesses werden alle Rohdaten in die geeignete C++-Form konvertiert Zum Beispiel werden Sammlungen von Daten, die zwischen OLE-Komponenten übermittelt werden, normalerweise in einen Rohdaten-

block des Speichers gepackt. Das Objekt CDriverAdminDisp sorgt für das Packen ausgehender Daten und für das Auspacken eingehender Daten. Das Packen von Daten beinhaltet das Konvertieren der Daten zwischen einem Rohformat und einem systemeigenen C++-Format.

[0212] Das Objekt CDriverInfoMap wird benutzt, um die Informationen abzubilden, die von dem Treiber-Administrator-CPL **38** verwendet werden, wenn Informationen über jeden Stream angezeigt werden.

[0213] Das Objekt CModuleMgr ist verantwortlich für das Management aller Stream- und Treibermodule in dem System. Eine Liste mit allen registrierten Treibern wird durch den CModuleMgr dauerhaft in der Registrierungsdatenbank gespeichert. Bei jedem Zugriff auf einen Treiber oder einen Stream wird der CModuleMgr benutzt, um das Modul zu holen.

[0214] Das Objekt CSimpleDriver wird verwendet, um direkt mit der Treiber-Komponente zu kommunizieren. Alle OLE-spezifischen Details sind in diesem Objekt gekapselt.

[0215] Das Objekt CSimpleStream wird verwendet, um direkt mit der Stream-Komponente zu kommunizieren. Alle OLE-spezifischen Details sind in diesem Objekt gekapselt.

[0216] Die Fig. 42 bis Fig. 49 beschreiben alle Hauptszenarien oder Operationen, die an dem Treiber-Administrator **32** stattfinden. Jedes Szenario-Abbild zeigt alle beteiligten Objekte und die Interaktionen zwischen diesen Objekten in der Reihenfolge, in der sie stattfinden.

[0217] Bezugnehmend auf Fig. 42 muss die Treiber-Administrator-Komponente, ehe sie benutzt wird, initialisiert werden. Fig. 42 beschreibt den Prozess des Initialisierens der Treiber-Administrator-Komponente entweder aus dem Dialogbox-Applet des Treiber-Administrators oder der Bewegungssteuerungs-Komponente. Während der Initialisierung finden die folgenden Schritte statt.

[0218] Zunächst müssen entweder das Dialogbox-Applet oder die Bewegungssteuerungs-Komponente ein Exemplar der Treiber-Administrator-Komponente erstellen, indem die Standard-OLE-Funktion CoCreateInstance aufgerufen wird. Als nächstes muss die aufgedeckte Schnittstellen-Methode aufgerufen werden. Wenn die Initialize-Methode aufgerufen wird, weist die Treiber-Administrator-Komponente das CDriverAdminDisp an, das System zu initialisieren. Dann weist das CDriverAdminDisp den CModuleMgr an, sich selbst und sämtliche von ihm verwaltete Module zu initialisieren. Der CModuleMgr lädt zuerst alle Informationen aus der Registrierungsdatenbank. Dann erstellt der CModuleMgr für jeden registrierten Treiber ein Exemplar des Treibers, indem er die Standard-OLE-Funktion CoCreateInstance aufruft. Anschließend ruft der CModuleMgr die Initialize-Methode eines jeden Treibers auf, indem er die CLSID der anzumeldenden Treiber-Komponente an die Methode übermittelt.

[0219] Der Treiber-Administrator **32** kann sowohl Treiber als auch Streams registrieren. Das Registrieren von Treibern erfolgt sehr direkt, da der Treiber-Administrator die in dem System registrierten Treiber verwaltet. Dagegen ist die Registrierung von Streams schwieriger, da jeder Stream bei einem Treiber registriert werden muss und der Treiber die bei ihm registrierten Streams verwaltet und nicht der Treiber-Administrator. Im Folgenden wird der Registrierungsprozess sowohl der Treiber als auch der Streams erläutert.

[0220] Die Registrierung eines Treibers hat zur Folge, dass verifiziert werden muss, dass das Modul tatsächlich ein Treiber ist und der Treiber geladen werden kann und dass die Treiberinformationen an einer bleibenden Stelle gespeichert wird. Fig. 43 beschreibt diesen Prozess. Beim Registrieren eines Treibers finden folgende Schritte statt.

[0221] Zuerst übermittelt die Treiber-Administrator-CPL den Namen des Treibers und weist die Treiber-Administrator-Komponente an, diesen zu registrieren. Dann übermittelt die Treiber-Administrator-Komponente den Treibernamen an das CDriverAdminDisp und weist dieses an, das Modul zu registrieren. Das CDriverAdminDisp weist den CModuleMgr an, den neuen Treiber zu registrieren. Der CModuleMgr erstellt ein neues CSimpleDriver und fordert es auf, den Treiber zu registrieren. Zuerst verifiziert das CSimpleDriver, dass der Treiber gültig ist, indem es seine exportierte Funktion DLLGetModuleType aufruft. Wenn die Funktion XMC_DRIVER_MT zurückgibt, ruft das CSimpleDriver schließlich die exportierte Funktion DLLRegisterServer des Treibers auf, um das Modul in dem OLE-System zu registrieren. Danach wird die CLSID von dem Modul abgefragt, indem seine exportierte Funktion DLLGetCLSID aufgerufen wird. Die zurückgegebene CLSID wird dann zum Laden des Treibers verwendet, indem die Standard-OLE-Funktion CoCreateInstance aufgerufen wird. Ist das CSimpleDriver erfolgreich, speichert der CModuleMgr die Treiber-CLSID in der Registrierungsda-

tenbank.

[0222] Die Registrierung eines Stream ist ähnlich wie die Registrierung eines Treibers, jedoch etwas komplizierter, da jeder Stream bei einem bestimmten Treiber registriert werden muss. Fig. 44 zeigt den Prozess der Registrierung eines Stream. Bei der Registrierung eines Stream finden folgende Schritte statt.

[0223] Zuerst übermittelt die Treiber-Administrator-CPL die CLSID des Treibers und den Dateinnamen des bei dem Treiber zu registrierenden Stream an die Treiber-Administrator-Komponente. Die Treiber-Administrator-Komponente weist das CDriverAdminDisp an, den Stream zu registrieren. Das Objekt CDriverAdminDisp weist den CModuleMgr an, den Stream zu registrieren. und übermittelt diesem die CLSID des Treibers zusammen mit dem Namen des Stream. Erst verifiziert der CModuleMgr, dass die CLSID des Treibers einer der registrierten Treiber ist. Wenn dies nicht der Fall ist, wird der Treiber wie oben beschrieben registriert.

[0224] Dann erstellt der CModuleMgr ein neues Objekt CSimpleStream und weist dieses an, die Stream-Komponente zu bestätigen und zu laden. Das CSimpleStream bestätigt zuerst, dass das Modul tatsächlich eine Stream-Komponente **28** ist, indem es seine exportierte Funktion DLLGetModuleType aufruft. Wenn die Funktion XMC_STREAM_MT zurückgibt, fährt das CSimpleStream fort und registriert die Stream-Komponente durch Aufrufen ihrer exportierten Funktion DLLRegisterServer. Schließlich fragt das Objekt CSimpleStream das neue Modul nach seiner CLSID, indem es die exportierte Funktion DLLGetCLSID des Moduls aufruft. Die neue CLSID wird von dem CSimpleStream benutzt, um die Stream-Komponente unter Verwendung der Standard-OLE-Funktion CoCreateInstance zu laden. Ist das CSimpleStream erfolgreich, wird die CLSID des Stream an den CSimpleDriver weiter übermittelt, der angewiesen wird, den Stream zu registrieren. Das CSimpleDriver übermittelt die CLSID an die Treiber-Komponente und weist sie an, den Stream zu registrieren.

[0225] Im Folgenden wird die Einstellung von Informationen entweder in einem Treiber oder einem Stream beschrieben. Wenn der Benutzer in dem Dialogbox-Applet **38** des Treiber-Administrators Informationen ediert bzw. aufbereitet, weist das Applet **38** den Treiber-Administrator **32** an, die Einstellungen für den Stream oder für den edierten Treiber aufzubereiten. Der folgende Beitrag beschreibt, wie dieser Konfigurationsprozess abläuft.

[0226] Die Aufbereitung der Einstellungen eines Treibers findet statt, wenn der Benutzer die in der Treiber-Administrator-CPL angezeigten Treibereinstellungen ändert. Eine Änderung dieser Einstellungen bewirkt, dass der in Fig. 45 beschriebene Prozess in der Treiber-Administrator-Komponente stattfindet. Die folgenden Schritte finden beim Einstellen der Treiberkonfiguration statt.

[0227] Wenn die Einstellungen in der CPL **38** geändert werden, weist die Treiber-Administrator-CPL die Treiber-Administrator-Komponente an, die geeigneten Änderungen an dem Treiber entsprechend dem Treiber-Handle vorzunehmen. Eine XMC_Driver_INFO-Struktur, die die neuen Werte für den Treiber beschreibt, wird an die Komponente **35** übermittelt. Der Treiber-Administrator bringt die XMC_DRIVER_INFO-Struktur und den Handle zu dem Treiber und übergibt die Information an das Objekt CDriverAdminDisp mit der Anweisung, die Einstellungen in dem Treiber zu ändern. Das Objekt CDriverAdminDisp weist den CModuleMgr an, den Treiber entsprechend dem Treiber-Handle zu edieren. Der CModuleMgr findet das CSimpleDriver mit dem Handle und weist dieses an, seine Einstellungen in diejenigen zu ändern, die in der XMC_DRIVER_INFO-Struktur gespeichert sind. Das CSimpleDriver übermittelt die XMC_DRIVER_INFO-Struktur an die Treiber-Komponente und weist sie an, ihre Einstellungen zu ändern.

[0228] Wie Fig. 46 zeigt, finden die folgenden Schritte statt, wenn der Benutzer Stream-Einstellungen in dem Treiber-Administrator-CPL **38** ändert.

[0229] Nachdem der Benutzer die Einstellungen für den Stream in der CPL geändert hat, weist die Treiber-Administrator-CPL die Treiber-Administrator-Komponente an, die Einstellungen des Stream zu ändern, und übermittelt einen Handle an den Treiber, der den Stream enthält, einen Handle an den Stream und eine XMC_STREAM_INFO-Struktur, die die neuen Werte beschreibt. Die Treiber-Administrator-Komponente weist das Objekt CDriverAdminDisp an, die Einstellungen der Streams zu ändern. Das Objekt CDriverAdminDisp weist den CModuleMgr an, die Einstellungen des Stream entsprechend dem Handle zu ändern.

[0230] Zuerst findet der CModuleMgr den Treiber entsprechend dem Treiber-Handle. Dann fordert er das CSimpleDriver auf, die Einstellungen für den Stream entsprechend dem Stream-Handle zu ändern. Das CSimpleDriver sucht nach dem Stream, der dem Stream-Handle entspricht, und weist diesen an, seine Einstellung

in diejenigen zu ändern, die in der XMC_STREAM_INFO-Struktur gespeichert sind. Das CSimpleStream kommuniziert direkt mit der Stream-Komponente und weist sie an, ihre Einstellungen in jene zu ändern, die in der XMC_STREAM_INFO-Struktur gespeichert sind.

[0231] Es gibt zwei verschiedene Arten von Informationen, nach denen der Treiber-Administrator **32** abgefragt werden kann: die Nummerierung aller registrierten Treiber und das Treiber-Informations-Abbild. Die Bewegungssteuerungs-Komponente **35** benutzt die Treibernummerierung bei der Auswahl der Gruppe von Treibern, die während der Bewegungssteuerungsoperationen zu benutzen und zu steuern/kontrollieren sind. Das Treiber-Informations-Abbild wird dagegen von dem Treiber-Administrator-CPL **38** benutzt, um die Benutzerschnittstellenanzeige, die alle in dem System registrierten Treiber und Streams beschreibt, zu aktualisieren. Der folgende Beitrag beschreibt die Abfrage sowohl der Treiber-Nummerierung als auch des Treiber-Informations-Abbilds. Die Abfrage nach der Treibernummerierung findet während der Initialisierung der Bewegungssteuerungs-Komponente **35** statt. Wenn sie initialisiert wird, muss die Komponente **35** wissen, welche Treiber sie benutzen muss, wenn sie Bewegungssteuerungsoperationen durchführt. Die Treiber-Administrator-Komponente **32** wird genau für diesen Zweck verwendet. Durch die Abfrage der Treibernummerierung wird ein Zeiger an die von der Treiber-Administrator-Komponente **32** aufgedeckte Schnittstelle IXMC_EnumDriver zurückgereicht. Die Benutzung der Schnittstelle führt dazu, dass die folgenden Schritte stattfinden.

[0232] Zuerst fragt die Bewegungssteuerungs-Komponente **35** die Treiber-Administrator-Komponente **32** nach dem nächsten Treiber. Dann weist die Treiber-Administrator-Komponente **35** das CDriverAdminDisp an, den nächsten unterstützten Treiber zu holen. Das CDriverAdminDisp weist den CModuleMgr an, den nächsten Treiber zu holen. Danach weist der CModuleMgr das CSimpleDriver an, entweder die CLSID oder einen Zeiger an die IUnknown-Schnittstelle für den Treiber zurückzureichen, abhängig von Parametern der Nummerierung. Wenn das CSimpleDriver aufgefordert wird, einen Zeiger an die IUnknown-Schnittstelle zurück zu reichen, wird die Schnittstelle von der Treiber-Komponente abgefragt.

[0233] Ein weiterer Satz von Informationen, die von dem Treiber-Administrator **32** abgefragt werden können, besteht aus dem Treiber-Informations-Abbild. Diese Daten werden von dem Treiber-Administrator-CPL **38** benutzt, wenn Informationen angezeigt werden, die die in dem System registrierten Treiber und Streams beschreiben. Wie in **Fig. 48** dargestellt ist, finden die folgenden Schritte statt, wenn das System nach dem Treiber-Schnittstellen-Abbild abgefragt wird. Wenn sie abgefragt wird, weist die Treiber-Administrator-Komponente das CDriverAdminDisp an, eine CDriverInfoMap-Klasse zu erstellen und zu laden. Das CDriverAdminDisp erstellt die CDriverInfoMap. Als nächstes übermittelt das CDriverAdminDisp die CDriverInfoMap an den CModuleMgr und weist diesen an, das Informations-Abbild zu laden. Der CModuelMgr fragt jeden registrierten Treiber nach seiner internen Information. Jedes CSimpleDriver kommuniziert direkt mit der Treiber-Komponente und fragt diese nach sämtlichen relevanten Treiberinformationen. Dann fragt der CModuleMgr jeden Treiber nach einer Liste aller Streams, die bei dem Treiber registriert sind. Unter Benutzung der Stream-Nummerierung erstellt jedes CSimpleDriver ein Array (Feldgruppe) von CSimpleStream-Objekten und reicht das Array an den CModuleMgr zurück.

[0234] Für jedes CSimpleStream-Objekt in jedem Array fragt der CModuleMgr nach sämtlichen relevanten Informationen. Jedes CSimpleStream kommuniziert direkt mit der Stream-Komponente und fragt sie nach sämtlichen Informationen, die den Stream beschreiben.

[0235] Nachdem das Treiber-Administrator-CPL **38** oder die Bewegungssteuerungs-Komponente **35** mit der Benutzung des Treiber-Administrators **32** fertig sind, müssen sie die Komponente **35** freigeben, um sämtliche von ihr benutzten Betriebsmittel zu löschen. **Fig. 49** beschreibt diesen Prozess. Beim Reinigen nach einem Aufruf an die Release-Methode finden die folgenden Schritte statt.

[0236] Zuerst müssen entweder das Treiber-Administrator-CPL **38** oder die Bewegungssteuerungs-Komponente **35** die Treiber-Administrator-Komponente **32** anweisen, sich selbst freizugeben, indem sie deren Release-Methode aufrufen. Dann weist die Treiber-Administrator-Komponente das Objekt CDriverAdminDisp an, sämtliche in dem System verwendeten Betriebsmittel zu löschen. Das CDriverAdminDisp weist dann den CModuleMgr an, jegliche von ihm verwendeten Betriebsmittel zu löschen. Zuerst verfolgt der CModuleMgr alle CSimpleDriver-Ojekte, deren jedes auf seine SCLSID und seinen Freigabezustand abgefragt wird. Danach wird jedes CSimpleDriver gelöscht. Jedes gelöschte CSimpleDriver-Objekt löscht alle bei sich registrierten Arrays von CSimpleStream-Objekten. Jedes CSimpleStream-Objekt gibt, wenn es gelöscht ist, alle Schnittstellen frei, die es von der Stream-Komponente benutzt hat Bei seiner endgültigen Reinigung gibt jedes CSimpleStream alle Schnittstellen frei, die es von der Treiber-Komponente benutzt hat. Die gesamten CLSID- und Freigabezustands-Informationen werden dauerhaft in der Registrierungsdatenbank gespeichert.

[0237] Fig. 50 zeigt ein Schnittstellen-Abbild für den Treiber-Administrator **32**.

VII. Treiber-Administrator-CPL-Applet

[0238] Dieses Dokument beschreibt die Planung des Control Panel Applet (kleines Anwendungsprogramm in der Dialogbox) des Treiber-Administrators (CPL) **38**, das von dem Benutzer verwendet wird, um sowohl Treiber **30** als auch Stream-Komponenten **28** hinzuzufügen, zu konfigurieren und zu entfernen, die später von der Komponente **35** benutzt werden, wenn diese angewiesen wird, Bewegungssteuerungsoperationen durchzuführen. Hinsichtlich der Planung existieren drei wesentliche "Blickwinkel", aus denen betrachtet wird, wie das Control Panel Applet arbeitet bzw. funktioniert.

[0239] Erstens zeigt ein in Fig. dargestelltes Modul-Interaktions-Abbild alle wesentlichen ausführbaren und benutzer-interaktionsfähigen Items oder Module, die das CPL benutzt und mit denen es interagiert. Wenn zum Beispiel ein Dialog von dem CPL als ausführbar angezeigt wird, gelten sowohl das Dialog- als auch das CPL-Modul als miteinander interagierend. Technisch gesehen ist der Dialog kein Modul, da er ein auf dem Bildschirm angezeigtes Phantasiegebilde ist, aber dennoch werden sie von Modul-Interaktions-Abbildern als solche klassifiziert, da sie Schlüsselbestimmungspunkte für die Benutzereingabe sind.

[0240] Zweitens zeigt ein in Fig. 52 dargestelltes Objekt-Interaktions-Abbild alle wesentlichen Objekte, aus denen die in dem Modul-Interaktions-Abbild beschriebenen Module gebildet sind. Objekte bestehen aus den tatsächlichen Instanzen von C++ Klassen, die jedes Objekt definieren. Alle Interaktionen zwischen den Objekten sind in diesem Interaktions-Abbild ausführlich gezeichnet.

[0241] Schließlich zeigen die Fig. 53 bis Fig. 57 ausführlich gezeichnete Szenario-Abbilder, für welche das Objekt-Interaktions-Abbild als Basis dient. Szenario-Interaktions-Abbilder beschreiben die Interaktionen, die während einer bestimmten Operation stattfinden. Initialisierung, Hinzufügen eines Treibers in dem System und Ansehen der von einem Treiber angebotenen Unterstützung sind alles Beispiele eines Szenario-Interaktions-Abbilds.

[0242] Die Planungsziele für den Treiber-Administrator **32** sind die folgenden:

1. Benutzer-Schnittstellen-Trennung – Implementieren aller für die Steuerung der Treiber-Administrator-Komponente **32** verwendeten Benutzer-Schnittstellen-Elemente.
2. Aufrüstbar auf den OCX-Client – Eventuell kann jeder Treiber und Stream alle UI-Elemente mit einem OCX implementieren, der dann alle Eingaben an den entsprechenden Treiber oder Stream übermittelt. Das Treiber-Administrator-CPL **38** muss so konzipiert sein, dass es einfach zu einem OCX-Client aufzurüsten ist.
3. Für Stream-Unabhängigkeit sorgen – Treiber **30** sollten nicht Streams **28** benutzen müssen, um arbeiten zu können. Die Planung des Treiber-Administrators **32** muss sicherstellen, dass er nicht von Operationen der Stream-Komponenten **28** abhängig ist, um zu arbeiten.
4. Windows 95 UI benutzen – Wann immer es möglich ist, sollten Windows 95 UI-Elemente benutzt werden. Zum Beispiel TreeViews (Ansichten der Baumstruktur), ImageLists (Bildlisten), ButtonBars (Schaltflächenleisten), Tab Dialogs (Benutzeroberflächen-Dialoge) und andere UI-Elemente (Benutzerschnittstellen-Elemente) sollten in Benutzung genommen werden, um ein Windows 95-Erscheinungsbild und -Gefühl sicherzustellen.

[0243] Die folgende Erläuterung beschreibt das Modul-Interaktions-Abbild für das Control Panel Applet **38**. Ein Modul ist entweder als eine ausführbare binäre, eine externe Datei oder ein Hauptbenutzer-Schnittstellenelement definiert, das im Dialog mit dem Benutzer verwendet wird. Fig. 51 ist eine Zeichnung sämtlicher Module, die miteinander interagieren, wenn das Control Panel Applet des Treiber-Administrators läuft.

[0244] Das Treiber-Administrator-CPL **38** ist ein Dialogbox-Applet Und ein Dialogbox-Applet ist eine spezielle DLL (Dynamische Bibliothek von Programmverbindungen), die verschiedene Funktionen exportiert, die es dem Windows Control Panel erlauben, mit dem Applet, d.h. dem kleinen Anwendungsprogramm zu kommunizieren.

[0245] Der Treiber-Administrator-Dialog ist der Hauptdialog, der erscheint, wenn die Ikone des Control Panel Applet in dem Windows Control Panel angewählt wird. Der Browse-Dialog (Anzeige-Dialog) wird benutzt, um den Benutzer nach einem Dateinamen zu fragen. Wenn zum Beispiel ein neuer Stream oder Treiber hinzugefügt werden, benutzt der Treiber-Administrator diesen Dialog, um den Benutzer nach der Speicherstelle des hinzuzufügenden neuen Treibers oder Stream zu fragen.

[0246] Der View-Support-Dialog zeigt die von dem gewählten Treiber **30** bereitgestellte Unterstützung an. Jeder Treiber kann einen andere Gruppe von erweiterten Funktionen unterstützen. Dieser Dialog zeigt dem Benutzer ganz genau, wie viel Unterstützung von jedem Treiber geleistet wird, und ermöglicht festzustellen, welche Funktionen möglicherweise nicht arbeiten, wenn der Treiber benutzt wird.

[0247] Anders als das oben beschriebene Modul-Interaktions-Abbild beschreibt das in Fig. 52 dargestellte Objekt-Interaktions-Abbild, wie die tatsächlichen Instanzen von C++-Objekten in jedem Modul miteinander interagieren.

[0248] Abgesehen von der Anzeige, dass jeder Dialog von dem Objekt verwaltet wird, dessen Name in dem Dialog angezeigt wird, sind der wesentliche Unterschied zu dem Modul-Interaktions-Abbild die C++-Objekte CComCPL und CDriverAdmin. Beide Objekte werden nachstehend beschrieben.

[0249] Da die Beschreibung einer jeden Dialogklasse relativ deutlich ist und der obenstehenden Dialogbeschreibung sehr ähnlich, werden diese Dialogklassen in diesem Abschnitt nicht beschrieben. Dieser Abschnitt beschreibt all anderen C++-Objekte.

[0250] Das Objekt CComCPL ist ein C++-Objekt, das von der COMBuilder-Anwendung aus einem Template (Schablone/Vorlage) generiert wird. Es wird für das Handling sämtlicher Windows Mitteilungen verwendet, die von der Control Panel-Anwendung gesendet werden.

[0251] Das Objekt CDriverAdmin wird zum Ansteuern, Steuern/Kontrollieren und Verwalten der Benutzung der Treiber-Administrator-Komponente **32** verwendet. Zum Beispiel werden die gesamte OLE 2.0-Schnittstellenverwaltung und die Datenübersetzung von diesem Objekt gehandhabt. Die Datenübersetzung beinhaltet die Übersetzung von Daten von einem C++-Standardformat in ein Rohformat, das mit dem OLE 2.0-Datenübertragungsmechanismus problemlos gehandhabt wird.

[0252] Szenario-Interaktions-Abbilder sind annähernd identisch mit Objekt-Interaktions-Abbildern, aber sie zeigen nur Objekte und Interaktionen, die an einer bestimmten Operation beteiligt sind. Die jeweiligen Interaktionen sind in der Reihenfolge, in der sie während der laufenden Operation stattfinden, nummeriert. Im Folgenden werden verschiedene Schlüsseloperationen beschrieben, die stattfinden, während man das Treiber-Administrator-CPL-Applet **38** laufen lässt.

[0253] Die Initialisierung findet statt, wenn der Benutzer das CPL-Applet zum ersten Mal laufen lässt. Während dieses Prozesses werden alle anderen Objekte initialisiert und verschiedene Module geladen. Zwei Schritte finden während des Initialisierungsprozesses statt. Erstens wird die Anmeldung initialisiert, und zweites wird der Dialog mit Werten initialisiert, die aus der Treiber-Administrator-Komponente **32** abgefragt werden. Die folgenden Abschnitte beschreiben jeden dieser Schritte.

[0254] Die Initialisierung der Anwendung, die in Fig. 53 gezeigt ist, findet statt, wenn man die Anwendung zum ersten Mal laufen lässt und der Hauptdialog noch nicht angezeigt wurde. Bei der Initialisierung finden folgende Schritte statt.

[0255] Durch eine Windows-Mitteilung informiert Windows das Objekt CComCPL, dass das Control Panel Applet eben geladen wurde. Das CComCPL lädt dann das CDriverAdminDialog und teilt diesem mit, vor dem Übergang zu den Moden die Dialogvorbereitung zu machen. Als nächstes lädt das CDriverAdminDialog sämtliche in der Registrierungsdatenbank gespeicherten Einstellungen. Zum Beispiel können die Position des momentanen Fensters und der aktiven Schaltfläche in der Datenbank gespeichert sein.

[0256] Das CDriverAdminDialog lädt dann die CDriverAdmin-Klasse und weist diese an, sich selbst zu initialisieren. Während der Initialisierung erstellt das CDriverAdminDialog ein Exemplar des Treiber-Administrators **32** und fragt alle Schnittstellen ab, die benutzt werden.

[0257] Ist die Anwendung erst initialisiert, müssen die in dem Dialog anzuzeigenden Standardeinstellungen (default settings) eingestellt werden. Diese Werte werden eingestellt, wenn der Dialog unmittelbar vor seiner Anzeige initialisiert wird. Fig. 54 beschreibt diesen Prozess. Während des Prozesses der Initialisierung des Dialogs finden die folgenden Schritte statt.

[0258] Während der Dialogvorbereitung, die vor dem Aufruf DoModal stattfindet, fragt das CDriverDialog das Objekt CDriverAdmin nach der Treibernummerierung, die zu verwenden ist, wenn die anzuzeigenden Anfangs-

werte in dem Dialogfenster eingestellt werden. Das CDriverAdmin benutzt die Treiber-Administrator-Komponente **32**, um das Treiber-Informations-Abbild abzufragen, das dann an das CDriverAdminDialog zurückgegeben wird. Hat das CDriverAdminDialog das Treiber-Informations-Abbild einmal empfangen, benutzt es die Informationen, um alle Schnittstellenelemente, die sich entweder auf Treiber oder Streams beziehen, zu aktualisieren.

[0259] Das Hinzufügen eines Treibers in dem System **22** kann in zwei Stufen aufgeschlossen werden. Zuerst muss der Modulname in dem System hinzugefügt werden. Dann muss der Hauptdialog des Treiber-Administrators **32** sich selbst aktualisieren, um den soeben hinzugefügten neuen Treiber wiederzugeben.

[0260] Ein neuer Treiber wird hinzugefügt, wenn der Benutzer die Schaltfläche "Add ..." (Hinzufügen) in dem Hauptdialog des Treiber-Administrators **32** aktiviert wird. Fig. 55 beschreibt diesen Prozess. Wenn ein neuer Treiber hinzugefügt wird, finden folgende Schritte statt.

[0261] Fügt der Benutzer einen neuen Treiber hinzu, muss er erst die Schaltfläche "Add ..." aktivieren. Das CCDriverAdminDialog öffnet den allgemeinen Dialog der offenen Datei. Der Benutzer muss den Dateinamen des hinzuzufügenden Treibers eingeben und den Dialog schließen. Das CDriverAdminDialog übermittelt den Dateinamen dann an das Objekt CDriverAdmin und ruft die Methode RegisterDriver auf, indem es den Namen des als Treiber zu registrierenden Moduls eingibt Das CDriverAdmin reicht den Treiberdateinamen dann an die Treiber-Administrator-Komponente **32** weiter und weist diese an, den Treiber in dem System **22** zu registrieren.

[0262] Der Prozess des Aktualisierens des Hauptdialogs ist identisch mit dem Prozess des Initialisierens des oben beschriebenen Dialogs.

[0263] Ähnlich wie der Prozess des Hinzufügens eines Treibers beinhaltet das Entfernen eines Treibers sowohl das Entfernen des Treibers aus dem System als auch die anschließende Aktualisierung des Hauptdialogs.

[0264] Aktiviert man die Schaltfläche "Remove" (Entfernen), wird ein Treiber aus dem XMC-Softwaresystem entfernt. Fig. 56 beschreibt diesen Prozess. Die folgenden Schritte finden statt, wenn ein Treiber entfernt wird.

[0265] Um einen Treiber zu entfernen, muss der Benutzer zunächst die Schallfläche "Remove" wählen. Hat er die Schaltfläche angeklickt, wird der gewählte oder der Elterntreiber für den gewählten Stream entfernt. Das CDriverAdminDialog übermittelt XMC_HDRIVER des Treibers an das CDriverAdmin und weist dieses an, den Treiber durch Aufrufen seiner Methode UnRegister zu entfernen. Das CDriverAdmin übergibt XMC_HDRIVER an die Treiber-Administrator-Komponente **32** und weist diese an, den Treiber abzumelden (UnRegister).

[0266] Der Prozess des Aktualisierens des Hauptdialogs ist identisch mit dem Prozess des Initialisierens des oben erläuterten Dialogs.

[0267] Viewing Support beinhaltet die Einsicht in die Stufe der Unterstützung durch den gewählten Treiber. Fig. 57 beschreibt den Prozess der Bereitstellung dieser Informationen für den Benutzer über den View Support Dialog. Beim Einsehen der durch den Treiber bereitgestellten Unterstützung finden die folgenden Schritte statt.

[0268] Zuerst muss der Benutzer die Schaltfläche "View Support" (Unterstützung ansehen) im Hauptdialogfeld des Treiber-Administrators anklicken. Wenn das CDriverAdminDialog gewählt ist, fragt es das CDriverAdmin nach der Information über die Treiberunterstützung. Das CDriverAdmin leitet die Abfrage weiter an die Treiber-Administrator-Komponente **32**, die die Informationen tatsächlich ausfüllt. Sobald die abgefragte Information zurückgegeben wird, leitet sie das CDriverAdminDialog weiter an das CViewSupportDialog. Das CViewSupportDialog initialisiert sich selbst unter Verwendung der Treiberunterstützungsinformationen.

VIII. Treiber-Administrator CPL Applet

[0269] Dieser Abschnitt enthält eine Beschreibung des Control Panel Applet **38** (Dialogbox-Applet) für den Treiber-Administrator. Wenn der Treiber-Administrator **32** zum Konfigurieren des Bewegungssteuerungssystems benutzt wird, gibt es zwei wesentliche Elemente, mit denen der Benutzer arbeiten wird: Treiber und Streams. Jeder Treiber **30** generiert die hardware-spezifischen Steuercodes, die dann zu der gewählten Stream-Komponente **28** gesandt werden. Streams ermöglichen die Datentransportschicht zwischen dem Treiber und dem Steuercode-Bestimmungsort.

[0270] Abhängig von der momentanen Hardware-Konfiguration können verschiedene Streams verwendet werden. Wenn zum Beispiel die Hardware mit dem PC-Bus verbunden ist, wird man für die Kommunikation mit diesem einen PC-Bus-Stream benutzen. Ist die Hardware dagegen durch ein serielles Kabel mit einer seriellen E/A-Anschlussstelle verbunden, wird man den seriellen Stream benutzen. Schließlich können alle Hardware-Konfigurationen den File Stream (Datei-Stream) benutzen. Wenn der File Stream benutzt wird, werden alle Steuercodes zu der angegebenen Datei gesandt, die später auf die Hardware heruntergeladen werden kann.

[0271] Dieser Abschnitt beschreibt sowohl Treiber als auch Streams und wie diese jeweils konfiguriert sind. Zu Beginn beschreibt dieser Abschnitt die Treiberelemente und alle Property Pages (Eigenschaftsseiten), die für ihre Editierung verwendet werden. Schließlich beschreibt dieser Abschnitt noch die About Box, die Details über die Software enthält.

[0272] Der Hauptzweck eines jeden Treibers ist das Generieren der hardware-spezifischen Steuercodes, die die Hardware anweisen, die speziellen Bewegungssteuerungsaktionen durchzuführen. Solche Aktionen können zum Beispiel das Abfragen der Hardware nach momentanen Positionen oder die Anweisung der Hardware zur Bewegung zu einer vorgegebenen Stelle in dem System enthalten. Der folgende Beitrag beschreibt die Property Pages, die zum Konfigurieren jedes Treibers verwendet werden.

[0273] Es gibt zwei Arten von Properties (Eigenschaften), die jeden Treiber beeinflussen. Zuerst kann eine Gruppe von Standardvorgaben eingestellt werden, die von der Bewegungssteuerungs-Komponente **35** als empfohlene Werte verwendet werden. Die verwendete Skalierung und die verwendeten Einheiten sind einige Beispiel-Vorgabewerte. Wenn der Treiber eine weiter verbesserte Konfiguration unterstützt, wird durch das Anklicken der Schaltfläche Advanced ... ein Dialogfenster geöffnet, das zur Einstellung der Treiberkonfiguration verwendet wird. Wenn ein Treiber beispielsweise keine Streams unterstützt, ermöglicht der von dem Treiber zur Verfügung gestellte Dialog für die erweiterte Konfiguration dem Benutzer die Einstellung der E/A-Anschlussstelle und die IRQ-Einstellungen (Interrupt Request = Einstellungen für die Anforderung der Unterbrechung).

[0274] Die die Treiber **30** beeinflussenden Properties sind wie folgt.

[0275] Skalieren – Das Einstellen der Skaliereigenschaften beeinflusst die Vorgabenskalierung, die an allen Achsen in dem Bewegungssteuerungssystem verwendet wird. Der Bereich für die Skalierung von Werten ist (0,0, 1,0]. Die Vorgabeneinstellung kann beim Programmieren von XMC überholt werden, indem die Schnittstelle IXMC_StaticState benutzt wird.

[0276] Einheiten – Das Einstellen der Einheiten-Property beeinflusst sämtliche Koordinaten, die beim Programmieren des Systems **22** verwendet werden.

[0277] Die Einheitenbeschreibungen sind wie folgt:
MM_ENGLISH – Inches werden als Basiseinheit für alle Koordinaten verwendet.
MM_METRIC – Millimeter werden als Basiseinheit für alle Koordinaten verwendet.
MM_NATIVE – Die durch das Hardwaresystem definierten systemeigenen Koordinaten werden verwendet.
Für die XMG-Programmierung verwendete Koordinaten werden 1:1 auf die Hardware-Koordinaten abgebildet.
Advanced ... – Die Aktivierung dieser Schaltfläche öffnet einen Dialog, der benutzt wird, um erweiterte Eigenschaften für den Treiber zu edieren, der von dem Benutzer ediert werden kann.

[0278] Außer dass sie dem Benutzer die Einstellung von Eigenschaften ermöglicht, zeigt jede Treiber-Property Page auch die vollständigen Namen sowohl der unterstützten Hardware als auch des Hardware-Herstellers an.

[0279] Die Schaltflächen entlang der unteren Leiste des Fensters arbeiten mit dem gewählten Treiber oder Stream.

[0280] Die folgende Erläuterung beschreibt jede Schaltfläche und was sie macht.

[0281] Das Aktivieren der Schaltfläche Make Default wählt den momentanen Treiber als Vorgabe. Wenn ein Stream gewählt wird, wird sein Elterntreiber zu dem Vorgabe-Treiber. Der Vorgabe-Treiber wird später von der Bewegungssteuerungs-Komponente **35** benutzt.

[0282] Das Aktivieren der Schaltfläche Add ... zeigt den Add Module-Dialog. Dieser Dialog wird verwendet, um in dem System **22** neue Treiber und Streams hinzuzufügen. Einmal gewählt, werden der neue Treiber oder der neue Stream in der Ansicht des Treiberbaums dargestellt. Wenn ein Stream hinzugefügt wird, so geschieht dies unter dem momentan gewählten Treiber. Um den Stream freizugeben, muss man das Freigabe-Kontrollfenster auf der Property Page der Streams anklicken.

[0283] Die Wahl der Schaltfläche Remove entfernt den gegenwärtig gewählten Treiber oder Stream. Wird ein Treiber entfernt, so werden auch alle seine Streams entfernt.

[0284] Die Wahl der Schaltfläche View Support ... öffnet einen Dialog, der verwendet wird, um die Ebene der von dem Treiber implementierten XMC-Unterstützung ansehen zu können. Zum Beispiel werden alle API-Schnittstellen und die nachfolgenden Methoden angezeigt Wenn eine fehlende Implementierung in dem Treiber eine API-Schnittstelle am Arbeiten hindert, wird die Treiber-Stubroutine **36** benutzt. Wenn die fehlende Implementierung in dem Treiber **30** nicht durch Operationen in der Treiber-Stubroutine **36** ersetzt werden kann, wird die Schnittstelle oder Methode gesperrt.

[0285] Das Folgende sind Beschreibungen jeder in dem XMC Support View-Dialog vorgefundenen Grafiken.
D – Diese Grafik bedeutet, dass die Schnittstelle oder Methode von dem Treiber **30** implementiert wird.
S – Diese Grafik bedeutet, dass die Schnittstelle oder Methode innerhalb der Treiber-Stubroutine **36** implementiert wird.
X – Diese Grafik bedeutet, dass die Schnittstelle oder Methode wegen mangelnder Implementierung in dem Treiber **30** gesperrt wird.

[0286] Wie die Properties Page ist auch eine Debug Page (Fehlerbeseitigungsseite) vorgesehen, um alle Fehlerbeseitigungseinstellungen für den Treiber vorzunehmen. Jeder Treiber kann angeben, dass alle zur Steuerung des Treibers verwendeten API-Aufrufe protokolliert werden. Die Protokollierungseinstellungen betreffen nur den gegenwärtig gewählten Treiber. Das Output Field (Ausgabefeld) ermöglicht die Wahl des Ausgabe-Stream, auf dem alle Debug-Informationen gesendet werden. Wenn Streams freigegeben ist, wird die Debug-Information an die angegebene Textdatei gesandt. Wenn der Debug-Monitor aktiviert wird, werden die Debug-Informationen zu dem Debug-Monitor gesandt, falls dieser in Betrieb ist. Verwendet man Enable, um einen Stream freizugeben, wird dieser dadurch aktiviert und veranlasst, dass alle erzeugten Debug-Informationen an den Stream gesandt werden. Es können mehr als ein Stream auf einmal aktiviert werden.

[0287] Die Stream-Einstellungen sind für jeden unterstützten Debug-Stream verfügbar. Text File erlaubt die Einstellung des Namens der Textdatei. Der Debug-Monitor kann lediglich aktiviert und deaktiviert werden.

[0288] Ein Stream ist die Transportschicht, die von dem Treiber benutzt wird, um Daten zum Zielort zu übermitteln. Der Zielort kann die eigentliche Bewegungssteuerungs-Hardware oder sogar eine Textdatei sein. Normalerweise wird die von dem Hardware-Hersteller benutzte Steuersprache von mehreren unterschiedlichen Arten ihrer Bewegungssteuerungs-Hardware unterstützt. Zum Beispiel haben manche Hersteller Bewegungssteuerungs-Hardware, die sowohl auf einem PC-Bus als auch auf einem seriellen E/A basiert, die dieselbe Steuersprache verstehen. In einem solchen Fall würde derselbe Treiber für jede Hardware-Konfiguration verwendet werden, würde aber abhängig von der speziellen Hardware-Konfiguration mit verschiedenen Streams kommunizieren. Grafisch ist jeder Stream unter jedem Treiber gelistet, der den Stream benutzt.

[0289] Dieser Abschnitt beschreibt die von dem System **22** unterstützten Streams und wie sie konfiguriert sind.

[0290] Der PC-Bus-Stream sendet alle Daten direkt an ein Bewegungssteuerungs-Hardwaresystem, das auf einem PC-Bus basiert, durch Einlesen in die angegebenen E/A-Anschlussstellen und die durch die Hardware definierten IRQs. Dieser Abschnitt beschreibt sowohl die Properties als auch Debug-Einstellungen, die für den PC-Bus-Stream verfügbar sind.

[0291] Stream Properties betreffen nur den gegenwärtig gewählten Stream. Der Benutzer muss bestimmte Einstellungen wählen, wie zum Beispiel die E/A-Anschlusstelle und die IRQ (Anforderung der Unterbrechung). Ohne eine Einstellung dieser Werte kann der PC-Bus-Stream nicht mit der Hardware kommunizieren. Die die PC-Bus-Streams betreffenden Properties werden nachfolgend beschrieben.

[0292] Die E/A-Anschlussstelle ist der Basis-Anschlussstelle, die für die Kommunikation mit derjenigen Bewegungssteuerungs-Hardware benutzt wird, an die der Stream Daten senden soll.

[0293] Die IRQ ist die Unterbrechungsanforderungsebene, die von der Hardware benutzt wird.

[0294] Aktiviert man die Schaltfläche Advanced ..., öffnet sich ein Dialog, der dem Benutzer die Editierung mehrerer fortgeschrittener Stream-Optionen erlaubt. Wenn zum Beispiel der Stream ein E/A-Anschlussstellen-Abbild unterstützt, das der Benutzer edieren kann, würde das Anschlussstellen-Abbild in diesem Dialog dargestellt werden. Diese Schaltfläche wird nur für Streams aktiviert, die die erweiterten Merkmale unterstützen, die der Benutzer edieren kann.

[0295] Beim Austesten (Debugging) eines Anwendungsprogramms kann es nützlich sein zu sehen, welche Codes tatsächlich an die Hardeware gesandt werden. Die Debug Settings Page (Seite mit Fehlersucheinstellungen) für Streams erlaubt dem Benutzer die Aktivierung sowohl der Cmd- als auch der Bit-Streams. Die Cmd-Streams werden benutzt, um alle an die Hardware gesandten Befehlscodes zu protokollieren. Wenn man auf dieser Ebene nicht genügend Detailinformationen erhält, kann man den Bit-Stream benutzen. Wenn der Bit-Stream aktiviert ist, protokolliert er alle Werte, durch jede Hardware-Anschlussstelle gesendet werden. Alle Werte, die aus jeder von der Hardware benutzten Anschlussstelle ausgelesen oder in diese eingelesen werden, werden protokolliert. Hier ist anzumerken, dass beide Streams, wenn sie freigegeben sind, die das Bewegungssteuerungssystem programmierende Anwendung bedeutend verlangsamen können.

[0296] Serial RS-232-Streams werden verwendet, um Daten von dem Treiber zu einer Bewegungssteuerungs-Hardware zu senden, die über die serielle E/A-Anschlussstelle mit dem Computer verbunden ist. Sowohl Property- als auch Debug-Einstellungen betreffen den gewählten Serial RS-232-Stream. Der folgende Beitrag beschreibt die jeweils verfügbaren Einstellungen im Detail.

[0297] Alle Serial RS-232 Property-Einstellungen müssen von dem Benutzer vorgenommen werden, da sie dem Stream mitteilen, welche E/A-Anschlussstelle und welches Kommunikationsprotokoll bei der Kommunikation mit der Hardware zu benutzen sind. Die die Serial RS-232-Streams betreffenden Properties werden nachfolgend beschrieben.

[0298] Die Anschlussstelle ist die serielle Anschlussstelle, mit der die Hardware verbunden ist. COM1 bis COM4 sind gültige Anschlussstellen (Ports), die benutzt werden können.

[0299] Die Baudrate ist die Geschwindigkeit der von der Hardware unterstützten Datenübertragung.

[0300] Wenn Hardware gewählt wird, die zwar effizienter, aber schlechter kompatibel ist, wird für die Kommunikation mit der Hardware ein Kommunikationsprotokoll verwendet. Falls Fehler auftreten, wenn dieses Protokoll gewählt wird, ist das XON/XOFF-Kommunikationsprotokoll zu benutzen.

[0301] Wenn das XON/XOFF-Kommunikationsprotokoll gewählt wird, wird ein einfacheres und besser kompatibles Kommunikationsprotokoll benutzt.

[0302] Debug-Einstellungen für den Serial RS-232-Stream sind jenen sehr ähnlich, die von dem PC-Bus-Stream unterstützt werden. Serial RS-232-Streams unterstützen nur die Befehls-Aufzeichnung durch den Cmd-Stream, aber sie unterstützen keine Bit-Aufzeichnung.

[0303] Der Text File Stream (Textdatei-Stream) wird benutzt, um Steuercode-Programme für die spätere Verwendung zu erstellen. Die Benutzung dieses Stream erleichtert das Laufen der XMC-Software im Code-Generierungs-Modus. In diesem Modus finden keine Bewegungssteuerungsaktionen statt. Stattdessen können Steuercode-Programme erstellt und in Dateien gespeichert werden. Später, nachdem die Programme erstellt und gesichert sind, können sie in die Bewegungssteuerungs-Hardware heruntergeladen werden und auf dieser laufen. Der folgende Beitrag beschreibt die Property- und Debug-Einstellungen für den Text File Stream.

[0304] Die wesentliche Eigenschaft, die beim Konfigurieren eines Text File Stream eingestellt wird, sind der tatsächliche Name und die Speicherstelle der zu verwendenden Datei. Ist diese Einstellung einmal erfolgt, ist der Stream bereit für die Benutzung.

[0305] Die folgenden Eigenschaften können für den Text File Stream konfiguriert werden:
Filename ist der Dateiname und die Speicherstelle der Datei, die zum Speichern aller von dem gewählten Treiber **30** generierten Steuercodes verwendet wird. Das Aktivieren der Schaltfläche Browse ... öffnet einen Dialog, der eine grafische Wahl der Speicherstelle und des zu benutzenden Dateinamens erlaubt.

[0306] Es sind keine Debug-Einstellungen für den Text File Stream verfügbar.

IX. Sprachtreiber

[0307] Fig. 58 enthält ein Modul-Interaktions-Abbild, das einen Sprachtreiber **44** zeigt und darstellt, wie der Sprachtreiber **44** mit den Streams **28**, dem Treiber-Administrator **32**, der Bewegungssteuerungs-Komponente **35** und der Registrierungsdatenbank **42** interagiert.

[0308] Wie bei den vorstehend beschriebenen Software-Treibern **30** wird ein Sprachtreiber **44** pro Bewegungssteuerungsvorrichtung **20** der Gruppe von unterstützten Bewegungssteuerungsvorrichtungen verwendet. Die Sprachtreiber **44** führen die gleichen Basisfunktionen aus wie die vorstehend unter Bezugnahme auf die **Fig.** 1 und Fig. 12–Fig. 21 beschriebenen Software-Treiber **30**. Die Sprachtreiber **44** erhalten Zugriff auf das Softwaresystem **22** und sie reagieren in der gleichen Weise wie die Software-Treiber **30**; der Unterschied zwischen den Sprachtreibern **44** und den Software-Treibern **30** ist gänzlich intern.

[0309] Der primäre Unterschied zwischen diesen Treibern **30** und **44** ist, dass die Sprachtreiber **44** eine Datenbank benutzten, deren Schlüsselfelder ein Indexfeld, ein Befehlsformatfeld und ein Antwortformatfeld sind. Jeder Eintrag oder jede Reihe in der Datenbank entspricht einer gegebenen Treiberfunktion.

[0310] Der Zweck der Befehls- und Antwortformatvorlagen (Templates) ist die Formalisierung und Vereinfachung des Prozesses zur Bildung von Befehlsdaten-Strings (Befehlsdatenketten) und Formatdaten-Strings (Formatdatenketten), die Befehle und Parameter enthalten, die zu den Bewegungssteuerungsvorrichtungen **20** übertragen werden müssen. Die Formatvorlagen definieren, wie das Softwaresystem **22** bei einem gegebenen SPI-Befehl mit einer herstellerspezifischen Hardware-Befehlssprache kommuniziert, die einer gegebenen Bewegungssteuerungsvorrichtung **20** zugeordnet ist, welche wiederum einem Sprachtreiber **44** zugeordnet ist. Demzufolge wird für jede solche Sprache eine Datenbank, die Befehlsformatvorlagen und Antwortformatvorlagen enthält, erstellt.

[0311] Das Befehlsformatfeld enthält eine Befehlsformatvorlage, und das Antwortformatfeld enthält einen Antwortformatvorlage. Jede dieser Vorlagen (Templates) umfasst eine Sequenz von Datentyp-Bezeichnern, Makro-Bezeichnern, Syntaxzeichen und/oder ASCII-Zeichen.

[0312] Das Indexfeld enthält einen Wert, der einmalig ist für jede Treiberfunktion, die den Vorgang des Verweises auf die einer gegebenen Treiberfunktion zugeordneten Befehls- und Antwortformatvorlagen ermöglicht.

[0313] Der Softwaresystem-Planer definiert die vorgenannten Datentyp-Bezeichner, Makro-Bezeichner und Syntaxzeichen. Im Allgemeinen haben die Befehlsformatvorlage und auch die Antwortformatvorlage gemeinsame Datentyp-Bezeichner und Syntaxzeichen.

[0314] Die Makro-Bezeichner entsprechen normalerweise den Makros, die entweder der Befehlsformatvorlage oder der Antwortformatvorlage zugeordnet sind. Die ASCII-Zeichen werden durch die Treiberfunktion und die spezielle Bewegungssteuerungsvorrichtung **20** definiert, der ein gegebener Sprachtreiber **44** zugeordnet ist.

[0315] Ein Excel-Tabellenblatt (Spreadsheet) kann als organisatorisches Tool (Werkzeug) verwendet werden, das die Erstellung der von dem Sprachtreiber **44** benutzten Datenbank erleichtert. Ein Beispiel eines für diesen Zweck verwendbaren Excel-Tabellenblatts ist in Fig. 67 dargestellt. Das in Fig. 67 dargestellte Tabellenblatt wird als tabdelimited file (Datei mit Tabulator als Delimiter) gesichert und dann in die in Fig. 59 gezeigte SPI-Datenbank kopiert.

[0316] Das in Fig. 67 gezeigte Tabellenblatt ist einfach ein organisatorisches Tool und wird hier nicht näher erläutert. Es sei jedoch erwähnt, dass das beispielhafte Tabellenblatt in Fig. 67 eine Liste von typischen Bezeichnern von Befehls- und Antwortdatentypen zusammen mit deren Beschreibungen und Listen von Befehls- und Antwortmakros angibt. Diese werden durch einen zusätzlichen STRUCT-Datentyp ergänzt, der es dem Programmierer erlaubt, einen Datentyp zu definieren, der die anderen, für eine gegebene Vorlage notwendigen Grunddaten kombiniert.

[0317] Die Sprachtreiber arbeiten daher im Allgemeinen wie folgt. Wie oben beschrieben, ruft die Bewegungs-Komponente **35** die durch den Sprachtreiber **44** implementierte Treiberfunktion auf und übermittelt in vielen Fällen die für die Durchführung dieser Funktion notwendigen Parameter. Der Sprachtreiber **44** benutzt

den Index für diese Treiberfunktion, um die der geeigneten Treiberfunktion zugeordnete Befehlsformatvorlage und Antwortformatvorlage aufzufinden.

[0318] Indem er die Befehlsformatvorlage benutzt, bildet der Sprachtreiber **44** einen Befehlsdaten-String, der ASCII-Zeichen enthält. Der Befehlsdaten-String trägt die Befehle und Parameter, die notwendig sind, um die gegebene Treiberfunktion in einer gewünschten Weise an der dem Sprachtreiber **44** zugeordneten Bewegungssteuerungsvorrichtung **20** zu implementieren.

[0319] Ähnlich benutzt der Sprachtreiber **44** die Antwortformatvorlage, um einen Antwortdaten-String, der in Reaktion auf den Befehlsdaten-String von der bestimmten Bewegungssteuerungsvorrichtung **20** gesandt wird, syntaktisch zu analysieren (parsen). Die Antwortformatvorlage erlaubt daher dem Sprachtreiber **44**, sämtliche Befehle und/oder Parameter, die notwendig sind, damit die Steuerungsanwendung **26** wie beabsichtigt funktioniert, von der Bewegungssteuerungsvorrichtung **20** an die Bewegungssteuerungs-Komponente **35** zu übermitteln.

[0320] Es folgen Beispiele des Prozesses zum Generieren eines Befehlsdaten-String und zum Parsen eines Antwortdaten-String bei einer gegebenen Gruppe von Befehls- und Antwortformatvorlagen, die einer einzigen SPI zugeordnet ist.

BEISPIEL 1

[0321] Das erste Beispiel zeigt, wie sich der Sprachtreiber **44** mit der Treiberfunktion IXMC_DrvExt_Test::Move befassen könnte.

Befehlsformat:	D%d,+:@[snd]GO%b+:@[snd]
Antwortformat:	@[crlf]>@[rcv]@[crlf]>@[rcv]
Aufruf Treiberfunktion:	pXMCDrvExtTest->Move(20.0, 30.0)

[0322] Diese Funktion weist die Bewegungssteuerungsvorrichtung an, sich 20 Einheiten in der x-Richtung und 30 Einheiten in der y-Richtung zu bewegen.

[0323] Der Treiber kommuniziert mit dem Stream wie folgt:
Schritt 1. Führe die Operation in der Befehlsformatvorlage bis zum ersten @ Symbol durch. Dies bildet einen Rohbefehlsdaten-String von "D20.0, 30.0:".
Schritt 2. Nach dem ersten @ Symbol folgt der Sendebefehl, der den in Schritt 1 gebildeten Datenstring sendet. Der Sprachtreiber hat nun das G in der Befehlsformatvorlage erreicht.
Schritt 3. Nach dem Sendebefehl liest der Sprachtreiber eine Antwort aus dem Stream, um zu bestätigen, dass der Befehlsdaten-String richtig empfangen und verarbeitet wurde. Der von dem Stream empfangene Antwortdaten-String ist wie folgt: "\r\n>".
Schritt 4. Der Sprachtreiber benutzt als nächstes die Antwortformatvorlage, um den Rohantwortdaten-String syntaktisch zu analysieren (parsen), um die Operation zu verifizieren und Daten zu extrahieren. Der Sprachtreiber knüpft dann bei dem G in der Befehlsformatvorlage an und bildet den nächsten Rohbefehlsdaten-String von "GO11" und hört bei dem letzten Makro auf.
Schritt 5. Der Sprachtreiber, der bei dem letzten Makro in der Befehlsformatvorlage anknüpft, sendet dann den in Schritt 4 erstellten Rohbefehlsdaten-String an den Stream, womit die Befehlformatvorlage vollständig ist.
Schritt 6. Nach dem Sendebefehl empfängt der Sprachtreiber von dem Stream erneut einen Antwortdaten-String wie folgt: "\r\n>".
Schritt 7. Der Sprachtreiber analysiert dann den in Schritt 6 empfangenen Antwortdaten-String syntaktisch.

BEISPIEL 2

[0324] Das zweite Beispiel zeigt, wie sich der Sprachtreiber **44** mit der Treiberfunktion IXMC_DrvExt_Test::SetVelocity befassen könnte.

Befehlsformat:	V%lf,+:@[snd]
Antwortformat:	[crlf]>@[rcv]
Aufruf Treiberfunktion:	pXMCDrvExtTest->SetVelocity(NOP, 22.0)
Erklärung:	Setze Geschwindigkeit der y-Achse auf 22.0.
Rohbefehls-String:	"V,22.0:"
Rohantwort-String:	"\r\n>" (erwartet)

BEISPIEL 3

[0325] Das dritte Beispiel zeigt, wie sich der Sprachtreiber **44** mit der Treiberfunktion IXMC_DrvExt_Test::GetVelocity befassen könnte.

Befehlsformat:	GV%b+:@[snd]
Antwortformat:	%d,+@[crlf]>@[rcv]
Aufruf Treiberfunktion:	pXMCDrvExtTest->GetVelocity(NOP, &dfY_Vel)
Erklärung:	Hole für y-Achse eingestellte Geschwindigkeit
Rohbefehls-String:	"GV01:"
RohantwortString:	",44.0\r\n>" (erwartet)
DfY_Vel = 44.0	

BEISPIEL 4

[0326] Das vierte Beispiel zeigt, wie sich der Sprachtreiber **44** mit der Treiberfunktion IXMC_drvExt_Test:::Reset befassen könnte.

Befehlsformat	!RESET:@[snd]MA0:MC0:LH0@[snd]
Antwortformat:	@[crlf]*VENDOR NAME – MODELL@[rcv}@[crlf]@[rcv]
Aufruf Treiberfunktion:	pXMCDrvExtTest->Reset()
Erklärung:	Setze die Hardware zurück.
Rohbefehls-String 1:	"!RESET:"
Rohantwort-String 1:	"\r\n*VENDOR NAME – MODEL" (erwartet)
Rohbefehls-String 2:	"MA0:MC0:LH0:"
Rohantwort-String 2:	"\r\n>" (erwartet)

[0327] Während der Sprachtreiber **44** im Kontext mit dem vorstehend beschriebenen Softwaresystem **22** von besonderer Bedeutung ist, kann diese Technologie eine breitere Anwendung bei Hardware finden, bei der ASCII-Strings (ASCII-Zeichenketten) für die Übertragung von Befehlen zu Hardware und für den Empfang von Antworten von Hardware verwendet werden.

[0328] Der Sprachtreiber **44** wird im Folgenden näher erläutert. Der Sprachtreiber **44** wird sowohl von der Treiber-Administration **32** als auch der Bewegungssteuerungs-Komponente **35** benutzt. Seine Hauptaufgabe ist die Implementierung der Funktionalität, die Bewegungssteuerungsbefehle für die spezielle unterstützte Hardware generiert.

[0329] Zum Beispiel generiert der AT6400-Treiber, der zur Steuerung der Compumotor AT6400 Bewegungssteuerungs-Hardware verwendet wird, AT6400-Befehlscodes. Während der Initialisierungsphase des Systems **22** kommuniziert der Treiber-Administrator **32** mit jedem Sprachtreiber **44**, wodurch der Benutzer die Konfiguration des Treibers **44** hinzufügen, entfernen oder ändern kann. Wenn eine Anwendung unter Einsatz des Systems **22** läuft, kommuniziert die Komponente **35** mit dem Sprachtreiber **35** und weist diesen an, die geeigneten Bewegungssteuerungsoperationen durchzuführen.

[0330] Anders als der vorstehend beschriebene Treiber **30**, der durch direktes Senden von Binärcodes mit den Bewegungssteuerungsvorrichtungen **20** kommuniziert, sendet der Sprachtreiber **44** ASCII-Text an den Stream **28**, der dann die Informationen weiter an die Bewegungssteuerungsvorrichtung **20** sendet.

[0331] Dieser Abschnitt nimmt, um die in dem Sprachtreiber **44** implementierten Merkmale zu beschreiben, Bezug auf eine Anzahl von Zeichnungen: (a) das Modul-Interaktions-Abbild in Fig. 58, das alle binären Module zeigt, die mit dem Treiber interagieren und wie sie miteinander interagieren; (b) ein Objekt-Interaktions-Abbild (Fig. 59), das dem Modul-Interaktions-Abbild entspricht, jedoch um die internen C++-Objekte erweitert ist, die den Sprachtreiber **44** bilden, und das zeigt, wie die Objekte miteinander im Dialog stehen; (c) eine Anzahl von Szenario-Abbildern (Fig. 60–Fig. 65), die die Interaktionen zeigen, die während eines bestimmten Prozesses zwischen den beteiligten C++-Objekten stattfinden; ein Schnittstellen-Abbild, das die von der Sprachtreiber-Komponente **44** freigelegten Schnittstellen, alle verwendeten Datenstrukturen und die Definitionen jeder verwendeten C++-Klasse darstellt; und (b) eine Tabelle, die zeigt, wie eine von dem Sprachtreiber **44** benutzte typische Datenbank aufgebaut ist (Fig. 67).

[0332] Das Modul-Interaktions-Abbild in Fig. 58 zeigt alle binären Module und ihre Interaktionen mit dem Sprachtreiber **44**. Es gibt zwei Module, die direkt mit dem Sprachtreiber **44** interagieren: die Bewegungssteuerungs-Komponente **35** und der Treiber-Administrator **32**. Der Treiber-Administrator **32** fragt die Treibereinstellungen ab und ändert sie, und die Komponente **35** weist den Treiber **44** an, die Bewegungssteuerungsoperationen durchzuführen, wie zum Beispiel die Bewegung zu einer bestimmten Stelle in dem System.

[0333] Insbesondere enthält das in Fig. 58 gezeigte Modul-Interaktions-Abbild die folgenden Module:
Das Treiber-Administrator-Modul **32** wird zum Installieren, De-Installieren, Registrieren und Einrichten und Konfigurieren jedes Treibers und Stream-Moduls verwendet.

[0334] Die Bewegungs-Komponente ist die Bewegungssteuerungs-Komponente **35**, die von den Anwendungen **26** benutzt wird. Die Komponente **35** kommuniziert mit dem gegenwärtigen Treiber **44**, der ihr durch den Treiber-Administrator **32** zugewiesen wurde, um von der Anwendung **26** angeforderte Bewegungssteuerungsoperationen durchzuführen.

[0335] Der Sprachtreiber **44** generiert die ASCII-Codes, aus denen die Hardware-Befehlssprache gebildet wird. Ein gegebener Sprachtreiber **44** kommuniziert nur mit einem Stream, der speziell mit diesem Treiber **44** verbunden wurde. Sobald diese Verbindung einmal hergestellt ist, wird der Stream **28** verwendet, um mit dem Zielobjekt zu kommunizieren, z.B. mit dem PC-Bus, der seriellen E/A-Anschlussstelle, der Textdatei oder dem Debug-Monitor.

[0336] Die Streams **28** sind die eigentlichen Objekte, die als Datentransportschicht für jeden Treiber **44** arbeiten. Jeder Stream **28** hat ein unterschiedliches Ziel, das den Typ des Stream definiert.

[0337] Die Registratur **42** ist die standardmäßige Windows Registrierungsdatenbank.

[0338] Das Objekt-Interaktions-Abbild in Fig. 59 bricht das in Fig. 58 gezeigte Modul-Interaktions-Abbild auf in eine detaillierte Darstellung, indem es Interaktionen einschließt, die zwischen allen C++-Objekten stattfinden, die zum Implementieren des Treibers **44** verwendet werden.

[0339] Jedes Objekt in dem Abbild wird folgendermaßen beschrieben.

CDriverObject

[0340] Dies ist das C++-Hauptobjekt, das die ganze OLE-spezifische Funktionalität implementiert, einschließlich der Funktions-Shells für alle von dem Objekt aufgedeckten OLE-Schnittstellen.

CDrvCoreDisp

[0341] Dies ist das C++-Objekt, das zum das Senden aller SPI OLE-Schnittstellenkernfunktionen an ihre jeweiligen internen Implementierungen benutzt wird. Zusätzlich zu den von dem Objekt CLangDrvCoreDisp vererbten Methoden sendet dieses Objekt treiberspezifische SPI, wie zum Beispiel die Kern-XMCSPI-OLE-Schnittstellen-Methoden.

CLangDrvCoreDisp

[0342] Die ganze Kernfunktionalität des Sprachtreibers wird durch dieses Objekt zu ihrer internen Implementierung gesandt. Zum Beispiel wird die auswählbare Sprachtreiberimplementierung für die Initialisierung durch dieses Objekt zu seiner Implementierung gesandt, die sich in der LANG_DRV-Basiscode-Bibliothek befindet.

CDrvExtDisp

[0343] Dieses C++-Objekt wird zum Senden aller erweiterten SPI-OLE-Schnittstellenfunktionen zu ihren jeweiligen internen Implementierungen verwendet. Zusätzlich zu den von dem Objekt CLangDrvExtDisp vererbten Methoden sendet dieses Objekt treiberspezifische SPI, wie zum Beispiel die erweiterten XMCSPI-OLE-Schnittstellenmethoden.

CLangDrvExtDisp

[0344] Die ganze erweiterte Sprachtreiberfunktionalität wird durch dieses Objekt zu ihrer internen Implemen-

tierung weitergegeben. Zum Beispiel wird die gesamte Streamverwaltung durch dieses Objekt zu ihrer Implementierung weitergegeben, die sich in der LANG_DRV-Basiscode-Bibliothek befindet.

CCommandMgr

[0345] Dieses Objekt wird verwendet, um Befehle zu bilden, die an den Stream gesandt werden, und um Antworten zu extrahieren, die von dem Stream empfangen werden. Der CCommandMgr ist das Kontroll-Objekt, das die Objekte CCommand, CResponse und CCommandDatabase verwaltet.

CCommand

[0346] Das Objekt CCommand bildet Befehls-Strings, die dann an das CSimpleStream gesandt werden. Jeder gebildete Befehl ist ein kompletter Bewegungssteuerungsbefehls-String.

CResponse

[0347] Dieses Objekt konvertiert Rohantwort-Strings, die von dem CSimpleStream zurückgegeben werden, und konvertiert sie in C++-Datentypen. Zum Beispiel kann ein Antwort-String, der Positionsdaten enthält, in einen Satz von Doppelwerten konvertiert werden.

CCommandDatabase

[0348] Dieses Objekt speichert die komplette Datenbank von Befehlen, die die Bewegungssteuerungs-Befehlssprache bilden. Die Datenbank kann als eine tatsächliche externe Datenbank (wie z.B. eine SQL-Datenbank) dargestellt sein, als eine Textdatei oder als kundenspezifisches Betriebsmittel in dem Treiber gespeichert sein. Gegenwärtig unterstützt der Sprachtreiber nur Datenbanken, die als kundenspezifisches Betriebsmittel in dem Treibermodul gespeichert sind.

CSPIInfo

[0349] Dieses Objekt erstellt einen in dem Objekt CCommandDatabase gespeicherten Datenbankeintrag.

CStreamMgr

[0350] Dieses Objekt ist verantwortlich für die Verwaltung der in dem Treiber registrierten Gruppen von Streams. Streams können hinzugefügt, entfernt und freigegeben werden. Nur freigegebene bzw. aktivierte Streams können tatsächlich Daten an ihre Ziele senden. Zum Beispiel sendet nur ein freigegebener PC-Bus-Stream Daten an die Bewegungssteuerungskarte, die in den PC-Bus gesteckt ist.

CSimpleStream

[0351] Dieses Objekt wird verwendet zum Initialisieren, Erstellen und direkten Kommunizieren mit der zugrundeliegenden Stream-Komponente.

CDriverInfoMgr

[0352] Dieses Objekt ist verantwortlich für die Erstellung des Objekts CDriverInfo.

CDriverInfo

[0353] Dieses Objekt enthält den kompletten Satz von Zustandsdaten, aus denen der Sprachtreiber **44** aufgebaut ist. Alle Treibereinstellungen und eine Liste von allen benutzten XMC-Streams sind in diesem Objekt gespeichert. Basisabfragen werden direkt zu diesem Objekt geleitet. Komplexere Operationen werden von einem der verschiedenen Manager-Objekte gehandhabt, die sich mit diesem Objekt verbinden, die Operation durchführen und dann ihre Verbindung zu diesem Objekt trennen.

CRegistryMgr

[0354] Dieses Objekt wird benutzt, um in dem Objekt CDriverInfo gespeicherte Informationen zu sichern und in die und aus der Registrierungsdatenbank zu laden. Insbesondere enthält es den Code, der aufgerufen wird,

wenn die ICOM_PersistRegDB-Schnittstelle aufgerufen wird.

CRegistry

[0355] Dieses Objekt führt sämtliche Registrierungsdatenbank-Operation wie Erstellen von Keys (Schlüsseln), Sichern von Werten und Abfragen von Werten durch.

[0356] Alle Hauptszenarien oder Operation, die an dem Sprachtreiber **44** stattfinden, werden nun unter Bezugnahme auf die Fig. 60–Fig. 66 beschrieben. Jedes in diesen Figuren enthaltene Szenario-Abbild stellt alle beteiligten Objekte und die Interaktionen zwischen diesen Objekten in der Reihenfolge dar, in der sie stattfinden.

[0357] Es gibt zwei Arten von Operationen, die an dem Sprachtreiber **44** stattfinden. Erstens kann der Treiber-Administrator **32** Operationen initiieren, so zum Beispiel das Hinzufügen von Streams oder das Konfigurieren des Treibers. Und zweitens kann die Bewegungssteuerungs-Komponente **35** Operationen an dem Treiber initiieren, wenn eine Anwendung tatsächlich läuft.

[0358] Bezugnehmend auf die Fig. 60–Fig. 64 werden nun alle von dem Treiber-Administrator **32** an dem Treiber **44** durchgeführten Operationen beschrieben. Jede Figur wird in der Reihenfolge beschrieben, in der sie bei der Benutzung des Treibers **44** vorkommt.

[0359] Bevor ein Treiber von der XMC-Bewegungskomponente benutzt werden kann, muss er in dem OLE-System registriert werden. Um einen Treiber zu registrieren, wie in Fig. 60 gezeigt, verifiziert der Treiber-Administrator zuerst, dass das zu registrierende Modul tatsächlich ein geeigneter Sprachtreiber ist, dann ruft er die exportierte Funktion DLLRegisterServer auf, um den Treiber zu registrieren. Jedes Modul des Systems **22** exportiert eine Funktion, die DLLGetModuleType genannt wird. Diese Funktion wird verwendet, um zu verifizieren, dass das Modul eine geeignete Treiber-Komponente ist.

[0360] Als nächstes kann der Treiber-Administrator die Komponente unter Verwendung der OLE-Funktion CoCreateInstance laden. Während des Initialisierungsprozesses lädt der Treiber alle Registrierungsdaten und initialisiert das CDriverInfo- und alle C++-Manager-Objekte.

[0361] Im Folgenden wird jeder der in Fig. 60 angeführten Schritte im Detail erläutert.

1. Während der Initialisierung muss der Treiber-Administrator die DLL laden, die die Stream-Komponente enthält, und er muss verifizieren, dass das Modul ein XMC-Treiber ist. Um dies zu tun, ruft der Treiber-Administrator die von dem Treiber exportierte Funktion DLLGetModuleType auf. Wenn die Funktion einen Wert zurückgibt, der den Wert XMC_DRIVER_MT im höherwertigen Byte enthält, fährt der Treiber-Administrator fort und registriert den Treiber durch Aufrufen seiner exportierten Funktion DLLRegisterServer. Nach dem erfolgten Aufruf schreibt die Implementierung der DLLRegisterServer alle für die OLE-Registrierung notwendigen OLE 2.0-Registrierungsinformationen in die Windows Registrierungsdatenbank.
2. Als nächstes muss der Treiber-Administrator das Treiber-Modul auf seine CLSID (Clear Screen Identification = "lösche Bildschirm"-Identifikation) abfragen. Der Aufruf der exportierten Funktion des Treibers, DLLGetCLSID, führt zur Rückgabe der CLSID. Sobald der Treiber-Administrator die CLSID hat, kann er eine Instanz bzw. ein Exemplar des Treibers erstellen, indem er die Standard-OLE-Funktion CoCreateInstance aufruft.
3. CoCreateInstance initialisiert automatisch die Komponente, indem es die durch das CDriverObject implementierte Methode ICOM_Base::Initialize aufruft.
4. Die Implementierung von ICOM_Base::Initialize weist CDrvCoreDisp an, sich selbst zu initialisieren.
5. Innerhalb seiner Initialisierung initialisiert das Objekt CDrvCoreDisp jedes seiner Manager-Objekte, wobei es mit dem CommandMgr beginnt.
6. Während der Initialisierung weist der CCommandMgr die CCommandDatabase an, sich selbst zu laden.
7. Zum Laden der Datenbank liest das Objekt CCommandDatabase die Datenbank ein und erstellt die CSPIInfo-Listen von Datenbankelementen.
8. Nach dem Initialisieren des CCommandMgr weist das Objekt CDrvCoreDisp das Objekt CDriverInfoMgr an, das Objekt CDriverInfo zu erstellen, welches später den internen Zustand der Sprachtreiber-Komponente **44** speichern wird.
9. Das Objekt CDriverInfoMgr- erstellt das Objekt CDriverInfo und reicht dieses zurück an das Objekt Dispatch. Der Zeiger auf dieses Objekt wird später in dem Komponentenstatus-Handle gespeichert, wenn das CDriverObject den ICOM_Base2::SetStateHandle aufruft.
10. Als nächstes initialisiert das Objekt CDrvCoreDisp den CStreamMgr, der für die Durchführung sämtli-

cher Stream-Operationen benutzt wird.
11. Dann initialisiert das Objekt CDrvCoreDisp den CRegistryMgr, der benutzt wird, um alle Registrierungsdatenbank-Operationen durchzuführen.
12. Schließlich initialisiert das CDriverObject das Objekt CDrvExtDisp.

[0362] Hier ist zu erwähnen, dass die ganze Initialisierung durch den COM Auto-Init-Mechanismus initiiert wird. Auto-init erscheint bei der Erstellung eines Objekt. Wenn entweder CoCreateInstance aufgerufen wird oder die Methode IClassFactory::CreateInstance, ruft die interne COM-Implementierung die Methode ICOM_Base::Initialize auf. Diese Methode löst den kompletten, in diesem Abschnitt beschriebenen Initialisierungsprozess aus.

[0363] Nach dem Initialisieren des Treibers, kann der Treiber-Administrator Operationen an dem Treiber durchführen. Zum Beispiel kann der Treiber-Administrator den Treiber auffordern, einen Stream hinzuzufügen oder zu entfernen. **Fig. 61** zeigt die Abfolge der Ereignisse, wenn der Treiber aufgefordert wird, einen neuen Stream hinzuzufügen.

[0364] Beim Hinzufügen eines Stream finden die folgenden Schritte statt:
1. Der Treiber-Administrator weist den Treiber an, einen neuen Stream hinzuzufügen und übermittelt den Dateinamen und die CLSID des hinzuzufügenden Stream an den Treiber.
2. Der Treiber übergibt den Dateinamen und die CLSID an das Objekt CDriverObject und weist dieses an, den Stream hinzuzufügen, indem es seine eingebettete C++-Klassen-Methode CLNGStreamMgmt::AddStream aufruft
3. Das eingebettete C++-Objekt, das die OLE-Schnittstelle implementiert, weist das Objekt CDrvExtDisp an, den Stream hinzuzufügen und übergibt ihm einen Handle der Komponentenstatus-Daten.
4. Das Objekt CDrvExtDisp transformiert die Komponentenstatus-Daten zuerst in einen Zeiger bzw. einen Hinweis auf das Objekt CDriverInfo.
5. Als nächstes wird der CStreamMgr mit dem CDriverInfo-Objektzeiger verbunden und angewiesen, einen neuen Stream hinzuzufügen.
6. Um den neuen Stream hinzuzufügen, benutzt der CStreamMgr ein CSimpleStream zum Laden und Erstellen der Stream-Komponente.
7. Das Objekt CSimpleStream setzt zuerst Funktionszeiger auf die von der Komponente exportierten Funktionen DllGetCLSID, DllGetModuleType und DllRegisterServer. Vor dem Laden des Moduls stellt CSimpleStream erst sicher, dass das Modul tatsächlich ein XMC-Stream ist, indem es seinen Modultyp mit dem XMC_STREAM_MT-Modultyp vergleicht. Wenn dies zutrifft, wird die Komponente in der Registrierungsdatenbank als eine OLE-Komponente registriert.
8. Unter Verwendung der in dem vorhergehenden Schritt abgefragten DllGetCLSID holt der CSimpleStream die CLSID der Komponenten und ruft CoCreateInstance auf, um eine Instanz bzw. ein Exemplar des OLE-Objekts zu laden.
9. Ist CSimpleStream fertig, fügt der CStreamMgr den Stream dem Stream-Array von CDriverInfo hinzu.

[0365] Eine weitere Operation, die nach der Initialisierung von dem Treiber angefordert wird, ist die Abfrage seiner gegenwärtigen Einstellungen. Vor der Anzeige von Treiberinformationen wie beispielsweise die von dem Treiber unterstützte Hardware, muss der Treiber-Administrator den Treiber nach diesen Informationen fragen. **Fig. 62** zeigt ein Beispiel eines Prozesses der Abfrage des Treibers nach seinen Treibereinstellungen.

[0366] Beim Abfragen der Treiberinformationen finden die folgenden Schritte statt:
1. Der Treiber-Administrator ruft die für die Abfrage der Treiberinformationen benutzte Schnittstellen-Methode auf und übergibt dem Aufruf einen Zeiger bzw. Hinweis auf die XMC_DRIVER_INFO-Struktur.
2. Der Aufruf wird von der Implementierung der OLE-Schnittstellen-Methode verwaltet, die von einer der eingebetteten C++-Klassen des CDriverObject implementiert wird.
3. Das zur Verwaltung der Schnittstelle verwendete eingebettete C++-Objekt weist entweder das Objekt CDrvCoreDisp oder CDrvExtDisp an, die Operation durchzuführen und übermittelt dem Objekt den Handle zu den Komponentenstatus-Daten.
4. Das Objekt Dispatch transformiert den Status-Handle in einen CDriverInfo-Objektzeiger. Nach Umwandlung wird das Objekt CDriverInfo nach den geeigneten Daten abgefragt.

[0367] Auf Anfrage kann der Treiber seine gesamte Konfiguration sichern oder in die oder aus der Registrierungsdatenbank laden. Diese Operation wird von der XMC-Treiberadministrations-Komponente benutzt, die alle XMC-Konfigurationsdaten in der Registrierungsdatenbank speichert. **Fig. 63** zeigt die Abfolge von Ereignissen, die stattfinden, wenn die XMC-Treiberadministrations-Komponente den Treiber anweist, seine Informa-

tionen aus der Registratur zu laden.

[0368] Während der Registrierung finden folgende Schritte statt:

1. Unter Verwendung der von dem Treiber aufgedeckten OLE-Schnittstelle ICOM_PersistRegDB weist der Treiber-Administrator die Komponente zunächst an, ihre Konfigurationsdaten zu laden.
2. Das eingebettete Objekt des CDriverObject, das für die Verwaltung aller Aufrufe von ICOM_PersistRegDB verwendet wird, wird aufgerufen und führt die Operation durch.
3. Nach dem Aufruf weist das eingebettete Objekt das CDrvCoreDisp-Objekt an, das Laden durchzuführen und übergibt ihm einen Handle zu den Komponentenstatus-Daten.
4. Das Objekt CDrvCoreDisp transformiert den Status-Handle zuerst in einen CDriverInfo-Objektzeiger.
5. Als nächstes wird der CRegistryMgr mit dem CDriverInfo-Zeiger verbunden und angewiesen, seinen Inhalt in die oder aus der Registrierungsdatenbank zu laden.
6. Der CRegistryMgr lädt sämtliche allgemeinen Treiberinformationen und füllt die in dem Objekt CDriverInfo gespeicherte Treiberdatenstruktur XMC_DRIVER_INFO aus.
7. Wenn in dem Treiber irgendwelche Stream-Informationen gespeichert sind, lädt der CRegistryMgr die Stream-Informationen und führt die XMC_STREAM_INFO-Struktur aus. Die Struktur wird dann verwendet, um ein neues Objekt CSimpleStream zu erstellen.
8. Beim Erstellen eines Stream erfolgen durch das Objekt CSimpleStream erst die Abfrage und der Aufruf der exportierten Funktion DllGetModuleType und es verifiziert, dass das Modul tatsächlich eine Stream-Komponente ist. Wenn dies zutrifft, fragt CSimpleStream die von der Komponente exportierte Funktion DLLRegisterServer ab und ruft sie auf, um die Komponente zu registrieren.
9. Nach dem Registrieren der Komponente fragt das Objekt CSimpleStream die exportierte Funktion DllGetCLSID ab und ruft sie auf, um die CLSID der Komponente zu holen. Unter Verwendung der CLSID wird CoCreateInstance aufgerufen, um eine Instanz bzw. ein Exemplar des OLE-Objekts zu erstellen.
10. Ist CSimpleStream fertig, verbindet der CRegistryMgr eine temporäre Instanz des CStreamMgr mit dem CDriverInfo-Objektzeiger und weist ihn an, den neuen Stream hinzuzufügen.
11. Der CStreamMgr manipuliert das Stream-Array der CDriverInfo direkt, um den neuen Stream hinzuzufügen. Wenn der neue Stream hinzugefügt ist, wird eine neue Instanz bzw. ein neues Exemplar des Objekts CSimpleStream erstellt und an CSimpleStream angefügt, das an den CStreamMgr übergeben wird.
12. Beim Anfügen an den Stream fragt das neue CSimpleStream die ihm übermittelte Schnittstelle IXMC_StreamInit nach alle benutzten Schnittstellen ab, um die Referenzzählungen für die Komponente zu erhöhen.

[0369] Nachdem der Treiber-Administrator mit der Benutzung des Treibers fertig ist, muss er den Treiber durch Aufrufen seiner aufgedeckten Release-Methode freigeben. Der Aufruf dieser Methode weist den Treiber an, alle die von ihm benutzten Betriebsmittel freizugeben. **Fig. 64** zeigt den Prozess der Freigabe der Treiber-Komponente.

[0370] Während des Reinigungsprozesses finden die folgenden Schritte statt.

1. Zunächst rufen der XMC-Treiber-Administrator oder die XMC-Bewegungs-Komponente die endgültige IUnknown::Release auf.
2. Wenn aufgerufen, wird die durch das CDriverObject implementierte Methode IUnknown::Release aufgerufen. Nach diesem Aufruf wird die interne OLE-Referenzzählung veranlasst, auf Null zu gehen, der Treiber ruft seine Implementierung von ICOM_Base::Uninitialize auf, um alle benutzten Betriebsmittel zu löschen.
3. Zuerst weist ICOM_Base::Uninitialize das CDrvExtDisp an, sämtliche von ihm benutzten Betriebsmittel zu löschen.
4. Als nächstes weist ICOM_Base::Uninitialize das Objekt CDrvExtDisp an, sämtliche von ihm benutzten Betriebsmittel zu löschen.
5. Da das Objekt CDrvCoreDisp Instanzen bzw. Exemplare aller Manager-Objekte enthält, beginnt es sie zu löschen, indem es erst den CCommandMgr anweist, sämtliche von ihm benutzten Betriebsmittel zu löschen. Der CCommandMgr zerstört intern die CCommandDatabase und ihren gesamten Inhalt.
6. Als nächstes zerstört das Objekt CDrvCoreDisp ausdrücklich alle anderen Manager-Objekte, indem es deren Destruktoren aufruft.
7. Und als letzten Schritt löscht die Methode ICOM_Base::Unitialize den Status-Handle, der einen Zeiger auf das Objekt CDriverInfo enthält. Nach dem Zerstören löscht das Objekt CDriverInfo jedes Objekt CSimpleStream, das wiederum seine Instanzen bzw. Exemplare der XMC-Stream-Komponente freigibt. Nach Freigabe der letzten Instanz bzw. des letzten Exemplars der XMC-Stream-Komponente wird die Komponente dll aus dem Speicher gelöscht.

[0371] Nach erfolgreichem Installieren eines Treibers in dem XMC-System und Konfigurieren unter Verwen-

dung des Treiber-Administrators ist der Treiber bereit für die Benutzung durch die XMC-Bewegungssteuerungs-Komponente. Die Komponente benutzt den Treiber, wenn sie Bewegungssteuerungsoperationen durchführt, die von einer die Komponente benutzenden Anwendung angefordert werden. Fig. 65 beschreibt die von der Komponente angewiesenen Operationen, die an dem Treiber stattfinden können.

[0372] Bevor sie arbeitet, muss die XMC-Bewegungs-Komponente die Treiber-Administrator-Komponente **32** nach ihrer Nummerierung abfragen. Die zurückgegebene Nummerierung wird für den Zugriff auf alle freigegebenen Treiber verwendet, die zur Durchführung von XMC SPI-Operationen durch die XMC-Bewegungs-Komponente angewiesen werden.

[0373] Ist die Treibernummerierung einmal erfasst, kann die Bewegungssteuerungs-Komponente **35** den oder die freigegebenen Treiber anweisen, bestimmte Befehlsoperationen durchzuführen. Befehlsoperationen sind standardmäßige Bewegungssteuerungsoperationen wie beispielsweise die Bewegung zu einer bestimmten Stelle in dem System oder die Abfrage des Systems nach der aktuellen Position. Fig. 65 beschreibt den Prozess der Befehlserteilung an den Treiber zur Durchführung bestimmter Operationen.

[0374] Bei der Befehlserteilung an den Treiber zur Durchführung einer bestimmten Operation finden die folgenden Schritte statt.

1. Die Bewegungs-Komponente weist den Treiber zunächst an, die Operation durchzuführen, z.B. die Bewegung in eine bestimmte Position oder die Abfrage des Systems nach der gegenwärtigen Position.
2. Der XMCSPI-Aufruf wird von dem CDriverObject verwaltet, das alle von der Komponente aufgedeckten OLE-Schnittstellen implementiert.
3. Das CDriverObject weist entweder das Objekt CDrvCoreDisp oder CDrvExtDisp an, die Operation durchzuführen, abhängig davon, ob die Operation eine Kern- oder eine erweiterte XMCSPI ist. Der Komponentenstatus-Handle wird bei Aufruf an das Dispatch-Objekt übergeben.
4. Das Dispatch-Objekt transformiert dann den Status-Handle in einen CDriverInfo-Objektzeiger.
5. Als nächstes verbindet das Dispatch-Objekt den CCommandMgr mit dem CDriverInfo-Objektzeiger und weist ihn an, die Operation entsprechend dem gesandten Datenbankindex durchzuführen. Der Datenbankindex entspricht der aufgerufenen XMCSPI und wird benutzt, um den Sprachdatenbankeintrag für diesen SPI-Aufruf zu finden.
6. Der CCommandMgr sucht in der CCommandDatabase nach dem Index und bildet ein der XMCSPI-Operation entsprechendes Objekt CCommand.
7. Als nächstes greift der CCommandMgr direkt auf die CDriverInfo zu und übermittelt den durch das Objekt CCommand aufgebauten Command-String an alle freigegebenen Streams.
8. Jeder freigegebene bzw. aktivierte Stream sendet den ASCII-Text an sein Ziel. Zum Beispiel sendet der PC-Bus-Stream seine Daten an die Bewegungssteuerungskarte, die sich auf dem PC-Bus befindet. Der Textdatei-Stream sendet dagegen seine Daten an die zugehörige Textdatei.
9. Auf Anweisung fragt der CCommandMgr den ersten lesbaren Stream nach den Ergebnissen der ihm zugesandten Befehle.
10. CSimpleStream liest die Rohantwort von dem Ziel und reicht sie zurück an den CCommandMgr.
11. Sobald der CCommandMgr die Rohantwort empfängt, benutzt er das Objekt CResponse, um die Rohantwort auf Grundlage des dem XMCSPI-Datenbankeintrag entsprechenden Antwortformats zu parsen, d.h. syntaktisch zu analysieren. Alle Antwortparameter werden entlang der Aufrufkette zurückgereicht und enden schließlich in den Händen des ursprünglichen Aufrufers, nämlich der XMC-Bewegungs-Komponente.

[0375] Die von der XMC-Bewegungs-Komponente initiierte Reinigung durch die Freigabe der XMC-Treiber-Komponente ist genauso wie die Reinigung, die stattfindet, wenn das Objekt Treiber-Administrator **32** die Komponente freigibt.

[0376] Der folgende Beitrag erläutert die tatsächlich aufgedeckten OLE-Schnittstellen, die beim Übermitteln von Daten benutzten Definitionen der Datenstrukturen und die Definitionen einer jeden von dem Treiber intern benutzten Klasse.

[0377] Das folgende Diagramm beschreibt alle von dem für die Interpretierbarkeit der Treiber-Komponente spezifischen Treiber aufgedeckten Schnittstellen. Fig. 66 zeigt die von der Komponente aufgedeckten Schnittstellen.

[0378] Außer den zwei aufgedeckten Standard-Schnittstellen wie IUnknown und IClassFactory gibt es drei Kategorien von Schnittstellen, die von der Komponente aufgedeckt werden. Die drei Kategorien sind wie folgt.

[0379] COM: Alle Schnittstellenbezeichnungen mit dem COM_Präfix implementieren die allgemeine COM-Funktionalität. Die Schnittstellen ICOM_Base, ICOM_Base2, ICOM_Persist2 und ICOM_PersistRegDB fallen zum Beispiel in diese Kategorie.

[0380] LNG: Alle Schnittstellenbezeichnungen mit dem LNG_Präfix implementieren die allgemeine Sprachtreiberfunktionalität. Die Schnittstellen ILNG_DrvCore_Init und ILNG_DrvExt_StreamMgmt fallen zum Beispiel in diese Kategorie.

[0381] XMC: Alle Schnittstellenbezeichnungen mit dem XMC_Präfix implementieren XMCSPI(Treiber)-Funktionalität.

[0382] Die folgenden Abschnitte beschreiben die Schnittstellen, die sowohl in die Kategorie COM als auch in die Kategorie LNG fallen. Alle anderen Schnittstellen sind XMCSPI-spezifisch und dienen dem alleinigen Zweck der Durchführung von Bewegungssteuerungsoperationen.

[0383] Die folgenden aufgedeckten Methoden in der Schnittstelle ICOM_Base werden beim Initialisieren und De-Initialisieren der Komponente verwendet. Jede dieser Methoden ruft versteckte Initialisierungs- und De-Initialisierungs-Schnittstellenmethoden auf, die von allen Schnittstellen der Komponente implementiert werden.

```
DECLARE_INTERFACE_( ICOM_BASE, IUnknown )
{
        STDMETHOD ( Initialize ) ( THIS_LPVOID pInitInfo ) PURE;
        STDMETHOD ( Unitialize ) ( THIS ) PURE;
};
```

[0384] Die Schnittstelle ICOM_Base2 erbt von der Schnittstelle ICOM_Base verschiedene Methoden, die zum Verwalten des internen Komponentenstatus-Handle verwendet werden, und addiert sie. Außerdem erlaubt es eine Methode dem Benutzer, ein von der Komponente zurückgereichtes HRESULT in einen von Menschen lesbaren Text-String zu übersetzten. Das Folgende ist eine Beschreibung der Schnittstelle ICOM_Base2.

```
DECLARE_INTERFACE_( ICOM_Base2, ICOM_Base )
{
        STDMETHOD ( SetStateData ) (THIS_COM_STATEHANDLE hState ) PURE;
        STDMETHOD ( GetStateData ) ( THIS_LPCOM_STATEHANDLE phState )
PURE;
        STDMETHOD ( GetErrorString ) ( THIS_HRESULT hr, LPSTR pszErr, DWORD
dwMax ) PURE;
};
```

[0385] Die Schnittstelle ICOM_Persist2 übernimmt und addiert verschiedene Methoden, die für die Abfrage der CLSID und des Modultyps verwendet werden, von der Standard-OLE-Schnittstelle IPersist. Das Folgende ist eine Beschreibung der Schnittstelle ICOM_Persist2.

```
DECLARE_INTERFACE_( ICOM_Persist2, IPersist )
{
        STDMETHOD ( GetID ) (THIS_LPDWORD pdwID ) PURE;
        STDMETHOD ( GetModule Type ) ( THIS_LPDWORD pdwMT ) PURE;
};
```

[0386] Die Schnittstelle ICOM_PersistRegDB implementiert eine Funktionalität, die jener ähnlich ist, die von

der Standard-OLE-Schnittstelle IPersistFile bereitgestellt wird. Anstelle des Sicherns und Ladens von Daten in und aus einer Datei arbeitet die ICOM_PersistRegDB an der Registrierungsdatenbank. Das Folgende ist eine Beschreibung der Schnittstelle ICOM_PersistRegDB.

```
DECLARE_INTERFACE_( ICOM_PersistRegDB, IUnknown )
{
        STDMETHOD ( IsDirty )( THIS ) PURE;
        STDMETHOD ( Load )( THIS_HKEY hKey ) PURE;
        STDMETHOD ( Save )( THIS_HKEY hKey ) PURE;
        STDMETHOD ( Clear )( THIS_HKEY hKey ) PURE;
};
```

[0387] Die Schnittstelle ILNG_DrvCore_Init wird zum Initialisieren der Sprachtreiber-Komponente verwendet. Das Folgende ist eine Beschreibung der Schnittstelle ILANG_DrvCore-Init.

```
DECLARE_INTERFACE_( ILANG_DrvCore_Init, IUnknown )
{
        STDMETHOD (Create)( THIS_LPLNG_DRIVER_INFO pDI ) PURE;
        STDMETHOD (Destroy)( THIS ) PURE;
        STDMETHOD (Setup)( THIS_LPLNG_DRIVER_INFO pDI ) PURE;
        STDMETHOD (Stat)( THIS_LPLNG_DRIVER_INFO pDI ) PURE;
        STDMETHOD (Register)( THIS ) PURE;
        STDMETHOD (UnRegister)( THIS ) PURE;
        STDMETHOD (IsRegistered)( THIS_LPBOOL pbRegistered ) PURE;
        STDMETHOD (Enable)( THIS_BOOL fEnable ) PURE;
        STDMETHOD (IsEnabled)( THIS_LPBOOL pbEnabled ) PURE;
};
```

[0388] Die Schnittstelle ILNG_DrvExt_StreamMgmt wird benutzt, um alle Stream-Operationen durchzuführen. Das Folgende ist eine Beschreibung der Schnittstelle ILNG_DrvExt_StreamMgmt.

```
DECLARE_INTERFACE_( ILNG_DrvExt_StreamMgmt, IUnknown )
{
        STDMETHOD (GetStreamEnumeration)( THIS_LPENUMUNKNOWN FAR
        *ppEnumStream ) PURE
        STDMETHOD (GetStreamCount)( THIS_LPDWORD pdwCount ) PURE;
```

```
        STDMETHOD (GetStreamInit)( THIS_XMC-STREAMID idStream,
        LPXMCSTREAMINIT FAR *ppStreamInit ) PURE;
        STDMETHOD (GetStreamInitAt)(THIS_ DWORD dwIdx, LPXMCSTREAMINIT
        FAR *ppStreamInit ) PURE;
        STDMETHOD (AddStream)(THIS_ LPXCSTREAMINIT pStreamInit) PURE;
        STDMETHOD (RemoveStream)( THIS_ LPXMCSTREAMINIT pStreamInit,
        BOOL bDestroy ) PURE;
        STDMETHOD (RemoveStream)( THIS_ XMC_STREAMID idStream, BOOL
        bDestroy ) PURE;
        STDMETHOD (RemoveAllStreams)( THIS_ BOOL bDestroy ) PURE;
        STDMETHOD (EnabledStreamsOnly)( THIS_ BOOL bEnabledOnly, LPBOOL
        pbOldEnabledOnly ) PURE;
};
```

[0389] Folgende sind die von der Treiber-DLL exportierten Funktionen.

XMC_DRIVER_MODULETYPE	DLLGetModuleType(void);
LPCLSID	DLLGetCLSID(void);
BOOL	DLLRegisterServer(void);
BOOL	DLLUnRegisterServer(void);

[0390] Im Folgenden werden alle von dem Treiber benutzten Strukturen und Nummerierungen definiert.

[0391] Die Nummerierung XMC_DRIVER_MODULETYPE definiert den Typ der verfügbaren Treiber. Jeder Treiber muss seinen Typ rückmelden, wenn der Benutzer die exportierte DLLGetModuleType-Funktion aufruft.

```
enum XMC_DRIVER_MODULETYPE
{
        XMC_DRIVER_MT               = 0x4000,
        XMC_DRIVER_MT_AT6400        = 0x4001,
        XMC_DRIVER_MT_DMC1000       = 0x4002,
        XMC_DRIVER_MT_DT2000        = 0x4003,
        XMC_DRIVER_MT_CUSTOM        = 0x4004
};
```

[0392] Die Nummerierung XMC_DRVCORE_CMD definiert einen Bezeichner für jeden dem XMC-Treiber bekannten Befehl. Zum Beispiel hat jede Kerntreiberfunktion einen entsprechenden XMC_DRVCORE_CMD-Bezeichner. Dieser Index wird verwendet, um den String-Block für den Befehl zu suchen. Die Definition der Nummerierung ist wie folgt.

```
enum XMC_DRVCORE_CMD
{
        XMC_DCC_MOTION_MOVEABS,
        XMC_DCC_MOTION_KILL,
        :
};
```

[0393] Die Nummerierung XMC_DRVEXT_CMD defininiert einen Bezeichnet für jeden dem XMC-Treiber bekannten erweiterten Befehl. Selbst wenn die Bezeichnet existieren, kann der Treiber die Gruppe von Befehlen implementieren oder nicht. Zum Beispiel hat jede erweiterte Treiberfunktion einen entsprechenden XMC_DRVEXT_CMD-Bezeichner. Dieser Index wird verwendet, um den String-Block für den Befehl zu suchen (wenn der Treiber den Befehl implementiert). Die Definition der Nummerierung ist wie folgt.

```
enum XMC_DRVEXT_CMD
{
        XMC_DCE_MOTION_MOVEREL,
        :
};
```

[0394] Die Nummerierung LNG_PARAM_DATATYPE definiert alle Datenblocktypen, die aus den von den Stream-Zielen zurückgesandten Antwort-Strings syntaktisch analysiert werden können.

```
typedef enum_LNG_PARAM_DATATYPE
{
        LNG_ADT_NOP,
        LNG_ADT_NOTYPE,
        LNG_ADT_NUMBER,
        LNG_ADT_STAT_STRING,
        LNG_ADT_MEM_STRING,
}LNG_PARAM_DATATYPE;
```

[0395] Die LNG_PARAM_DATA-Struktur speichert alle Datentypen, die einen Parameter beschreiben, der entweder in einen Befehl eingebaut ist oder aus einer Antwort geparst wird.

```
struct LNG_PARAM_DATA
{
        //---- Constructor & Destructor ----

        LNG_PARAM_DATA( void );
        ~LNG_PARAM_DATA( void );


        //---- Data ----

        LNG_PARAM_DATATYPEadt;

        union
        {
                double df;
                LPTSTR psz;
        };
};
```

[0396] Die LNG_DRIVER_INFO-Struktur wird beim Einrichten und Abfragen des Treiberstatus verwendet.

```
typedef struct_LNG_DRIVER_INFO
{
        DWORD                   m_mt;
        LNG_DRIVERID    m_ID;
        TCHAR                   m_szName[ LNG_DRIVER_NAME_LEN+1 ];
        TCHAR                   m_szDescription[ LNG_DRIVER_DESC_LEN+1 ];
        TCHAR                   m_szHWVendor[ LNG_DRIVER_NAME_LEN+1 ];
}LNG_DRIVER_INFO;
```

[0397] Jeder XMC-Treiber ist verantwortlich für die Verwaltung aller benutzten Streams. Um jeden Stream über die Zeit anhaltend zu verwalten, implementieren jedes Treiber- und Stream-Modul die durch die Schnittstelle ICOM_PersistRegDB aufgedeckte anhaltende Funktionalität. Wenn die Implementierung ICOM_PersistRegDB::Save des Treibers aufgerufen wird, werden die Daten in der Registratur in der nachstehenden Reihenfolge gesichert.

```
XMCDriverAdminObject.100
        |---- Drivers
                |---- dwCount = <# of drivers>
                |---- XMCDrv_0
                |       |---- CLSID = {clsid}
                |       |---- dwFlags = <driver flags>
                |       |---- dwID = <driver ID>
                |       |---- dwModuleType = XMC_DRIVER_MT_xxx
                |       |---- szDescription = <user desc of the driver>
                |       |---- Streams
                |               |---- Count = <# of streams>
                |               |---- XMCStrm_0
                |               |       |---- CLSID = {clsid}
                |               |       |---- dwID = <strm id>
                |               |       |---- dwModuleType = XMC_STREAM_MT_xxx
                |               |       |---- <stream specific values>
                |               |
|               |---- XMCStrm_<n>
|               |       |---- _
|
|---- XMDDrv_<n>
        |---- _
```

[0398] Aus obiger Beschreibung sollte erkennbar sein, dass andere spezielle Ausführungsformen der vorliegenden Erfindung möglich sind, ohne von den erfindungswesentlichen Merkmalen abzuweichen. Deshalb dienen die vorstehenden Ausführungsformen in jeder Hinsicht zur Veranschaulichung der Erfindung, ohne diese einzuschränken, wobei der Rahmen der Erfindung nicht durch deren Beschreibung, sondern vielmehr durch die anliegenden Ansprüche angegeben wird.

Patentansprüche

1. System (**10**) zur Erzeugung einer Folge von Steuerbefehlen zum Steuern einer gewählten Bewegungssteuerungsvorrichtung (**20**), die aus einer Gruppe von unterstützten Bewegungssteuerungsvorrichtungen ausgewählt wurde, umfassend:
einen Satz von Bewegungssteuerungsoperationen, wobei jede Bewegungssteuerungsoperation entweder eine Grundoperation ist, deren Implementierung nicht durch die Verwendung anderer Bewegungssteuenangsoperationen simuliert werden kann, oder eine Nicht-Grundoperation, die die Definition einer Grundoperation nicht erfüllt;
einen Kernsatz von Kerntreiberfunktionen, wobei jede Kerntreiberfunktion einer der Grundoperationen zugeordnet ist;
einen erweiterten Satz von erweiterten Treiberfunktionen, wobei jede erweiterte Treiberfunktion einer der Nicht-Grundoperationen zugeordnet ist;
einen Satz von Komponentenfunktionen;
einen Komponentencode, der jeder der Komponentenfunktionen zugeordnet ist, wobei der Komponentencode zumindest einige der Komponentenfunktionen zumindest einigen der Treiberfunktionen zuordnet;
einen Satz von Softwaretreibern (**30**), wobei jeder Softwaretreiber (**30**) zumindest einer der unterstützten Bewegungssteuerungsvorrichtungen (**20**) zugeordnet ist und einen Treibercode für die Implementierung der Treiberfunktionen umfasst, und

ein Anwendungsprogramm (**26**), das einen Satz von Komponentenfunktionen umfasst, die definieren, wie die Bewegungssteuerungsvorrichtungen in einer gewünschten Weise zu betätigen sind; und
ein Steuerbefehl-Erzeugungsmodul (**35**) zum Erzeugen der Steuerbefehlsfolge für die Steuerung der ausgewählten Bewegungssteuerungsvorrichtung (**18**) auf der Basis des Komponentencodes, der den Komponentenfunktionen des Anwendungsprogramms (**26**) zugeordnet ist, und des Treibercodes, der dem Softwaretreiber (**30**) zugeordnet ist, der der gewählten Bewegungssteuerungsvorrichtung (**18**) zugeordnet ist.

2. System nach Anspruch 1, bei welchem die Softwaretreiber (**30**) einen Treibercode für die Implementierung sämtlicher Kerntreiberfunktionen umfassen.

3. System nach Anspruch 2, bei welchem die Softwaretreiber (**30**) einen Treibercode für die Implementierung zumindest einiger der erweiterten Treiberfunktionen umfassen.

4. System nach Anspruch 3, bei welchem
nichtunterstützte erweiterte Treiberfunktionen erweiterte Treiberfunktionen sind, denen kein Treibercode zugeordnet ist; und
bei welchem das Steuerbefehl-Erzeugungsmodul Steuerbefehle auf der Basis des einer Kombination von Kerntreiberfunktionen zugeordneten Treibercodes erzeugt, um die Bewegungssteuerungsoperationen zu emulieren, die zumindest einigen der nichtunterstützten erweiterten Treiberfunktionen zugeordnet sind.

5. System nach Anspruch 1, ferner umfassend
Mittel zum Bestimmen eines von den Softwaretreibern verwendeten Treiber-Einheitensystems; und
Mittel zum Konvertieren eines von dem Anwendungsprogramm verwendeten Anwendungs-Einheitensystem in das Treiber-Einheitensystem.

6. System nach Anspruch 1, ferner umfassend
eine Vielzahl von Bestimmungsorten von Steuerbefehlen, wobei einer der Vielzahl von Bestimmungsorten von Steuerbefehlen ein ausgewählter Bestimmungsort von Steuerbefehlen ist;
eine Vielzahl von Datenströmen (**28**), wobei jeder Datenstrom einen Sendestromcode enthält, der bestimmt, wie die Steuerbefehle zu wenigstens einem der Vielzahl von Bestimmungsorten von Steuerbefehlen zu übertragen sind; und
Datenstrom-Steuermittel zum Kommunizieren der Steuerbefehle zu dem gewählten Bestimmungsort von Steuerbefehlen auf der Basis des enthaltenen Sendestromcodes durch den Datenstrom, der dem gewählten Bestimmungsort von Steuerbefehlen zugeordnet ist.

7. System nach Anspruch 6, bei welchem bestimmte der Bestimmungsorte von Steuerbefehlen Antwortdaten erzeugen, wobei
die Datenströme (**28**), die den Antwortdaten erzeugenden Bestimmungsorten von Steuerbefehlen zugeordnet sind, jeweils einem Antwortdatenstromcode zugeordnet sind; und wobei
das Datenstrom-Steuermittel die Antwortdaten auf der Basis des Antwortdatenstromcodes verarbeitet.

8. System nach Anspruch 1, ferner umfassend eine Befehlsformatschablone und
eine Antwortformatschablone, die jeder Treiberfunktion zugeordnet sind, wobei das Steuerbefehl-Erzeugungsmodul ferner umfasst:
Mittel zum Erzeugen von Befehlsdatenketten zum Steuern der ausgewählten Bewegungssteuerungsvorrichtung auf der Basis der Befehlsformatschablone und des Anwendungsprogramms; und
Mittel zum syntaktischen Analysieren von Antwortdatenketten, die von der ausgewählten Bewegungssteuerungsvorrichtung erzeugt werden, auf der Basis der Antwortformatschablone und des Anwendungsprogramms.

9. System nach Anspruch 1, ferner umfassend ein Netzwerk-Kommunikationsprotokoll, das eine Kommunikation der Steuerbefehle zwischen dem Steuerbefehl-Erzeugungsmodul und der ausgewählten Bewegungssteuerungsvorrichtung über ein Netzwerk erlaubt.

10. System nach Anspruch 1, ferner umfassend ein Netzwerk-Kommunikationsprotokoll, das eine Kommunikation der Komponentenfunktionen von dem Anwendungsprogramm zu dem Steuerbefehl-Erzeugungsmodul über ein Netzwerk erlaubt.

11. System nach Anspruch 9, bei welchem das Netzwerk-Kommunikationsprotokoll ferner eine Kommunikation der Komponentenfunktionen von dem Anwendungsprogramm zu dem Steuerbefehl-Erzeugungsmodul

über das Netzwerk erlaubt.

12. System nach Anspruch 10 oder 11, bei welchem das Anwendungsprogramm auf einem ersten Arbeitsplatzrechner und das Steuerbefehl-Erzeugungsmodul auf einem zweiten Arbeitsplatzrechner arbeitet.

Es folgen 65 Blatt Zeichnungen

Anhängende Zeichnungen

FIG. 1A

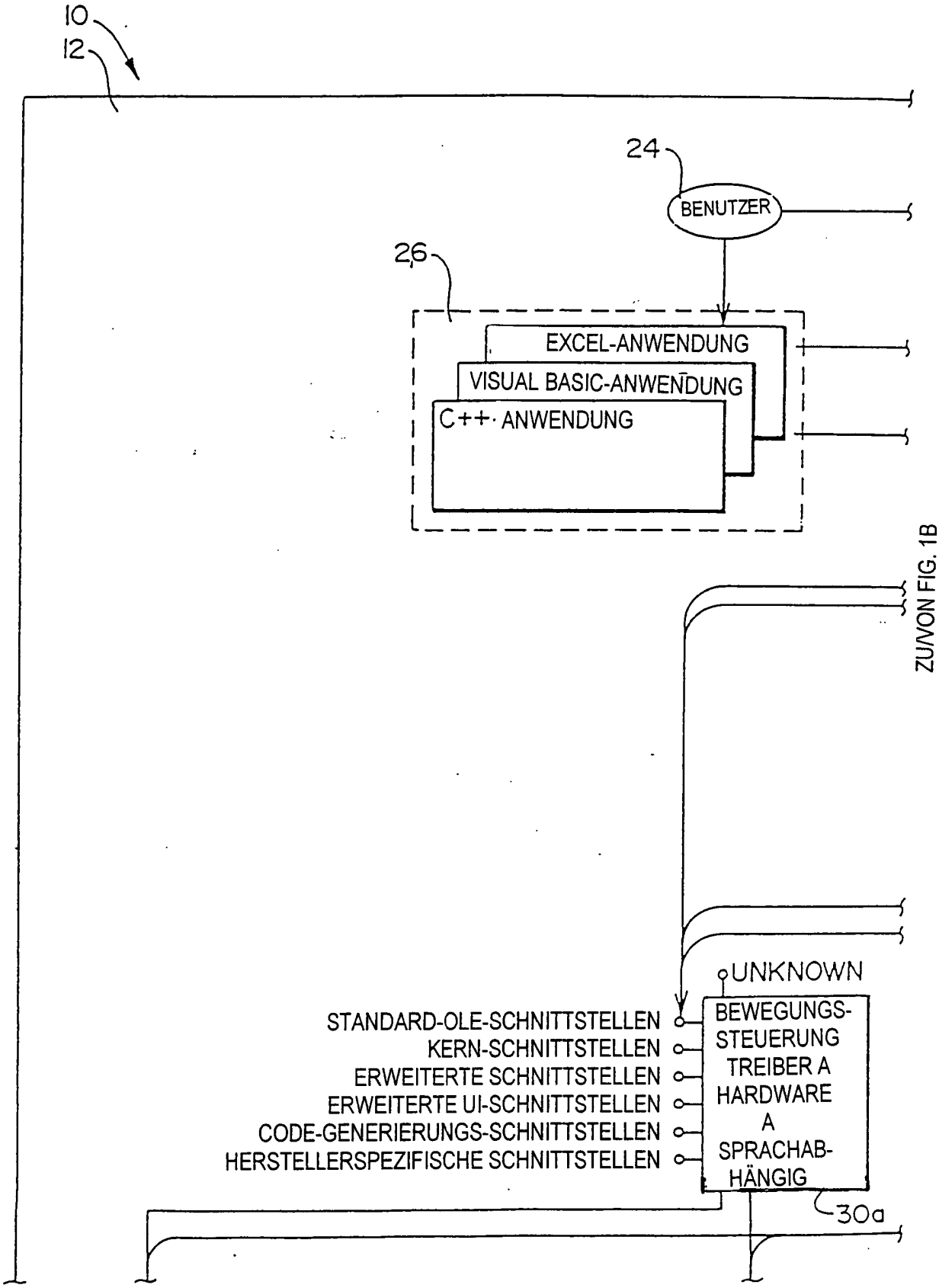


FIG. 1B

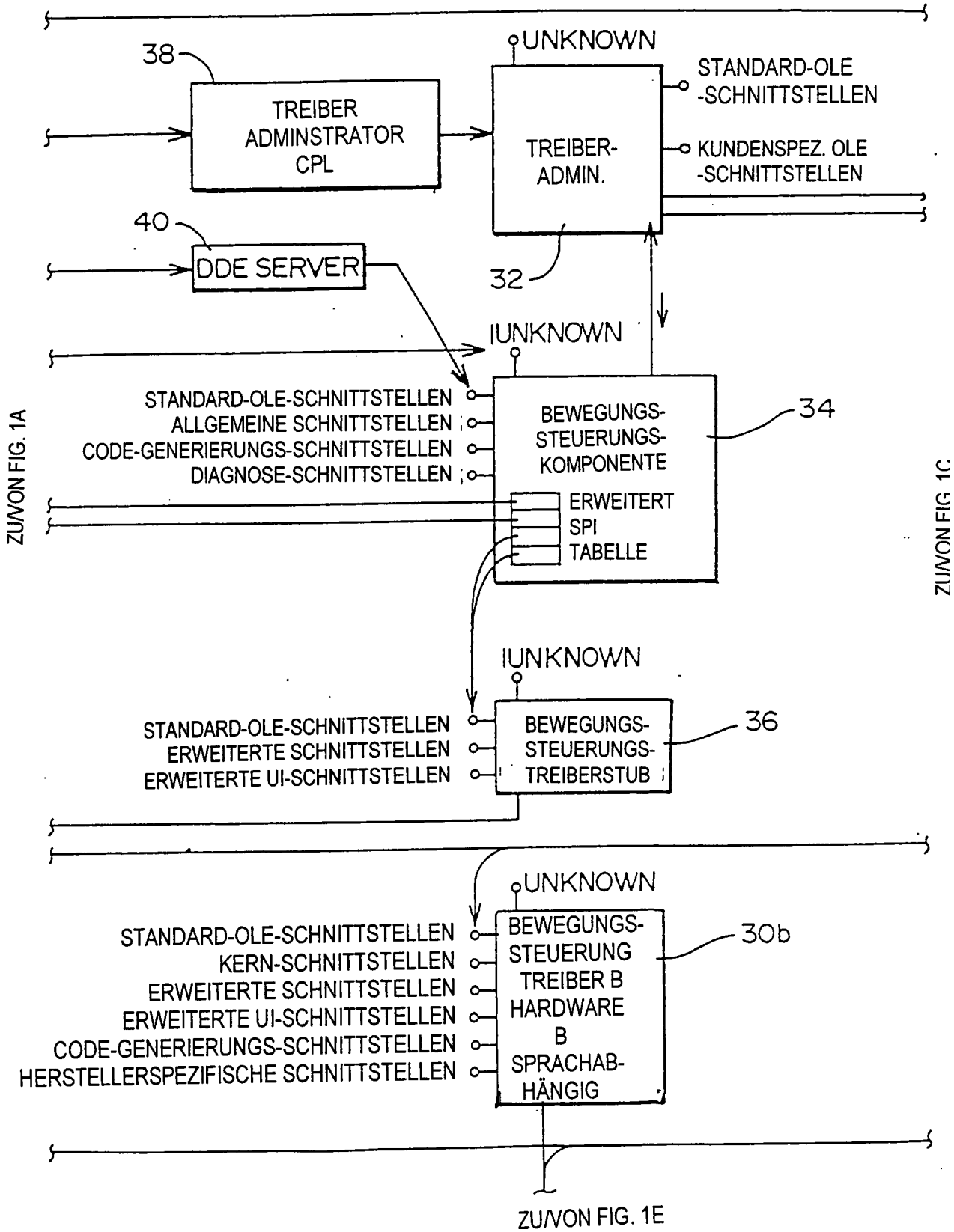

38
TREIBER ADMINSTRATOR CPL
UNKNOWN
TREIBER-ADMIN.
STANDARD-OLE -SCHNITTSTELLEN
KUNDENSPEZ. OLE -SCHNITTSTELLEN
40
DDE SERVER
32
IUNKNOWN
STANDARD-OLE-SCHNITTSTELLEN
ALLGEMEINE SCHNITTSTELLEN
CODE-GENERIERUNGS-SCHNITTSTELLEN
DIAGNOSE-SCHNITTSTELLEN
BEWEGUNGS-STEUERUNGS-KOMPONENTE
34
ERWEITERT
SPI
TABELLE
ZU/VON FIG. 1A
ZU/VON FIG. 1C
IUNKNOWN
STANDARD-OLE-SCHNITTSTELLEN
ERWEITERTE SCHNITTSTELLEN
ERWEITERTE UI-SCHNITTSTELLEN
BEWEGUNGS-STEUERUNGS-TREIBERSTUB
36
UNKNOWN
STANDARD-OLE-SCHNITTSTELLEN
KERN-SCHNITTSTELLEN
ERWEITERTE SCHNITTSTELLEN
ERWEITERTE UI-SCHNITTSTELLEN
CODE-GENERIERUNGS-SCHNITTSTELLEN
HERSTELLERSPEZIFISCHE SCHNITTSTELLEN
BEWEGUNGS-STEUERUNG TREIBER B HARDWARE B SPRACHAB-HÄNGIG
30b
ZU/VON FIG. 1E

FIG. IC

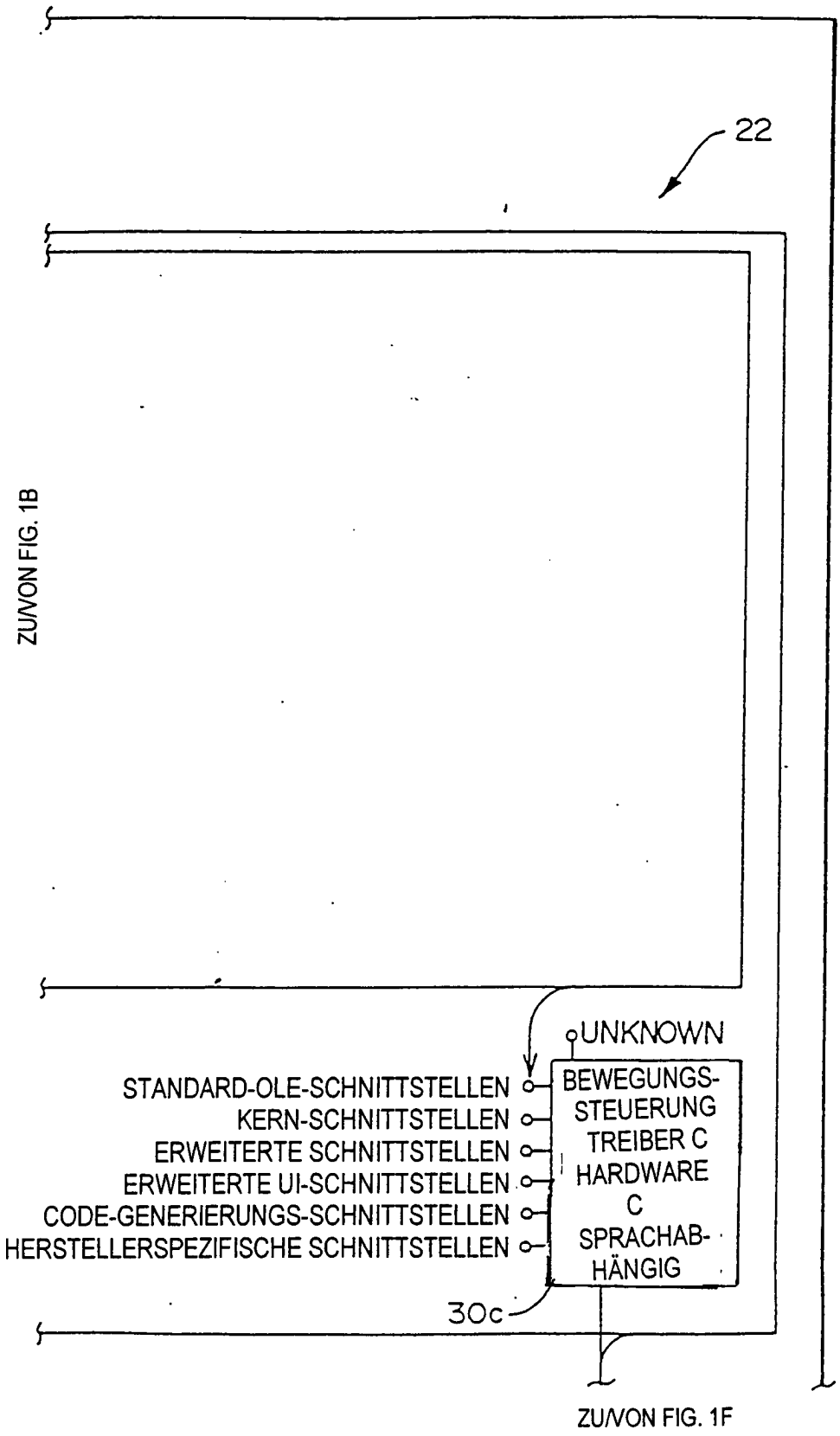

22
ZU/VON FIG. 1B
UNKNOWN
STANDARD-OLE-SCHNITTSTELLEN
KERN-SCHNITTSTELLEN
ERWEITERTE SCHNITTSTELLEN
ERWEITERTE UI-SCHNITTSTELLEN
CODE-GENERIERUNGS-SCHNITTSTELLEN
HERSTELLERSPEZIFISCHE SCHNITTSTELLEN
BEWEGUNGS-STEUERUNG TREIBER C HARDWARE C SPRACHAB-HÄNGIG
30c
ZU/VON FIG. 1F

FIG. 1D

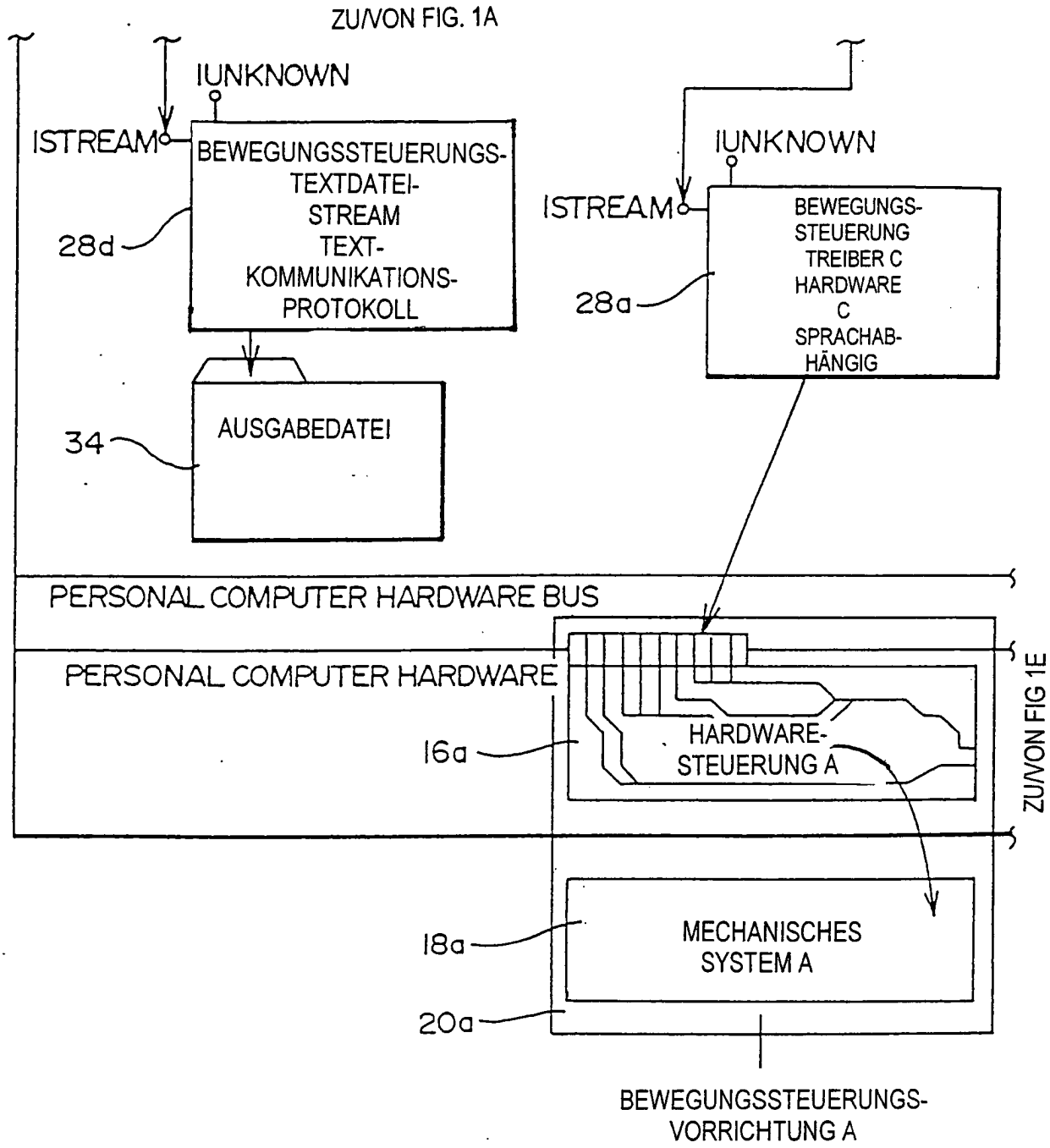

ZU/VON FIG. 1A
IUNKNOWN
ISTREAM
BEWEGUNGSSTEUERUNGS-TEXTDATEI-STREAM TEXT-KOMMUNIKATIONS-PROTOKOLL
28d
AUSGABEDATEI
34
IUNKNOWN
ISTREAM
BEWEGUNGS-STEUERUNG TREIBER C HARDWARE C SPRACHAB-HÄNGIG
28a
PERSONAL COMPUTER HARDWARE BUS
PERSONAL COMPUTER HARDWARE
16a
HARDWARE-STEUERUNG A
ZU/VON FIG 1E
18a
MECHANISCHES SYSTEM A
20a
BEWEGUNGSSTEUERUNGS-VORRICHTUNG A

FIG. IE

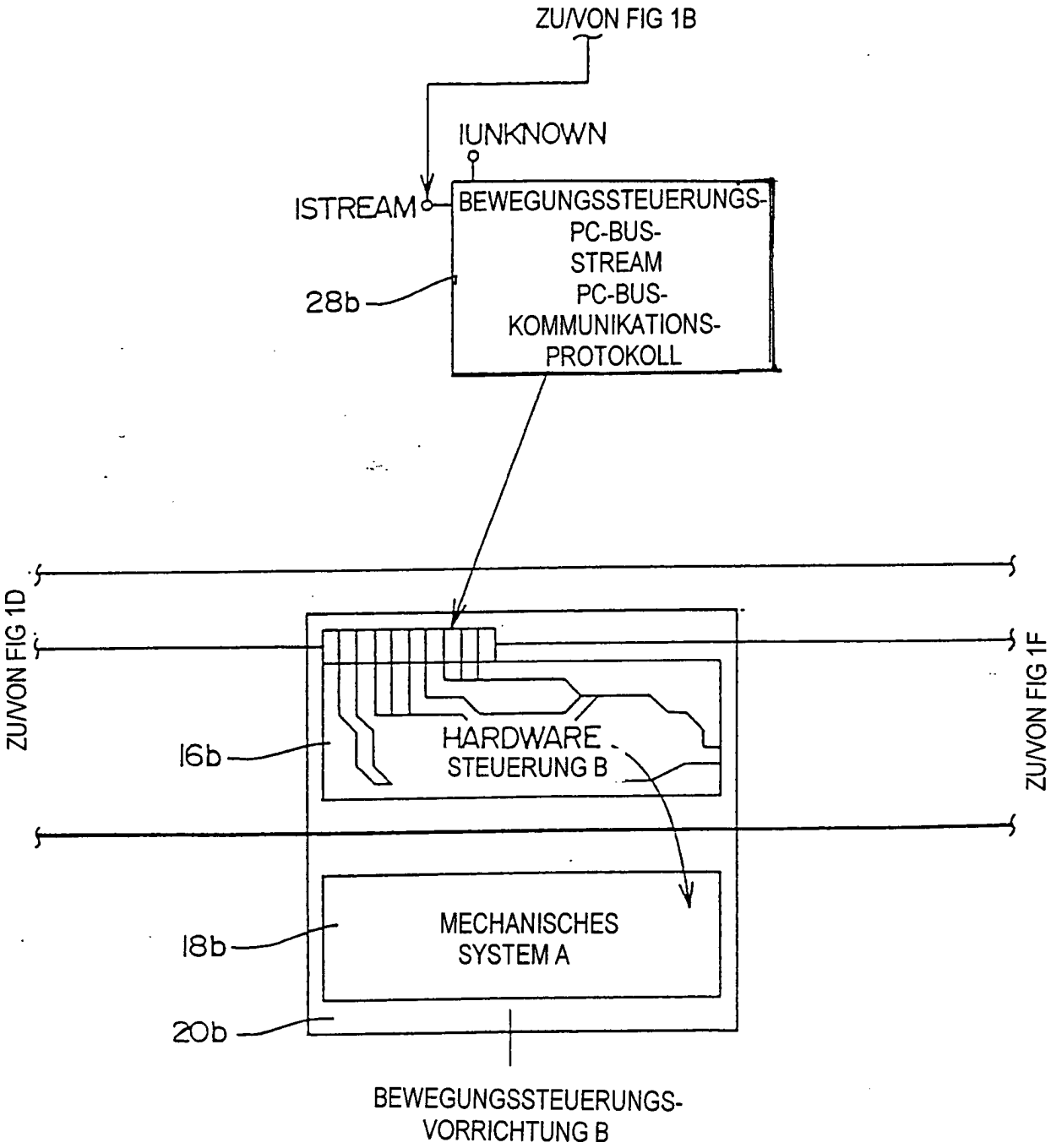

ZU/VON FIG 1B
IUNKNOWN
ISTREAM
BEWEGUNGSSTEUERUNGS-
PC-BUS-
STREAM
PC-BUS-
KOMMUNIKATIONS-
PROTOKOLL
28b
ZU/VON FIG 1D
ZU/VON FIG 1F
16b
HARDWARE
STEUERUNG B
18b
MECHANISCHES
SYSTEM A
20b
BEWEGUNGSSTEUERUNGS-
VORRICHTUNG B

FIG. 1F

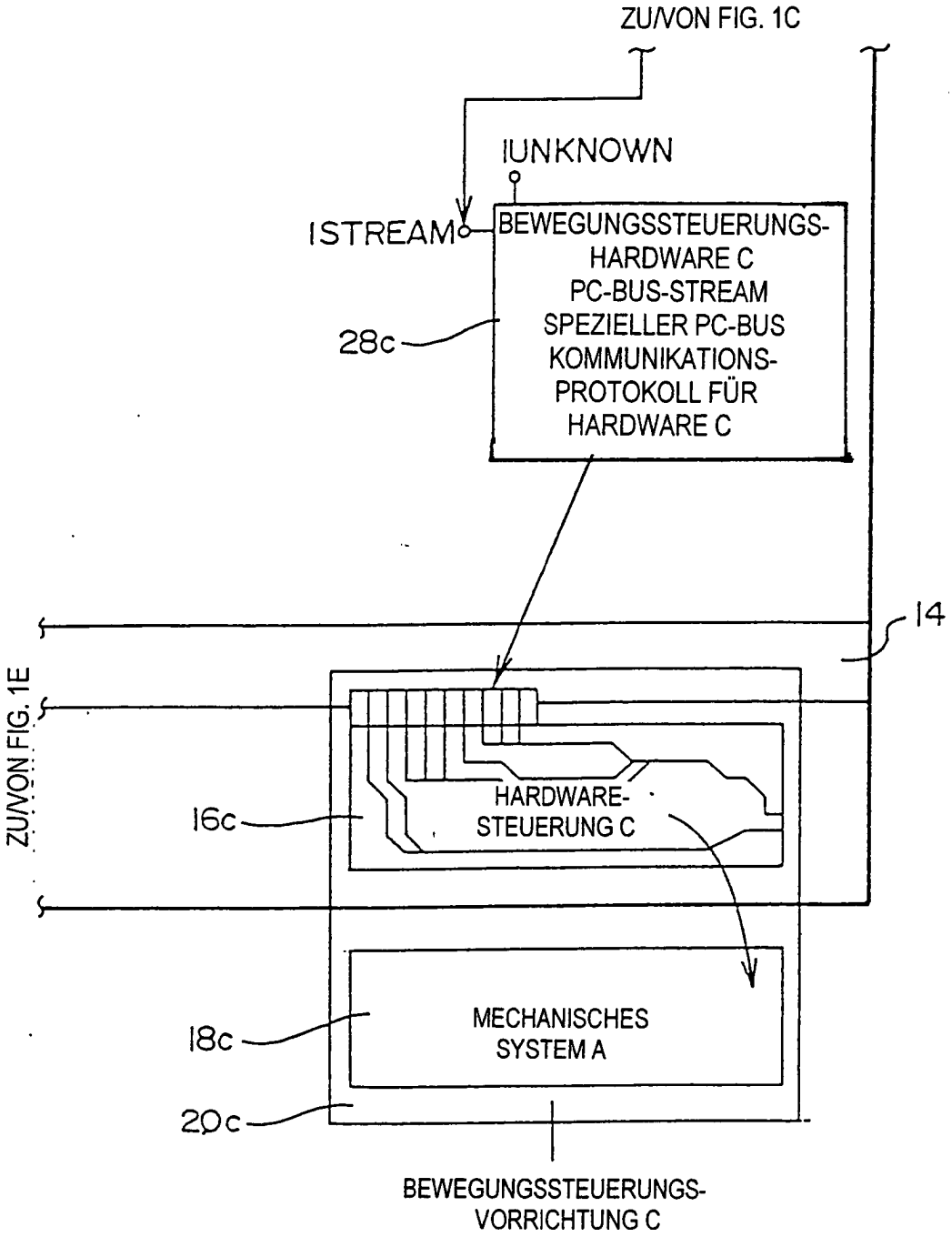

ZU/VON FIG. 1C
IUNKNOWN
ISTREAM
BEWEGUNGSSTEUERUNGS-HARDWARE C
PC-BUS-STREAM
SPEZIELLER PC-BUS KOMMUNIKATIONS-PROTOKOLL FÜR HARDWARE C
28c
14
ZU/VON FIG. 1E
16c
HARDWARE-STEUERUNG C
18c
MECHANISCHES SYSTEM A
20c
BEWEGUNGSSTEUERUNGS-VORRICHTUNG C

FIG. 2

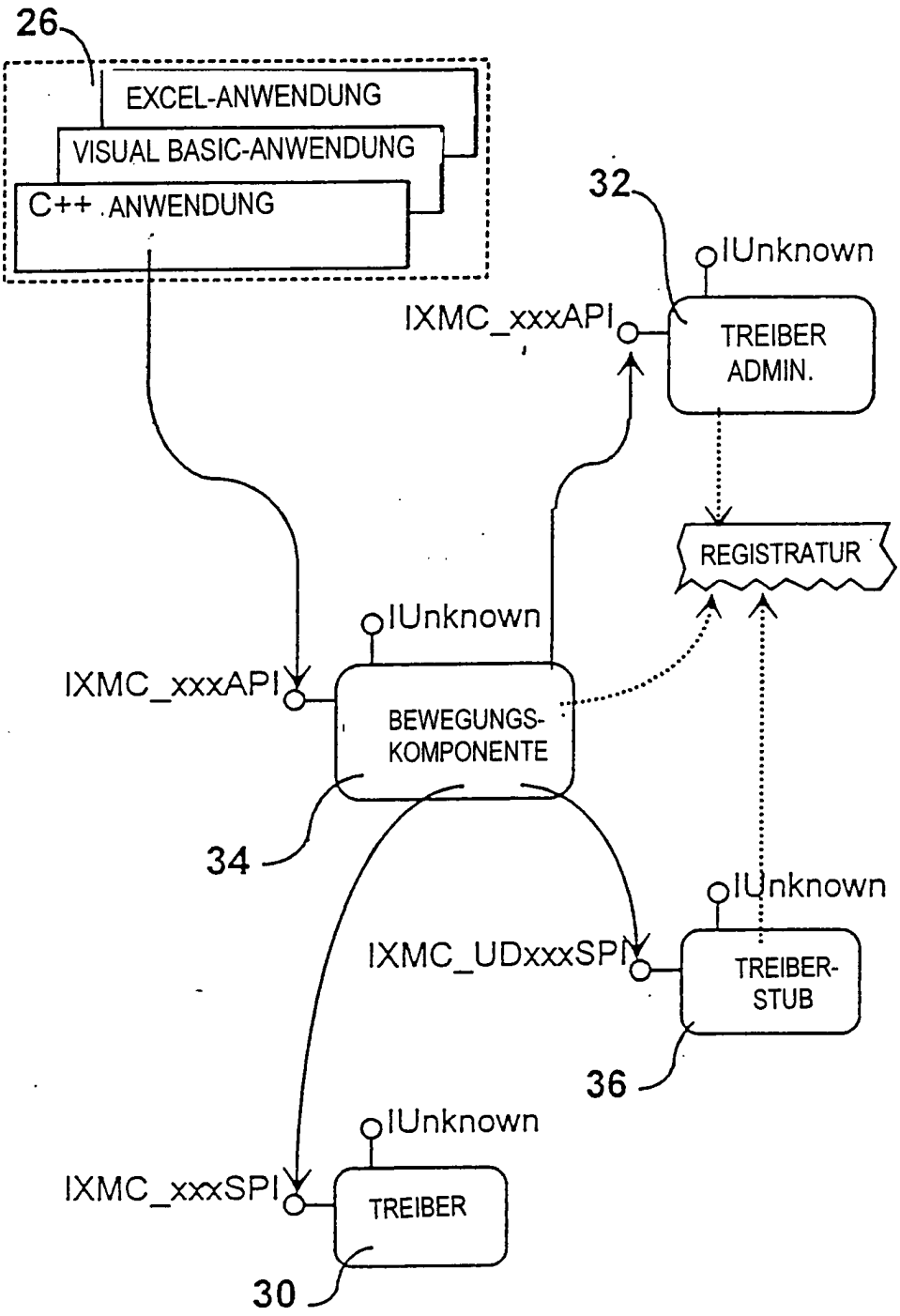

26
EXCEL-ANWENDUNG
VISUAL BASIC-ANWENDUNG
C++ .ANWENDUNG
32
IUnknown
IXMC_xxxAPI
TREIBER ADMIN.
REGISTRATUR
IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
34
IUnknown
IXMC_UDxxxSPI
TREIBER-STUB
36
IUnknown
IXMC_xxxSPI
TREIBER
30

FIG. 3

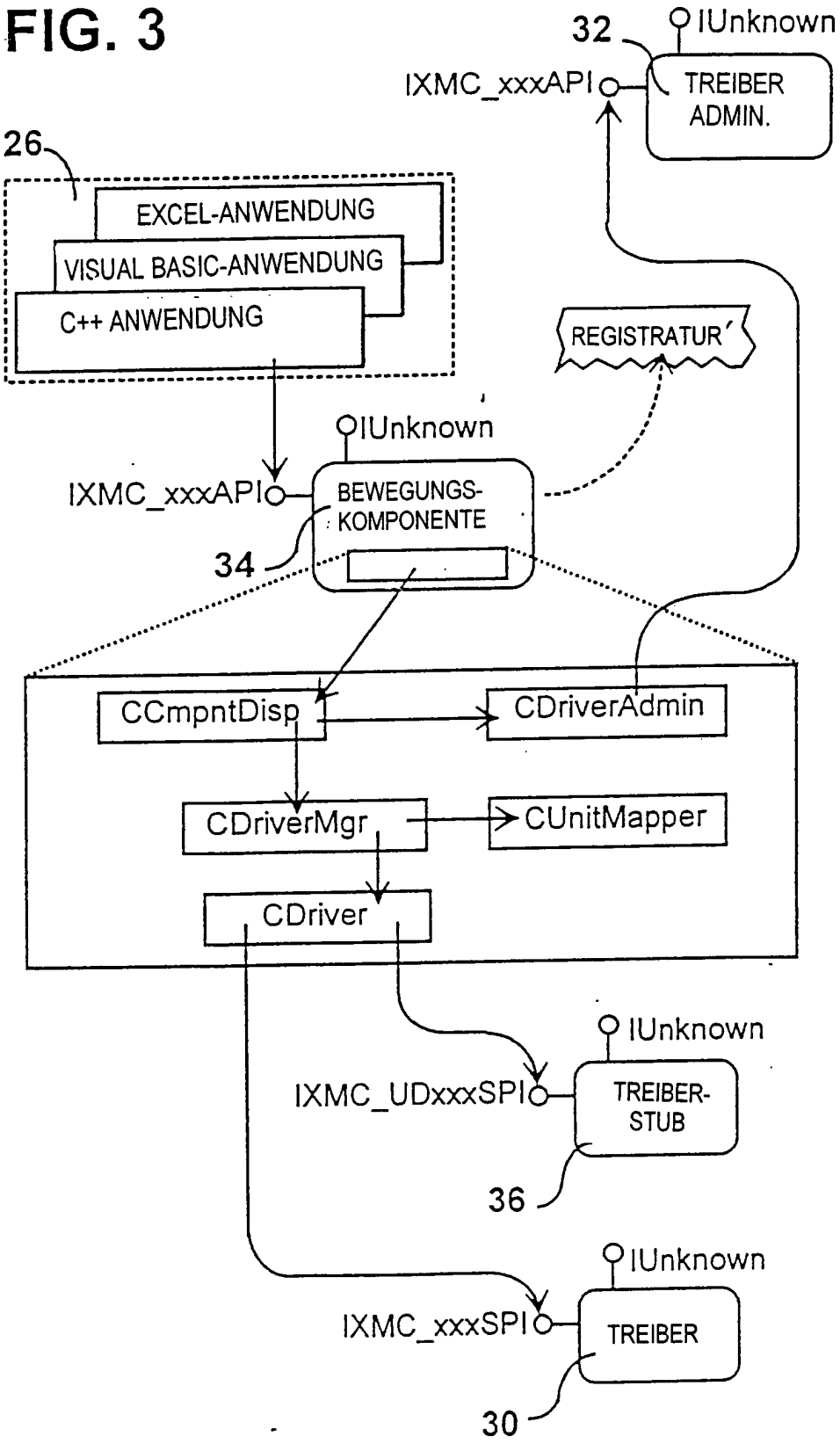

32
IUnknown
IXMC_xxxAPI
TREIBER ADMIN.
26
EXCEL-ANWENDUNG
VISUAL BASIC-ANWENDUNG
C++ ANWENDUNG
REGISTRATUR'
IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
34
CCmpntDisp
CDriverAdmin
CDriverMgr
CUnitMapper
CDriver
IUnknown
IXMC_UDxxxSPI
TREIBER-STUB
36
IUnknown
IXMC_xxxSPI
TREIBER
30

FIG. 4

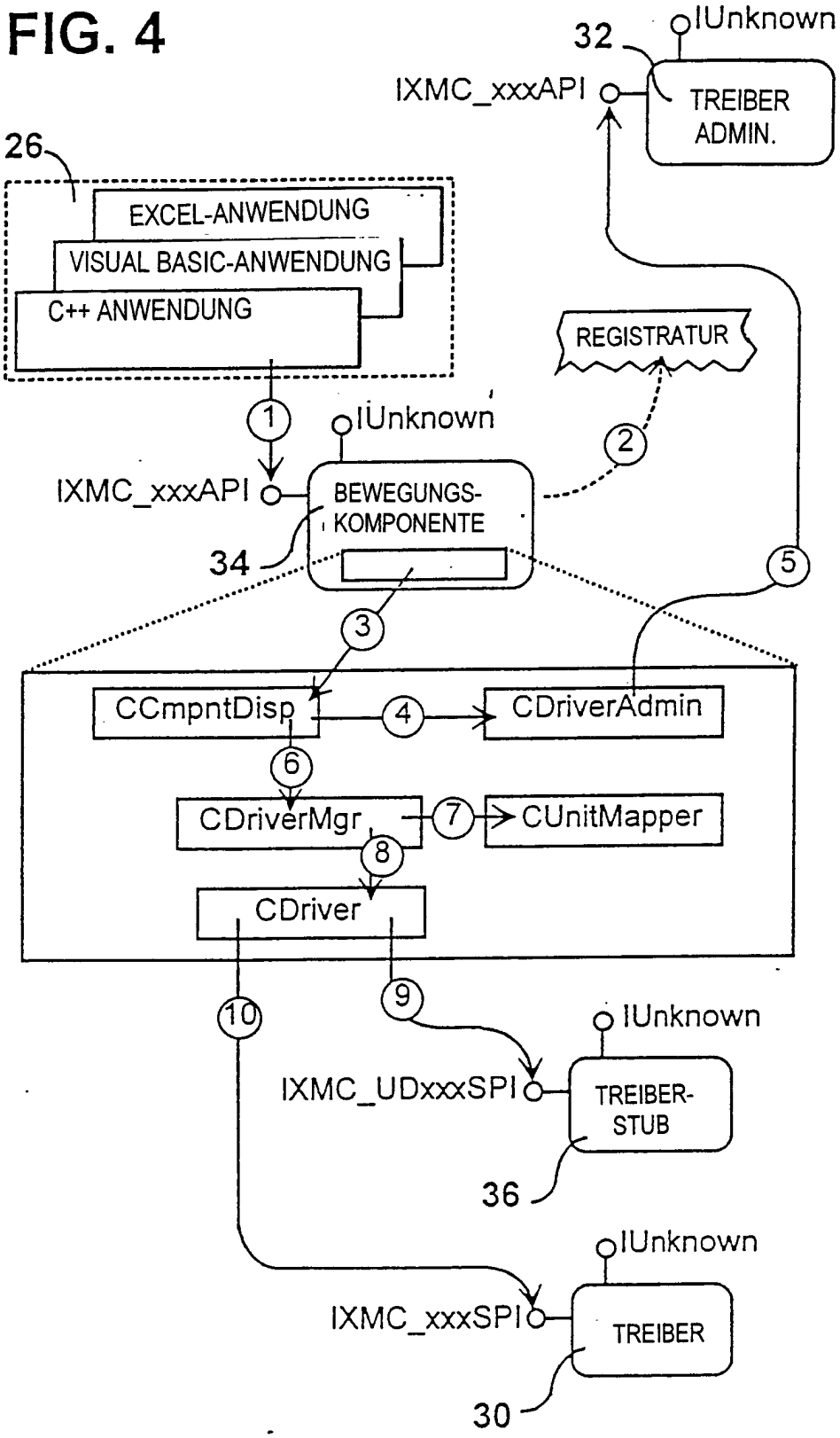

32
IUnknown
IXMC_xxxAPI
TREIBER ADMIN.
26
EXCEL-ANWENDUNG
VISUAL BASIC-ANWENDUNG
C++ ANWENDUNG
REGISTRATUR
1
IUnknown
2
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
34
5
3
CCmpntDisp
4
CDriverAdmin
6
CDriverMgr
7
CUnitMapper
8
CDriver
10
9
IUnknown
IXMC_UDxxxSPI
TREIBER-STUB
36
IUnknown
IXMC_xxxSPI
TREIBER
30

FIG. 5

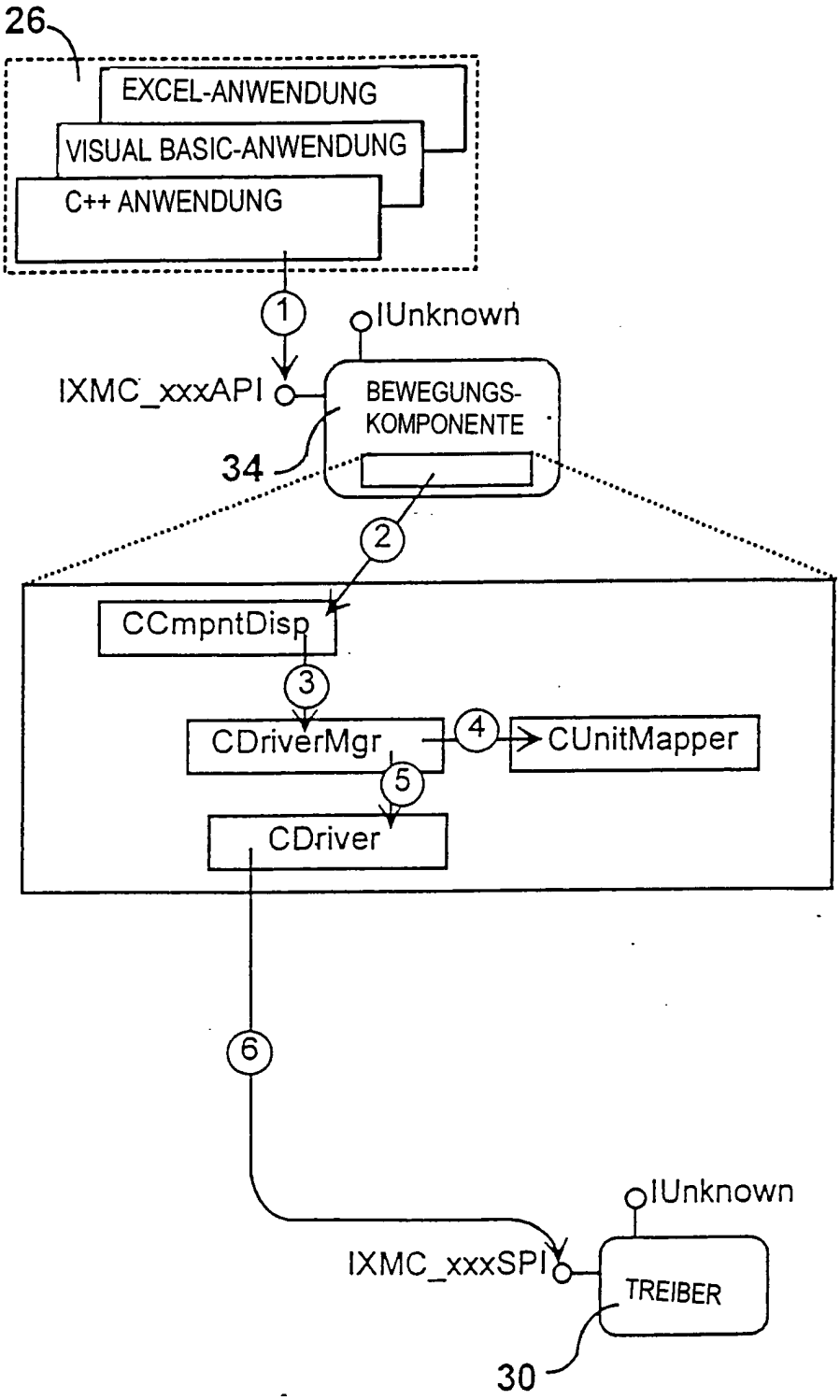

26
EXCEL-ANWENDUNG
VISUAL BASIC-ANWENDUNG
C++ ANWENDUNG
1
IUnknown
IXMC_xxxAPI
BEWEGUNGS-
KOMPONENTE
34
2
CCmpntDisp
3
CDriverMgr
4
CUnitMapper
5
CDriver
6
IUnknown
IXMC_xxxSPI
TREIBER
30

FIG. 6

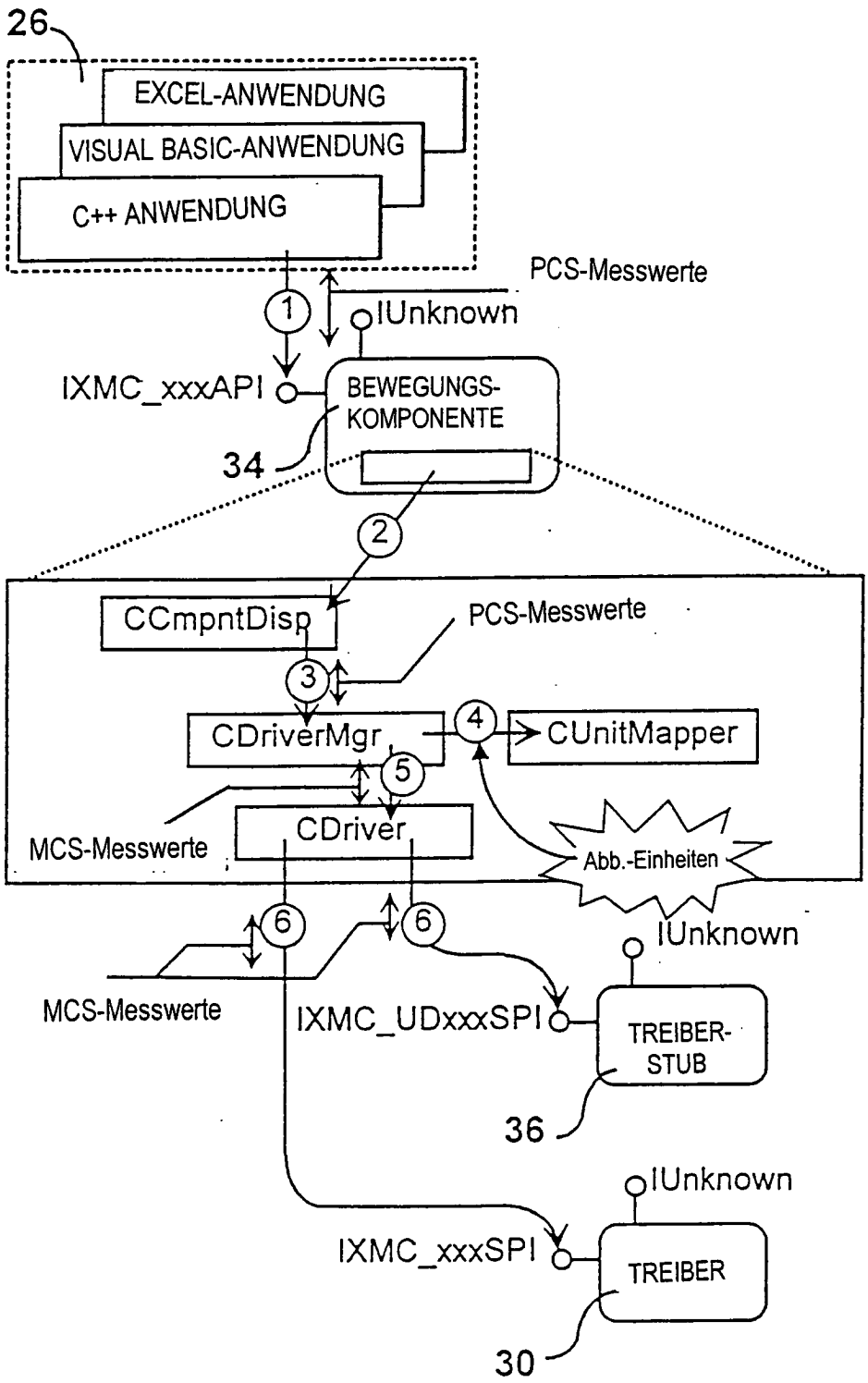

26
EXCEL-ANWENDUNG
VISUAL BASIC-ANWENDUNG
C++ ANWENDUNG
PCS-Messwerte
1
IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
34
2
CCmpntDisp
PCS-Messwerte
3
CDriverMgr
4
CUnitMapper
5
CDriver
MCS-Messwerte
Abb.-Einheiten
6
6
IUnknown
MCS-Messwerte
IXMC_UDxxxSPI
TREIBER-STUB
36
IUnknown
IXMC_xxxSPI
TREIBER
30

FIG. 7

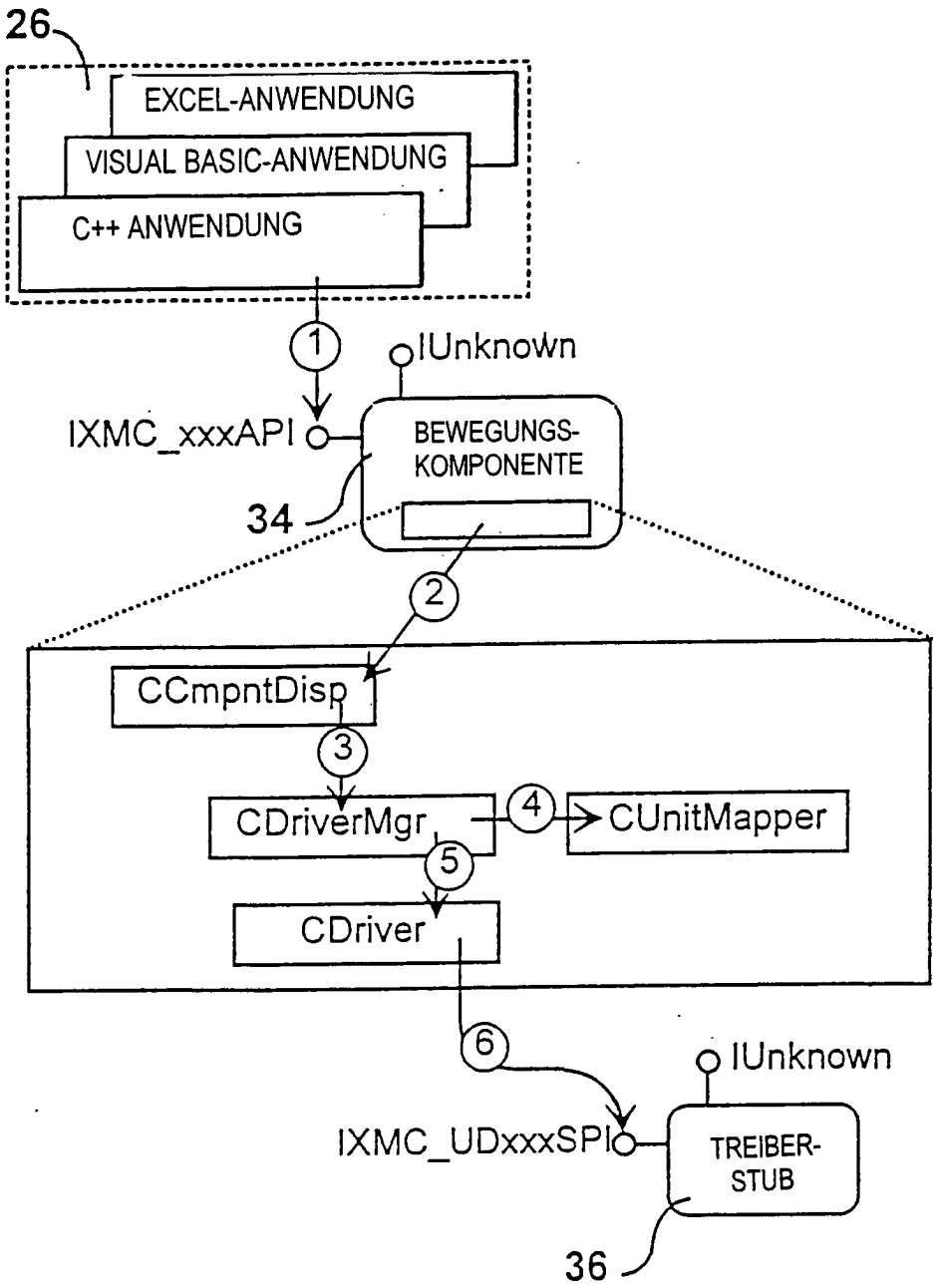

26
EXCEL-ANWENDUNG
VISUAL BASIC-ANWENDUNG
C++ ANWENDUNG
1
IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
34
2
CCmpntDisp
3
CDriverMgr
4
CUnitMapper
5
CDriver
6
IUnknown
IXMC_UDxxxSPI
TREIBER-STUB
36

FIG. 8

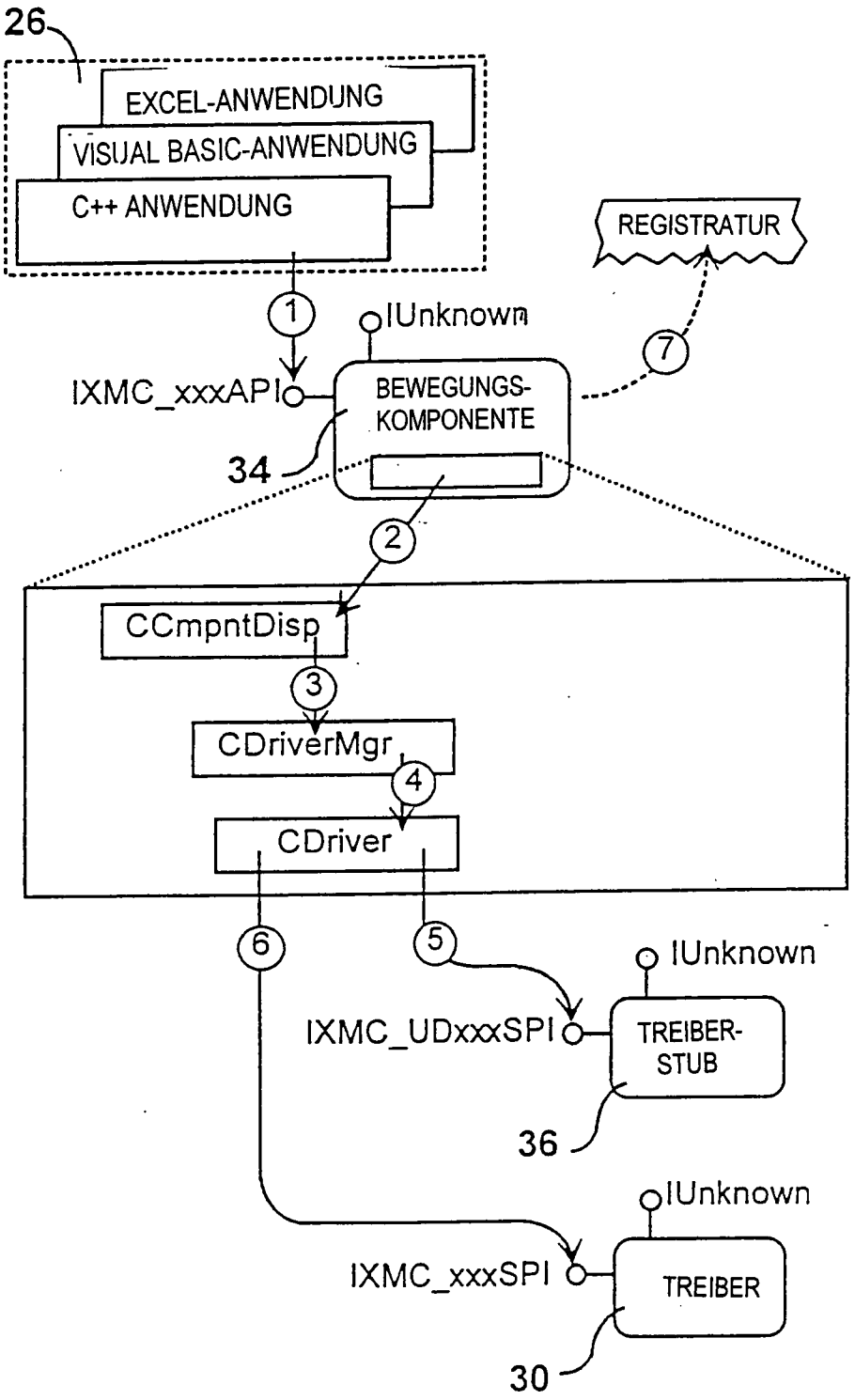

26
EXCEL-ANWENDUNG
VISUAL BASIC-ANWENDUNG
C++ ANWENDUNG
REGISTRATUR
1
IUnknown
IXMC_xxxAPI
BEWEGUNGS-
KOMPONENTE
7
34
2
CCmpntDisp
3
CDriverMgr
4
CDriver
6
5
IUnknown
IXMC_UDxxxSPI
TREIBER-
STUB
36
IUnknown
IXMC_xxxSPI
TREIBER
30

FIG. 9

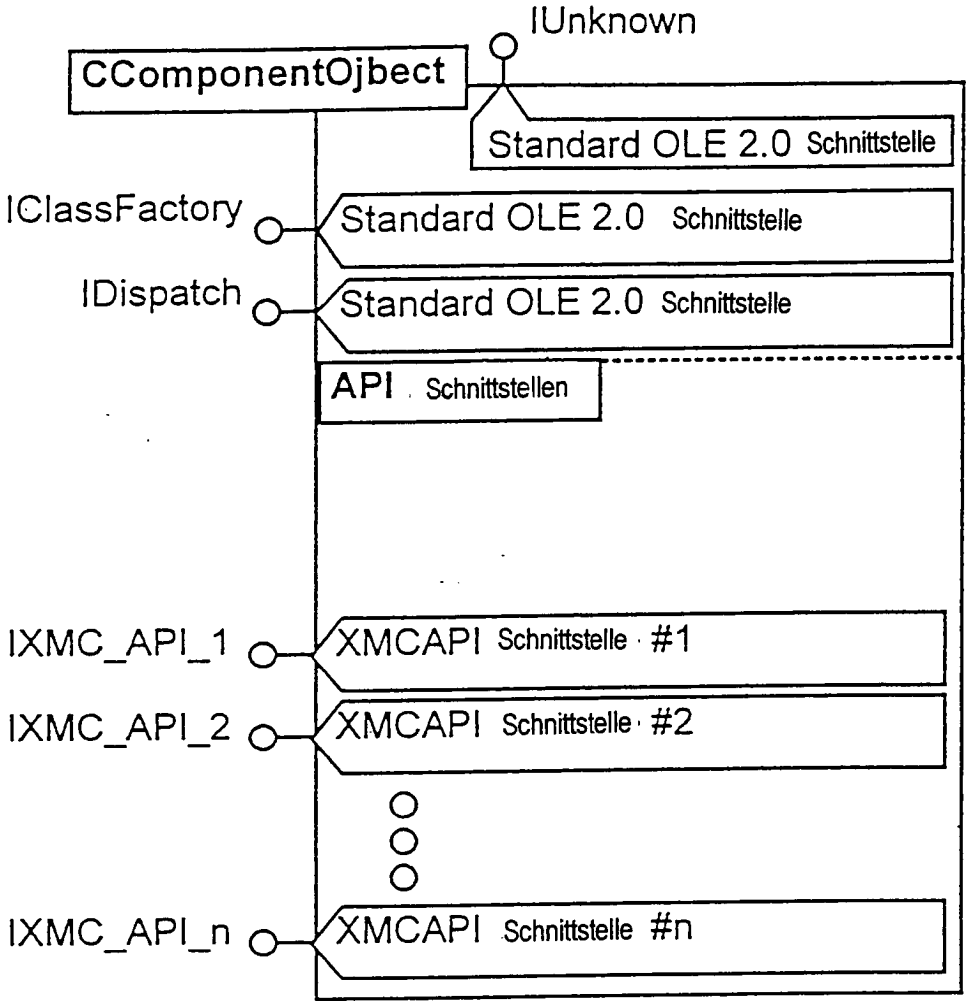

IUnknown
CComponentOjbect
Standard OLE 2.0 Schnittstelle
IClassFactory
Standard OLE 2.0 Schnittstelle
IDispatch
Standard OLE 2.0 Schnittstelle
API Schnittstellen
IXMC_API_1
XMCAPI Schnittstelle #1
IXMC_API_2
XMCAPI Schnittstelle #2
IXMC_API_n
XMCAPI Schnittstelle #n

FIG. 10

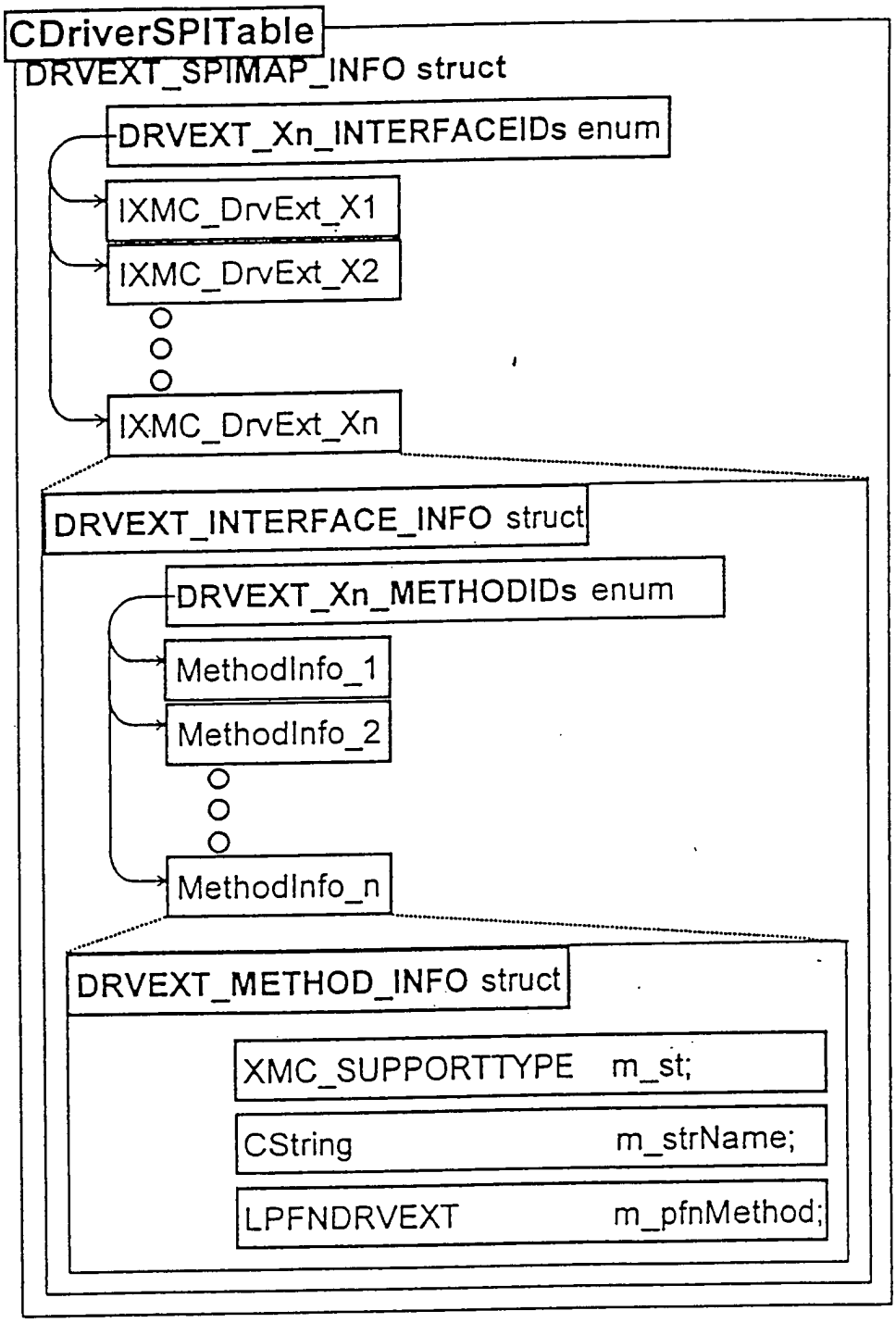

CDriverSPITable
DRVEXT_SPIMAP_INFO struct
DRVEXT_Xn_INTERFACEIDs enum
IXMC_DrvExt_X1
IXMC_DrvExt_X2
IXMC_DrvExt_Xn
DRVEXT_INTERFACE_INFO struct
DRVEXT_Xn_METHODIDs enum
MethodInfo_1
MethodInfo_2
MethodInfo_n
DRVEXT_METHOD_INFO struct
XMC_SUPPORTTYPE m_st;
CString m_strName;
LPFNDRVEXT m_pfnMethod;

FIG. 11

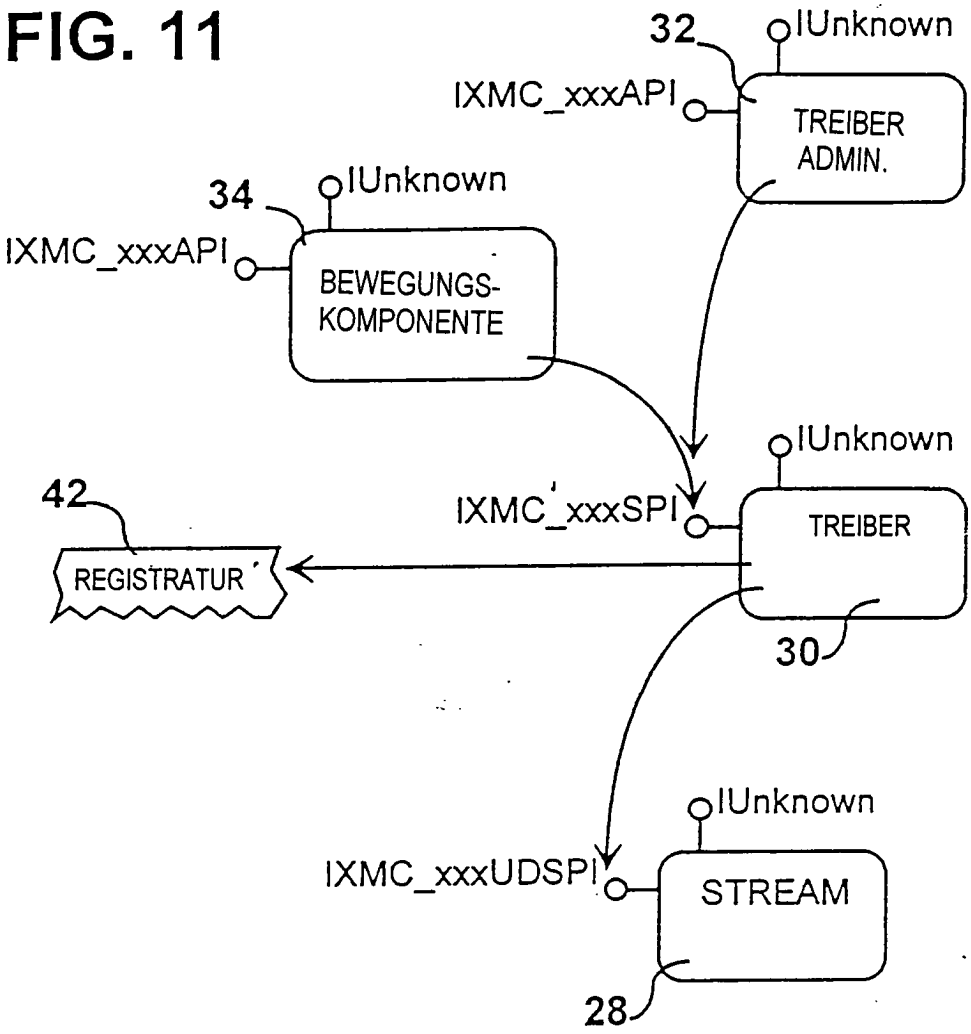

32
IUnknown
IXMC_xxxAPI
TREIBER ADMIN.
34
IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
IUnknown
42
IXMC_xxxSPI
TREIBER
REGISTRATUR
30
IUnknown
IXMC_xxxUDSPI
STREAM
28

FIG. 12

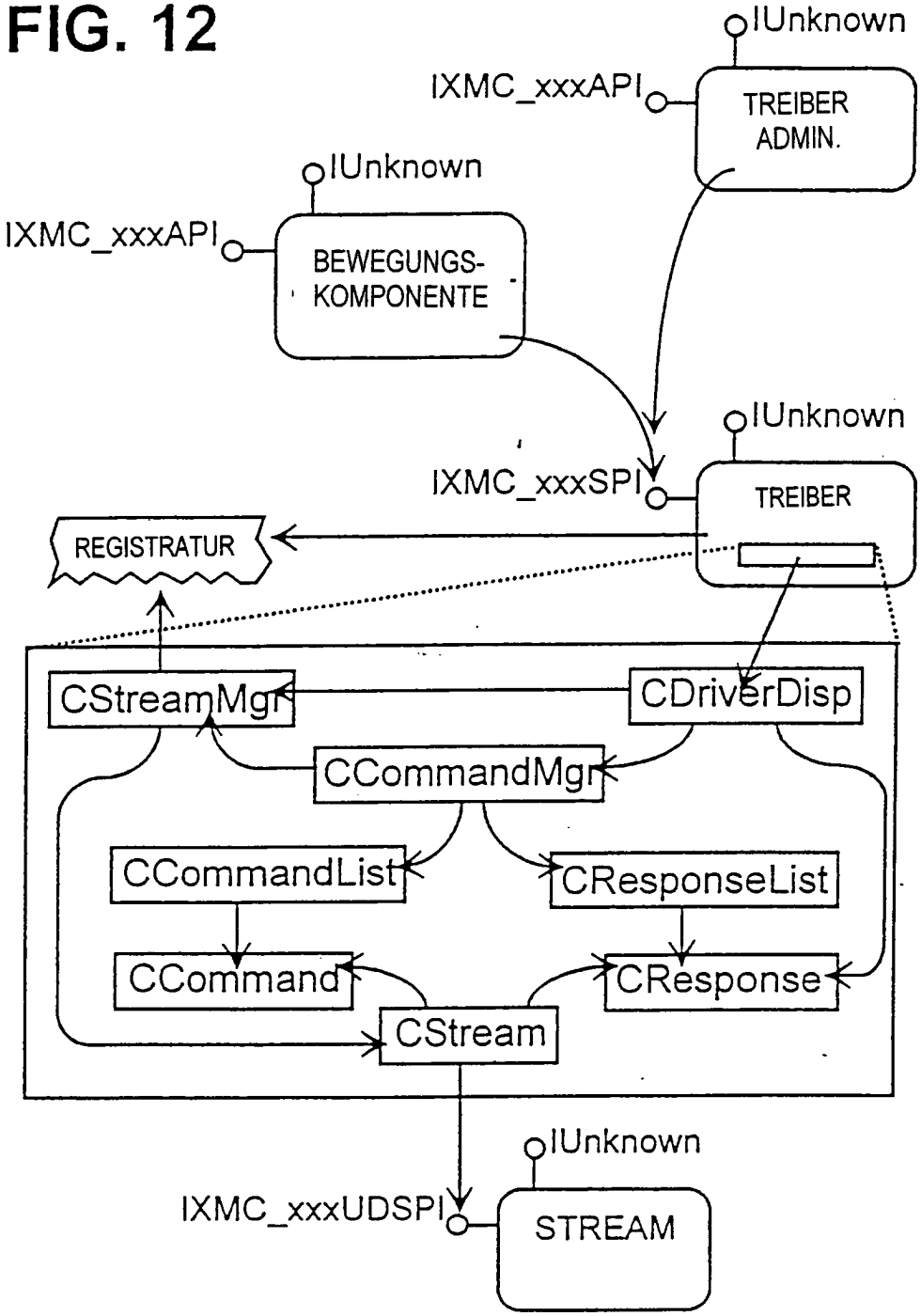

IUnknown
IXMC_xxxAPI
TREIBER ADMIN.
IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
IUnknown
IXMC_xxxSPI
TREIBER
REGISTRATUR
CStreamMgr
CDriverDisp
CCommandMgr
CCommandList
CResponseList
CCommand
CResponse
CStream
IUnknown
IXMC_xxxUDSPI
STREAM

FIG. 13

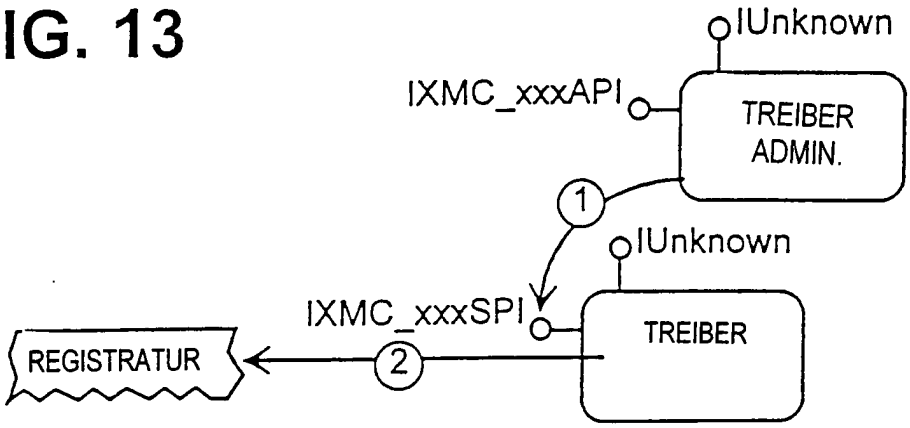

IUnknown
IXMC_xxxAPI
TREIBER ADMIN.
1
IUnknown
IXMC_xxxSPI
TREIBER
REGISTRATUR
2

FIG. 14

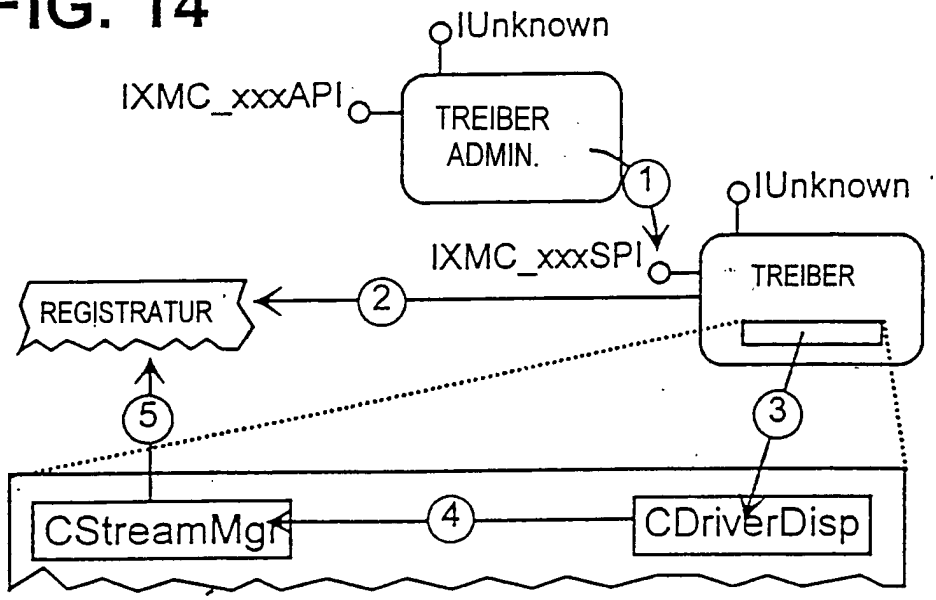

IUnknown
IXMC_xxxAPI
TREIBER ADMIN.
1
IUnknown
IXMC_xxxSPI
TREIBER
REGISTRATUR
2
5
3
CStreamMgr
4
CDriverDisp

FIG. 15

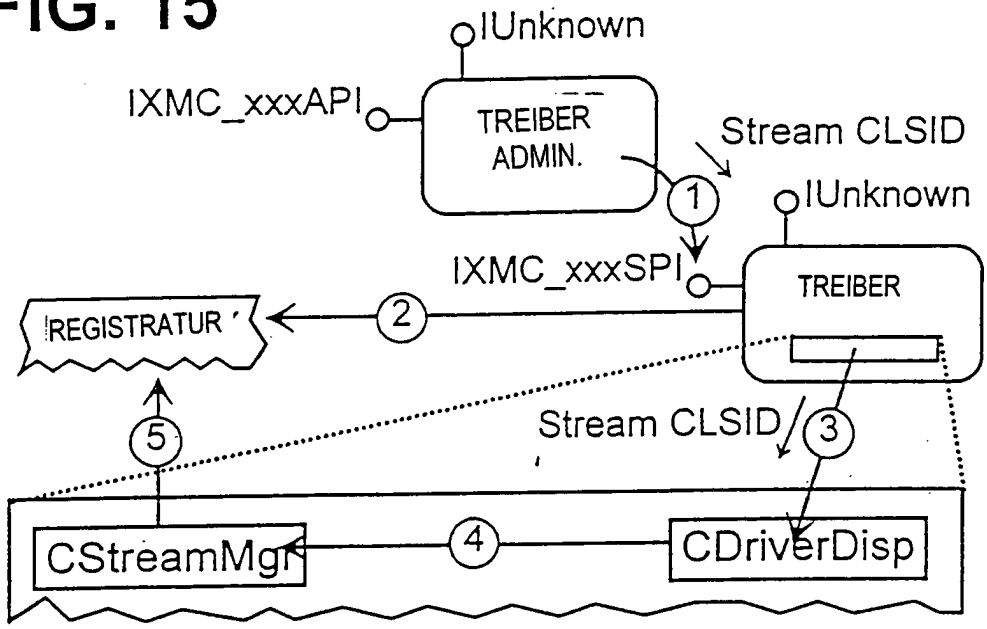

IUnknown
IXMC_xxxAPI
TREIBER ADMIN.
Stream CLSID
1
IUnknown
IXMC_xxxSPI
TREIBER
REGISTRATUR
2
5
Stream CLSID
3
CStreamMgr
4
CDriverDisp

FIG. 16

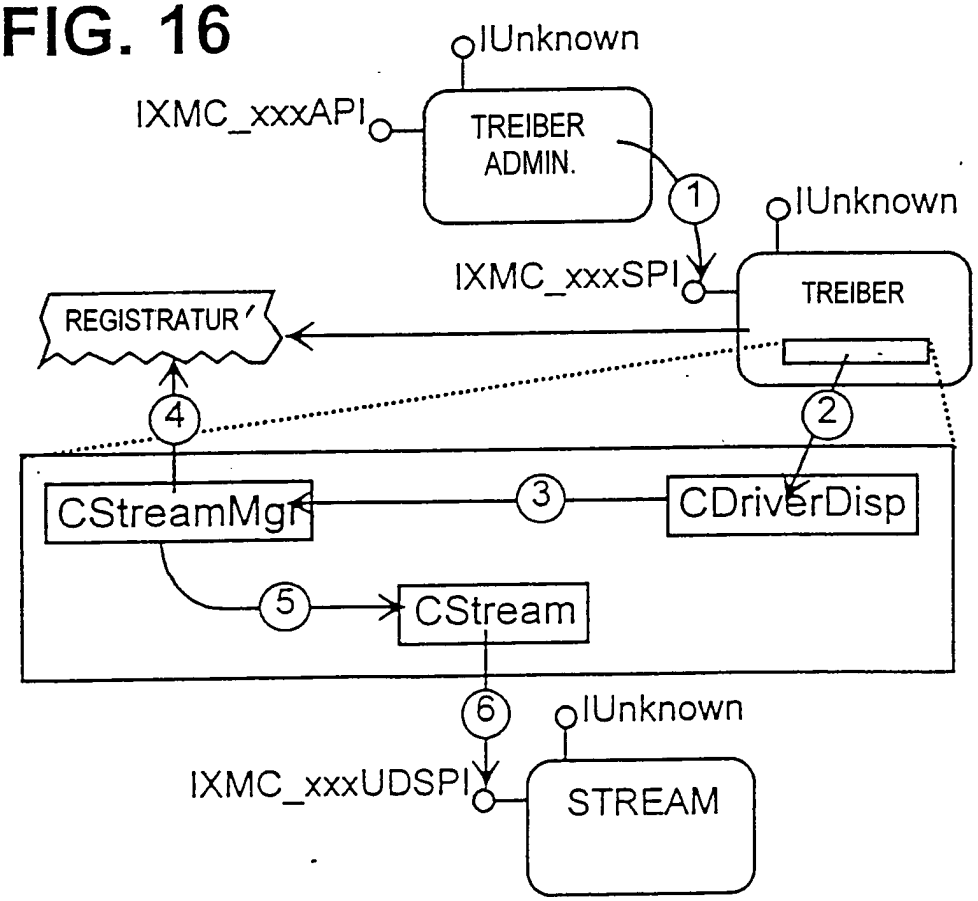

IUnknown
IXMC_xxxAPI
TREIBER ADMIN.
1
IUnknown
IXMC_xxxSPI
TREIBER
REGISTRATUR
4
2
CStreamMgr
3
CDriverDisp
5
CStream
6
IUnknown
IXMC_xxxUDSPI
STREAM

FIG. 17

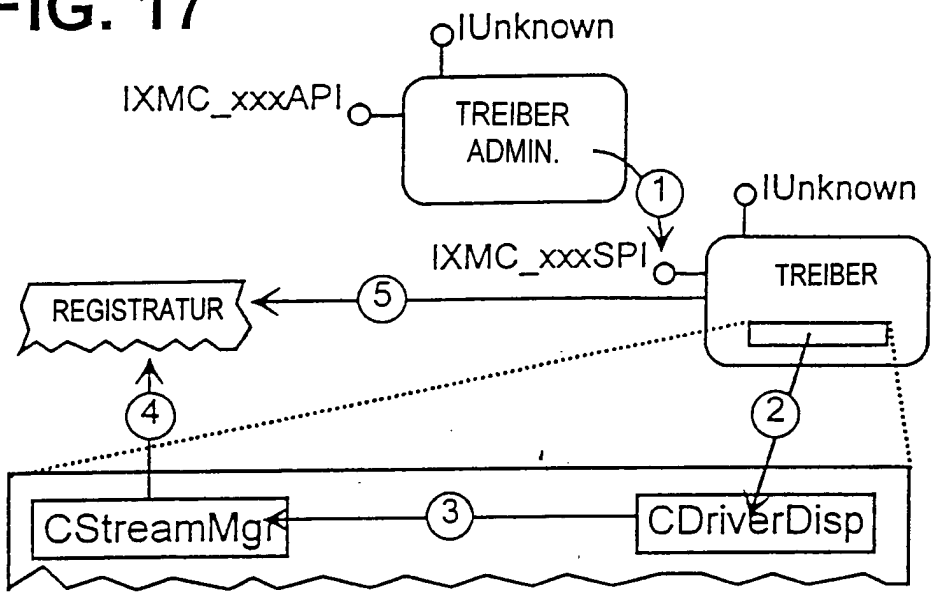

IUnknown
IXMC_xxxAPI
TREIBER ADMIN.
1
IUnknown
IXMC_xxxSPI
TREIBER
REGISTRATUR
5
4
2
CStreamMgr
3
CDriverDisp

FIG. 18

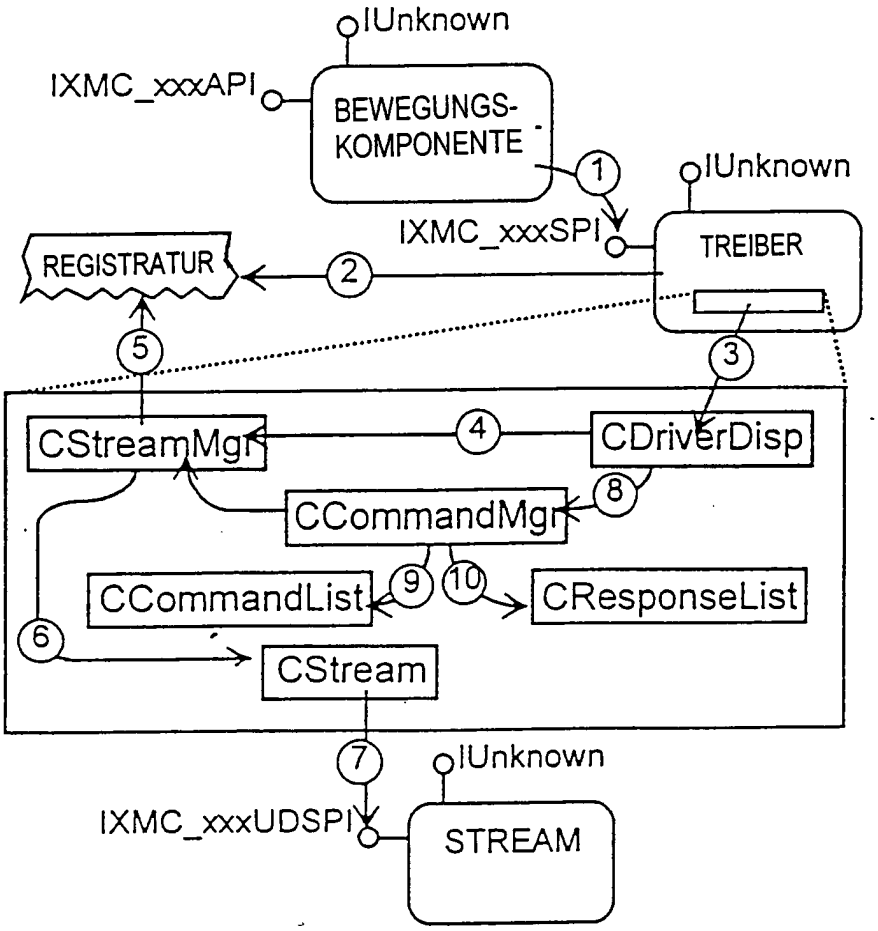

IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
1
IUnknown
IXMC_xxxSPI
TREIBER
REGISTRATUR
2
5
3
CStreamMgr
4
CDriverDisp
8
CCommandMgr
CCommandList
9
10
CResponseList
6
CStream
7
IUnknown
IXMC_xxxUDSPI
STREAM

FIG. 19

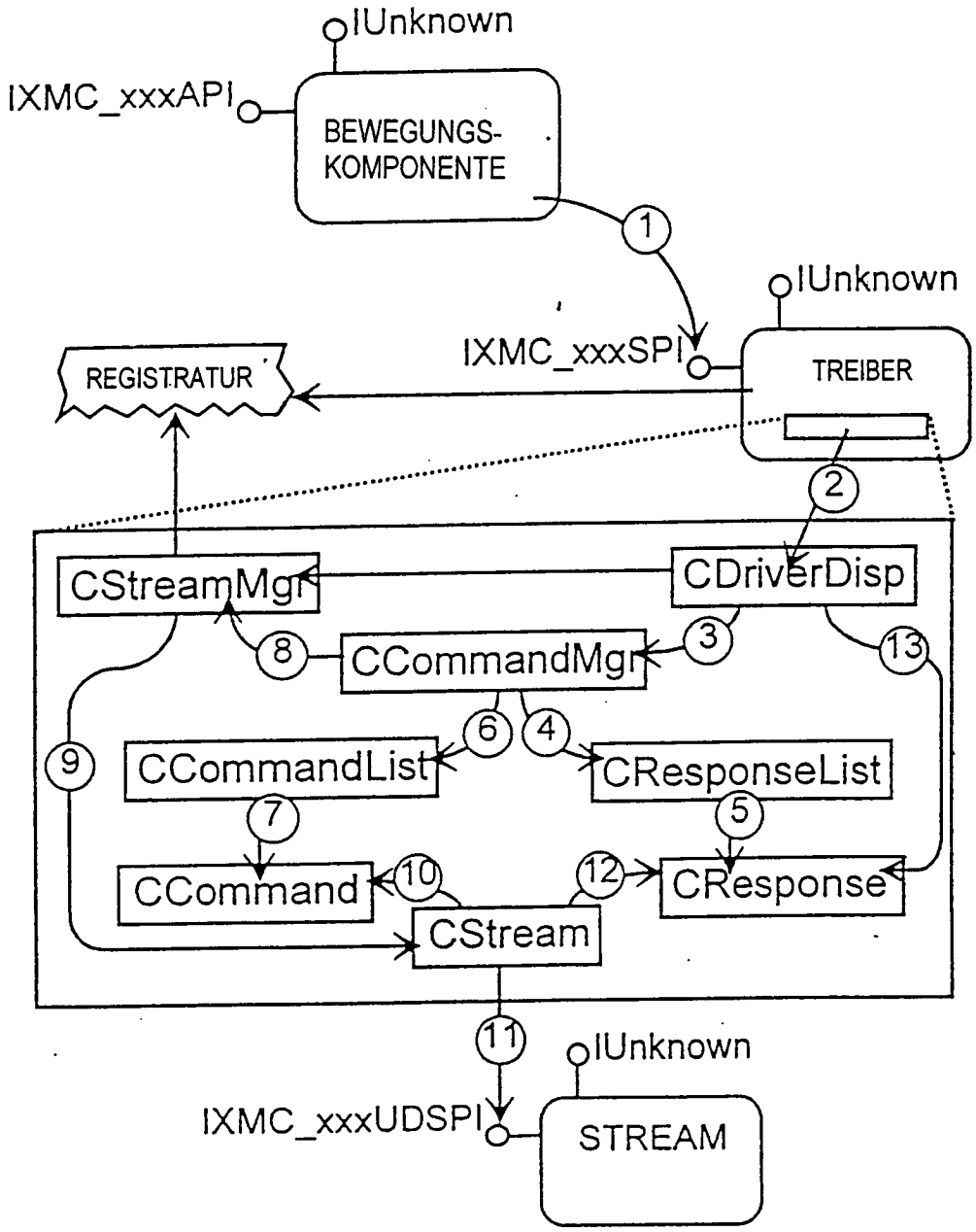

IUnknown
IXMC_xxxAPI
BEWEGUNGS-
KOMPONENTE
1
IUnknown
IXMC_xxxSPI
TREIBER
REGISTRATUR
2
CStreamMgr
CDriverDisp
8
CCommandMgr
3
13
6
4
9
CCommandList
CResponseList
7
5
CCommand
10
12
CResponse
CStream
11
IUnknown
IXMC_xxxUDSPI
STREAM

FIG. 20

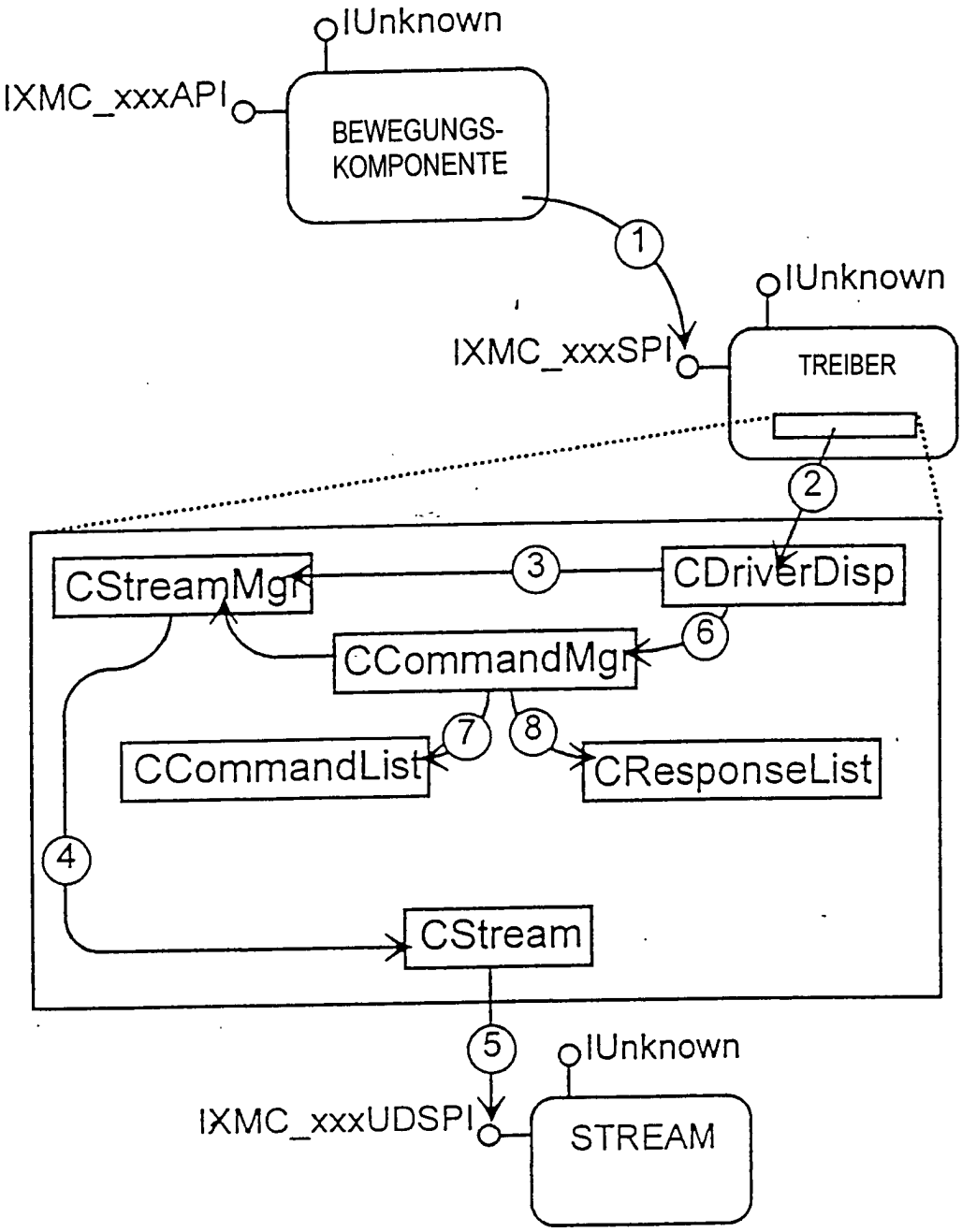

IUnknown
IXMC_xxxAPI
BEWEGUNGS-
KOMPONENTE
1
IUnknown
IXMC_xxxSPI
TREIBER
2
CStreamMgr
3
CDriverDisp
6
CCommandMgr
7
8
CCommandList
CResponseList
4
CStream
5
IUnknown
IXMC_xxxUDSPI
STREAM

FIG. 21

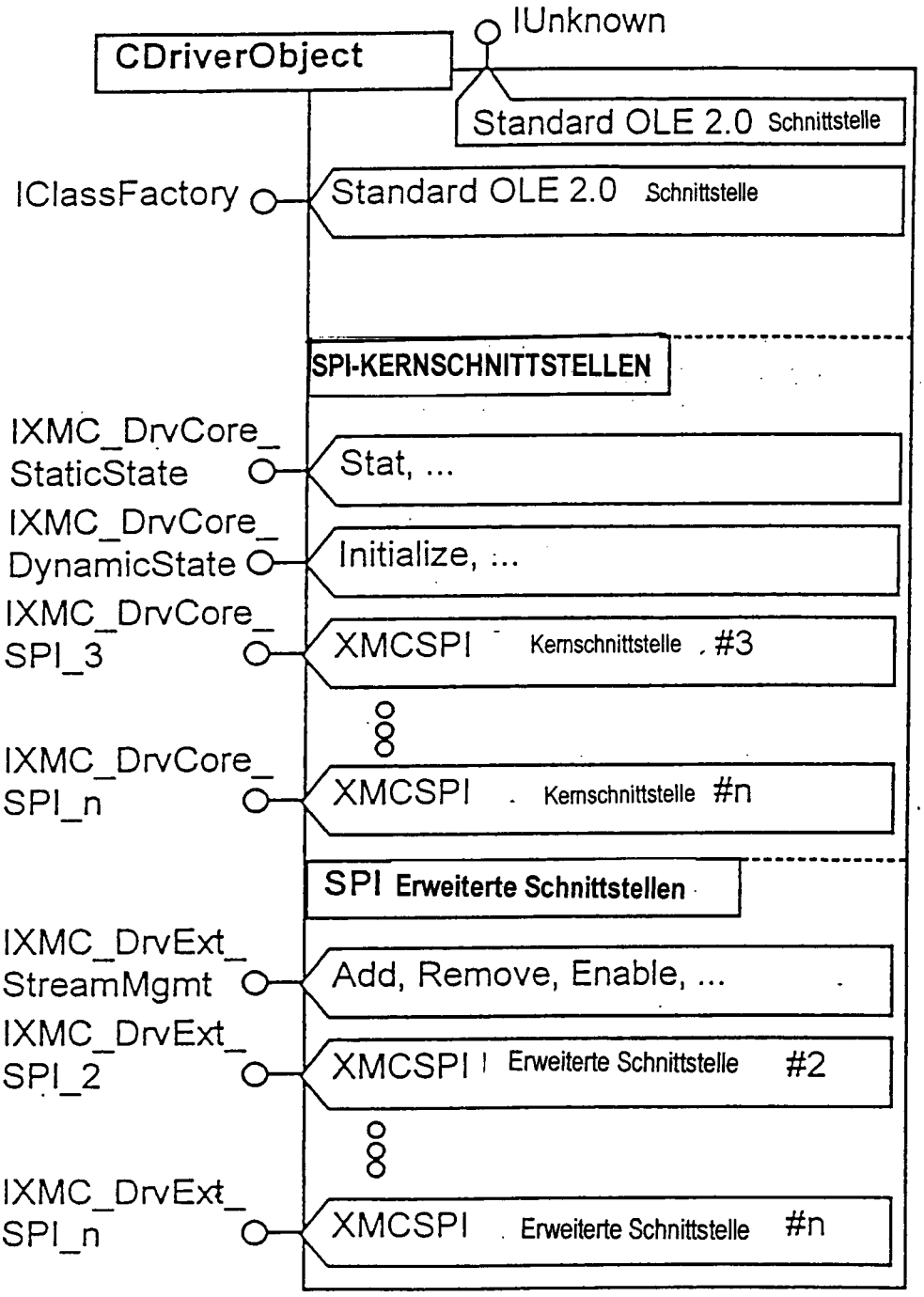

IUnknown
CDriverObject
Standard OLE 2.0 Schnittstelle
IClassFactory
Standard OLE 2.0 Schnittstelle
SPI-KERNSCHNITTSTELLEN
IXMC_DrvCore_
StaticState
Stat, ...
IXMC_DrvCore_
DynamicState
Initialize, ...
IXMC_DrvCore_
SPI_3
XMCSPI Kernschnittstelle #3
IXMC_DrvCore_
SPI_n
XMCSPI Kernschnittstelle #n
SPI Erweiterte Schnittstellen
IXMC_DrvExt_
StreamMgmt
Add, Remove, Enable, ...
IXMC_DrvExt_
SPI_2
XMCSPI Erweiterte Schnittstelle #2
IXMC_DrvExt_
SPI_n
XMCSPI Erweiterte Schnittstelle #n

FIG. 22

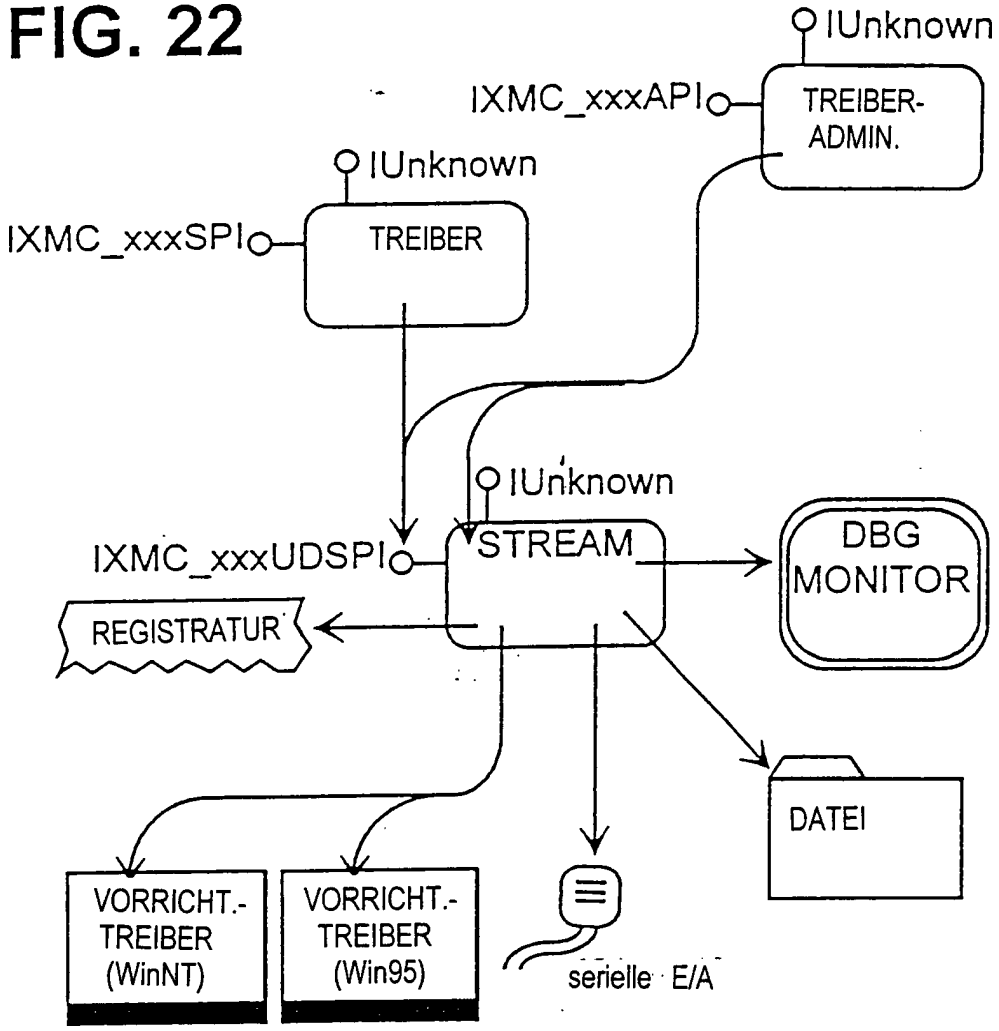

IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.
IUnknown
IXMC_xxxSPI
TREIBER
IUnknown
IXMC_xxxUDSPI
STREAM
DBG
MONITOR
REGISTRATUR
DATEI
VORRICHT.-
TREIBER
(WinNT)
VORRICHT.-
TREIBER
(Win95)
serielle E/A

FIG. 23

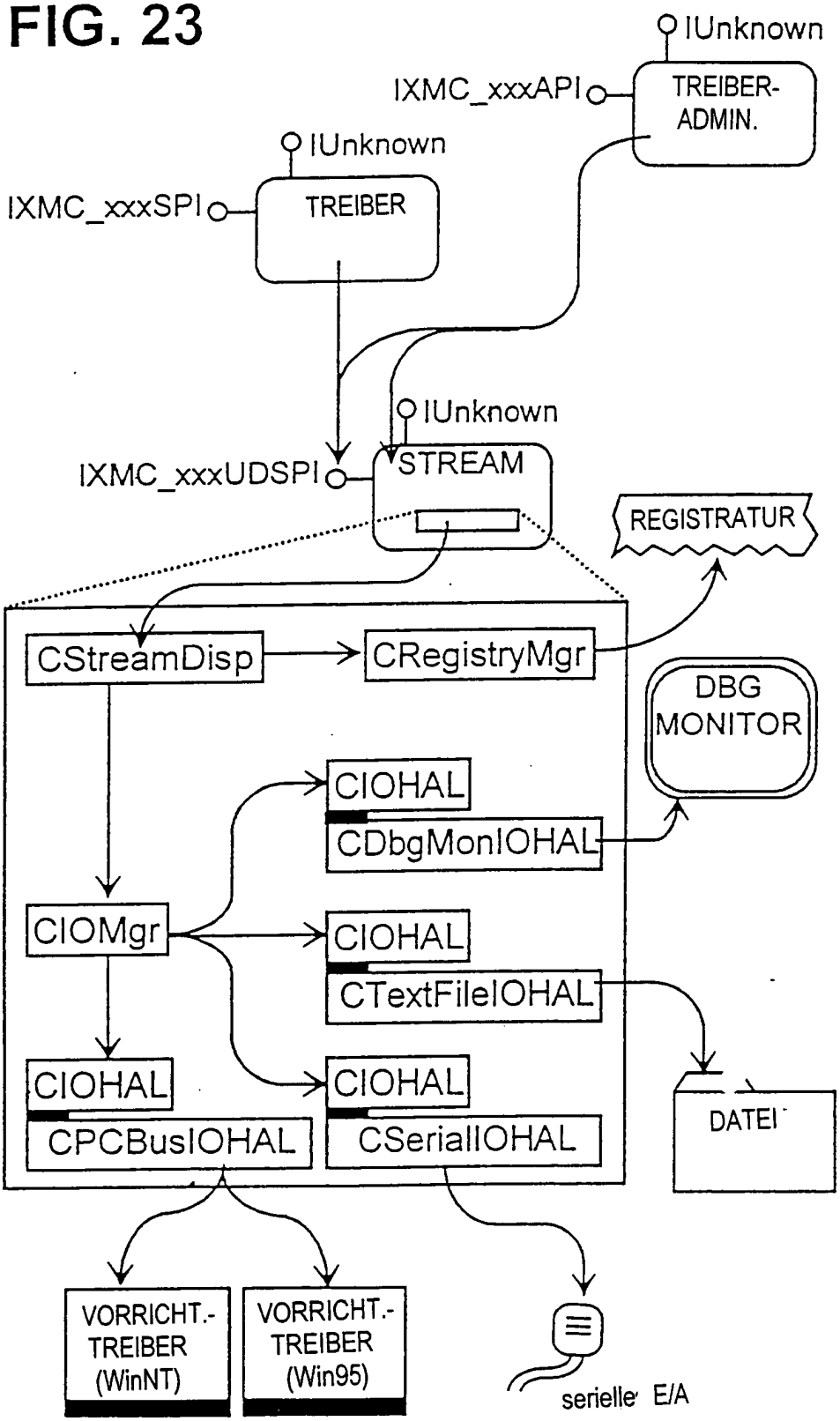

IUnknown
IXMC_xxxAPI
TREIBER-ADMIN.
IUnknown
IXMC_xxxSPI
TREIBER
IUnknown
IXMC_xxxUDSPI
STREAM
REGISTRATUR
CStreamDisp
CRegistryMgr
DBG MONITOR
CIOHAL
CDbgMonIOHAL
CIOMgr
CIOHAL
CTextFileIOHAL
CIOHAL
CIOHAL
CPCBusIOHAL
CSerialIOHAL
DATEI
VORRICHT.-TREIBER (WinNT)
VORRICHT.-TREIBER (Win95)
serielle E/A

FIG. 24

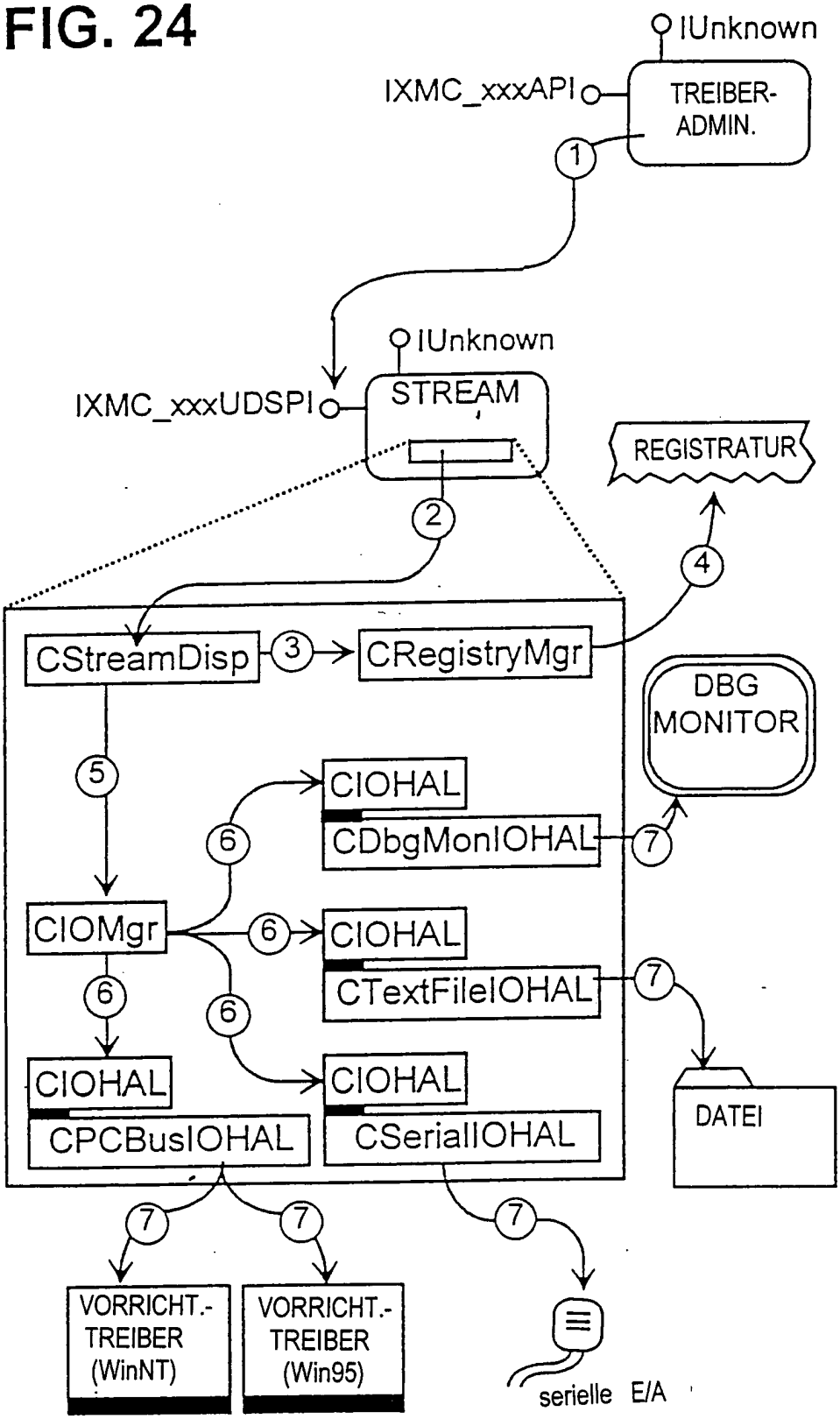

IUnknown
IXMC_xxxAPI
TREIBER-ADMIN.
1
IUnknown
IXMC_xxxUDSPI
STREAM
REGISTRATUR
2
4
CStreamDisp
3
CRegistryMgr
DBG MONITOR
5
CIOHAL
6
CDbgMonIOHAL
7
CIOMgr
6
CIOHAL
6
CTextFileIOHAL
7
6
CIOHAL
CIOHAL
CPCBusIOHAL
CSerialIOHAL
DATEI
7
7
7
VORRICHT.-TREIBER (WinNT)
VORRICHT.-TREIBER (Win95)
serielle E/A

FIG. 25

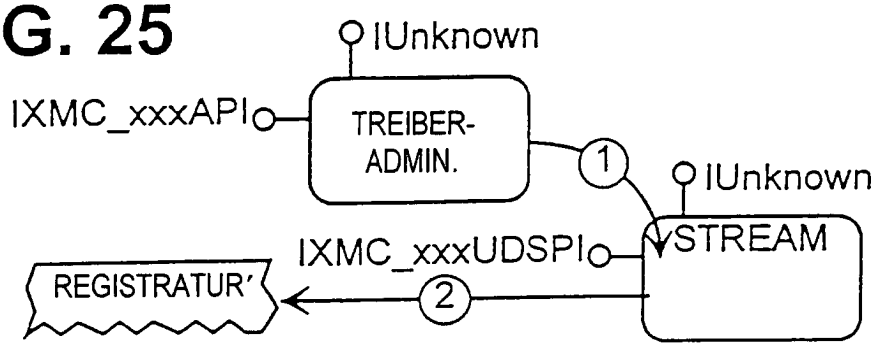

IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.
1
IUnknown
STREAM
IXMC_xxxUDSPI
REGISTRATUR'
2

FIG. 26

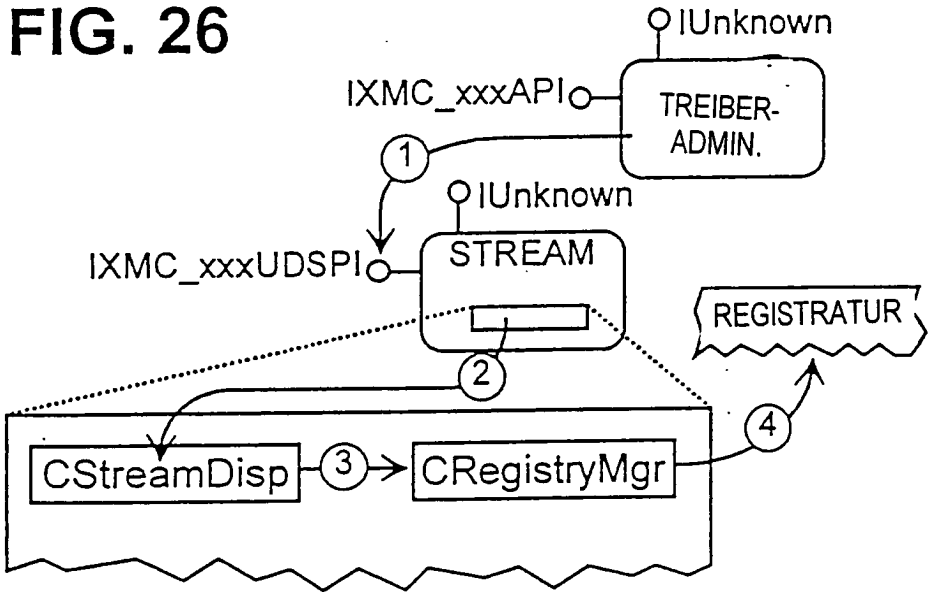

IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.
1
IUnknown
IXMC_xxxUDSPI
STREAM
REGISTRATUR
2
4
CStreamDisp
3
CRegistryMgr

FIG. 27

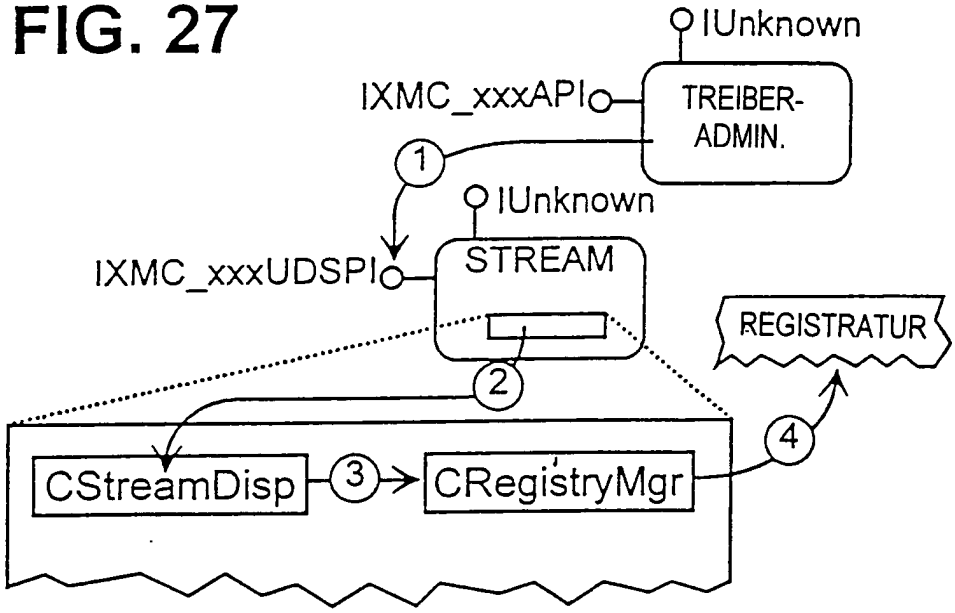

IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.
1
IUnknown
IXMC_xxxUDSPI
STREAM
REGISTRATUR
2
4
CStreamDisp
3
CRegistryMgr

FIG. 28

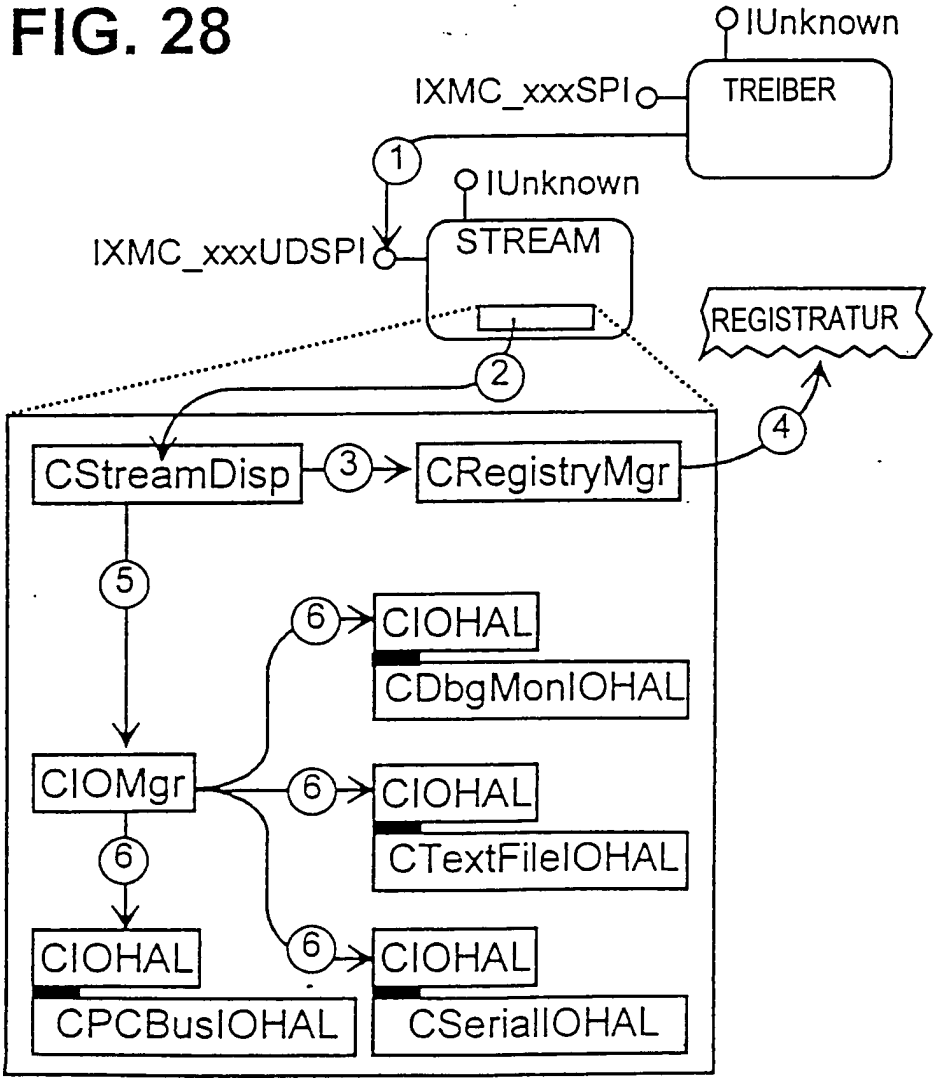

IUnknown
IXMC_xxxSPI
TREIBER
1
IUnknown
IXMC_xxxUDSPI
STREAM
REGISTRATUR
2
4
CStreamDisp
3
CRegistryMgr
5
6
CIOHAL
CDbgMonIOHAL
CIOMgr
6
CIOHAL
CTextFileIOHAL
6
CIOHAL
CIOHAL
CPCBusIOHAL
CSerialIOHAL

FIG. 29

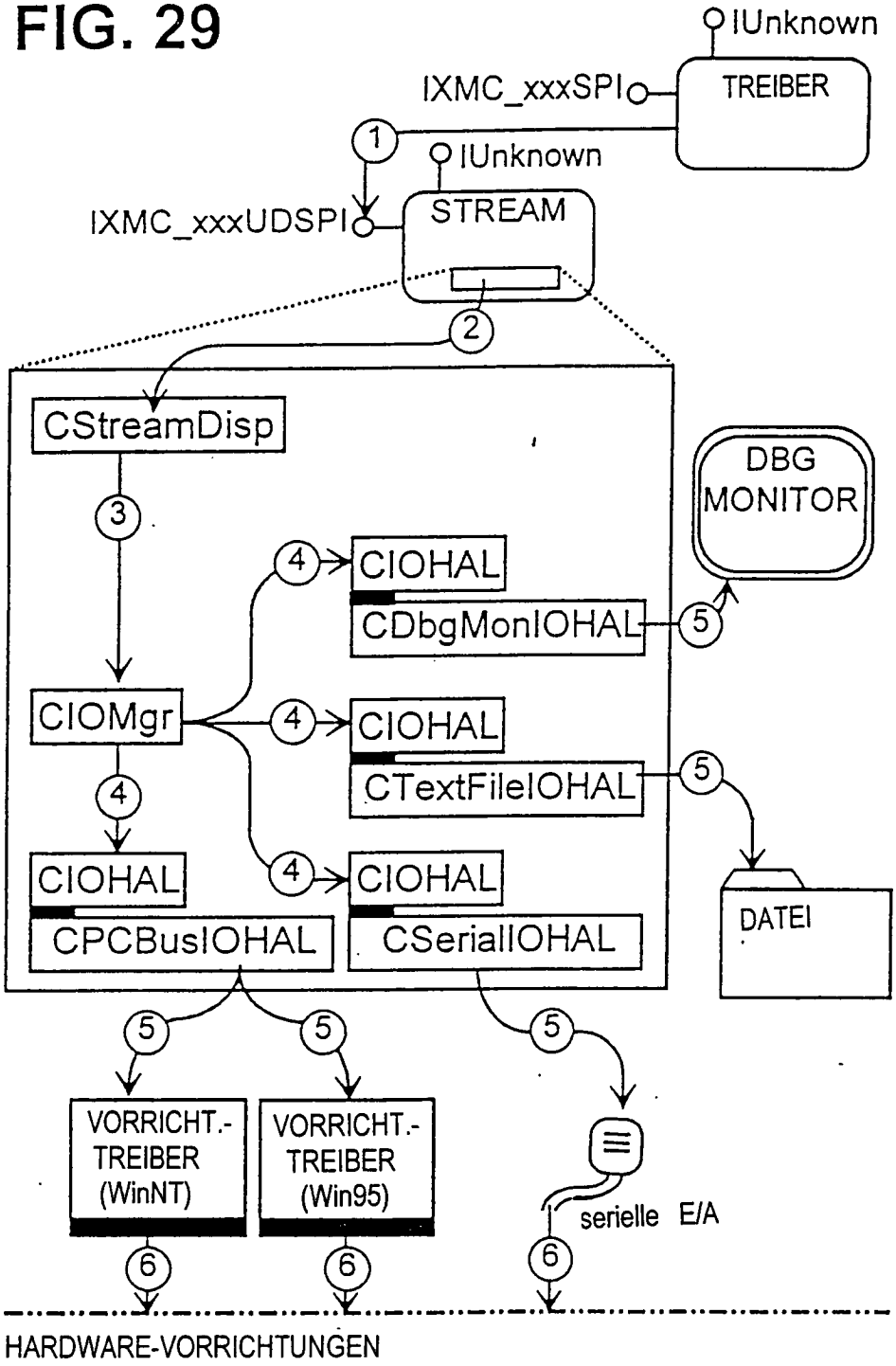

IUnknown
IXMC_xxxSPI
TREIBER
1
IUnknown
IXMC_xxxUDSPI
STREAM
2
CStreamDisp
DBG MONITOR
3
4
CIOHAL
CDbgMonIOHAL
5
CIOMgr
4
CIOHAL
CTextFileIOHAL
5
4
CIOHAL
CIOHAL
CPCBusIOHAL
CSerialIOHAL
DATEI
5
5
5
VORRICHT.-TREIBER (WinNT)
VORRICHT.-TREIBER (Win95)
serielle E/A
6
6
6
HARDWARE-VORRICHTUNGEN

FIG. 30

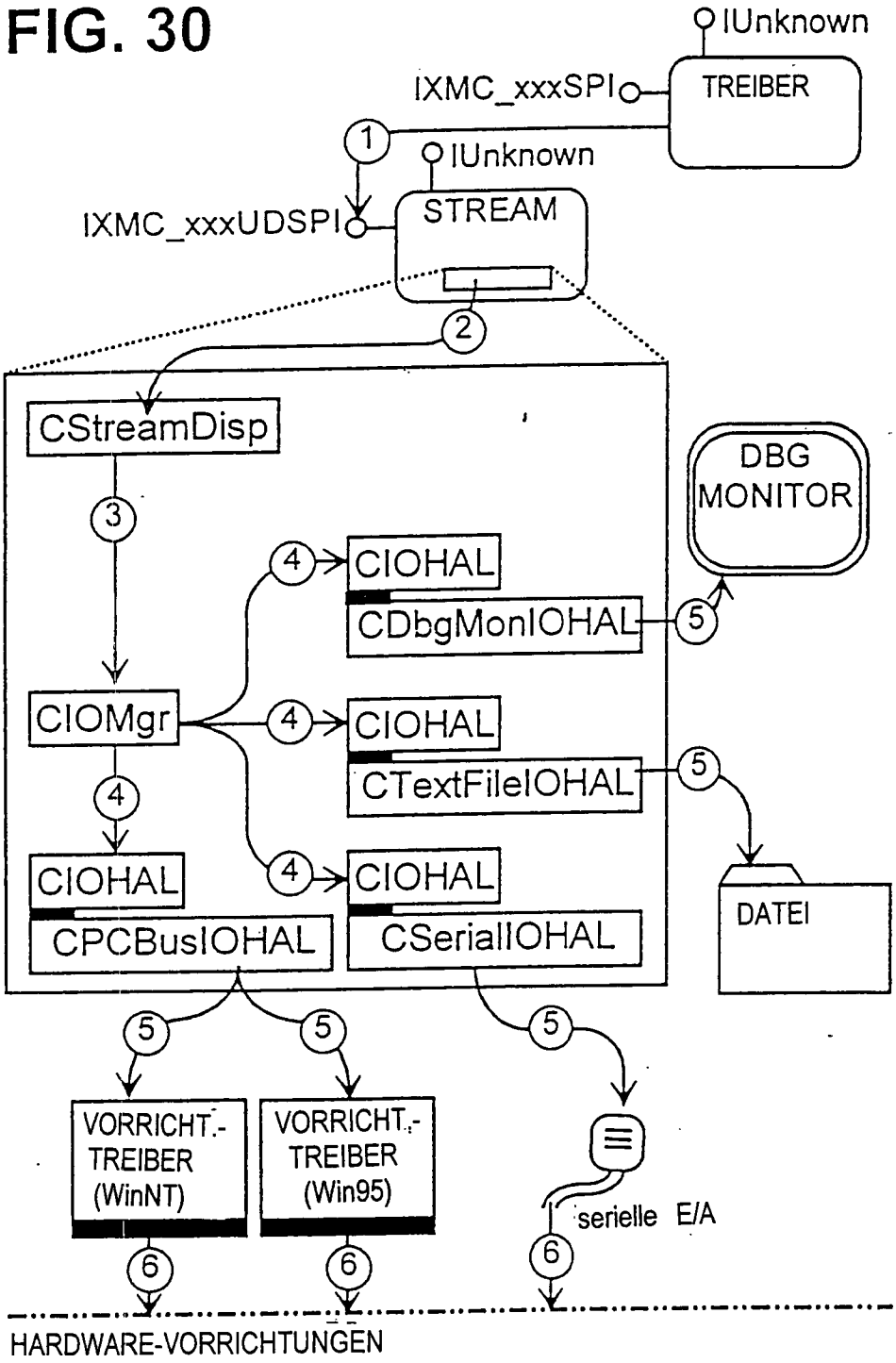

IUnknown
IXMC_xxxSPI
TREIBER
1
IUnknown
IXMC_xxxUDSPI
STREAM
2
CStreamDisp
DBG MONITOR
3
4
CIOHAL
CDbgMonIOHAL
5
CIOMgr
4
CIOHAL
CTextFileIOHAL
5
4
CIOHAL
CIOHAL
CPCBusIOHAL
CSerialIOHAL
DATEI
5
5
5
VORRICHT.-TREIBER (WinNT)
VORRICHT.-TREIBER (Win95)
serielle E/A
6
6
6
HARDWARE-VORRICHTUNGEN

FIG. 31

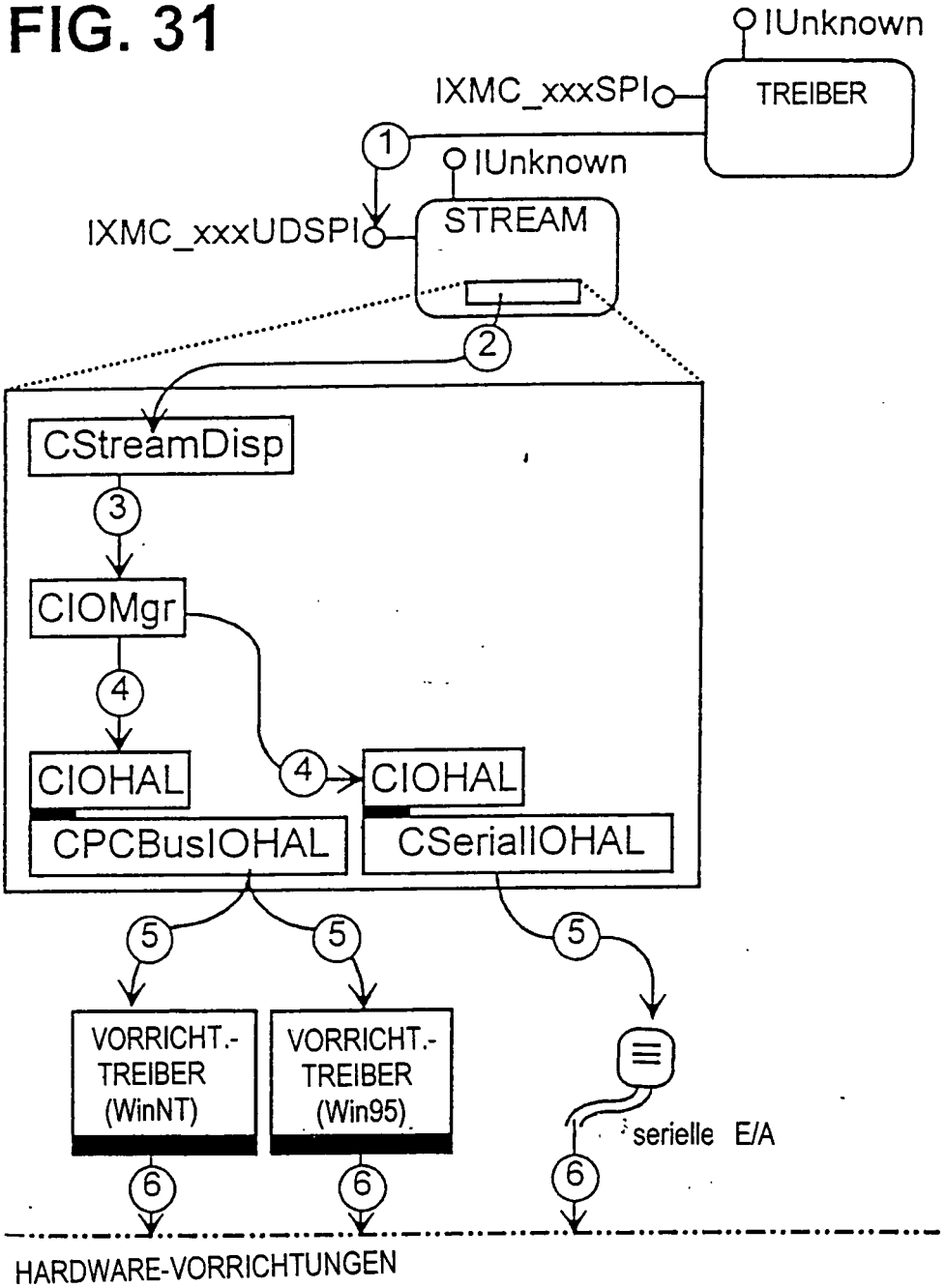

IUnknown
IXMC_xxxSPI
TREIBER
1
IUnknown
IXMC_xxxUDSPI
STREAM
2
CStreamDisp
3
CIOMgr
4
CIOHAL
4
CIOHAL
CPCBusIOHAL
CSerialIOHAL
5
5
5
VORRICHT.-
TREIBER
(WinNT)
VORRICHT.-
TREIBER
(Win95)
serielle E/A
6
6
6
HARDWARE-VORRICHTUNGEN

FIG. 32

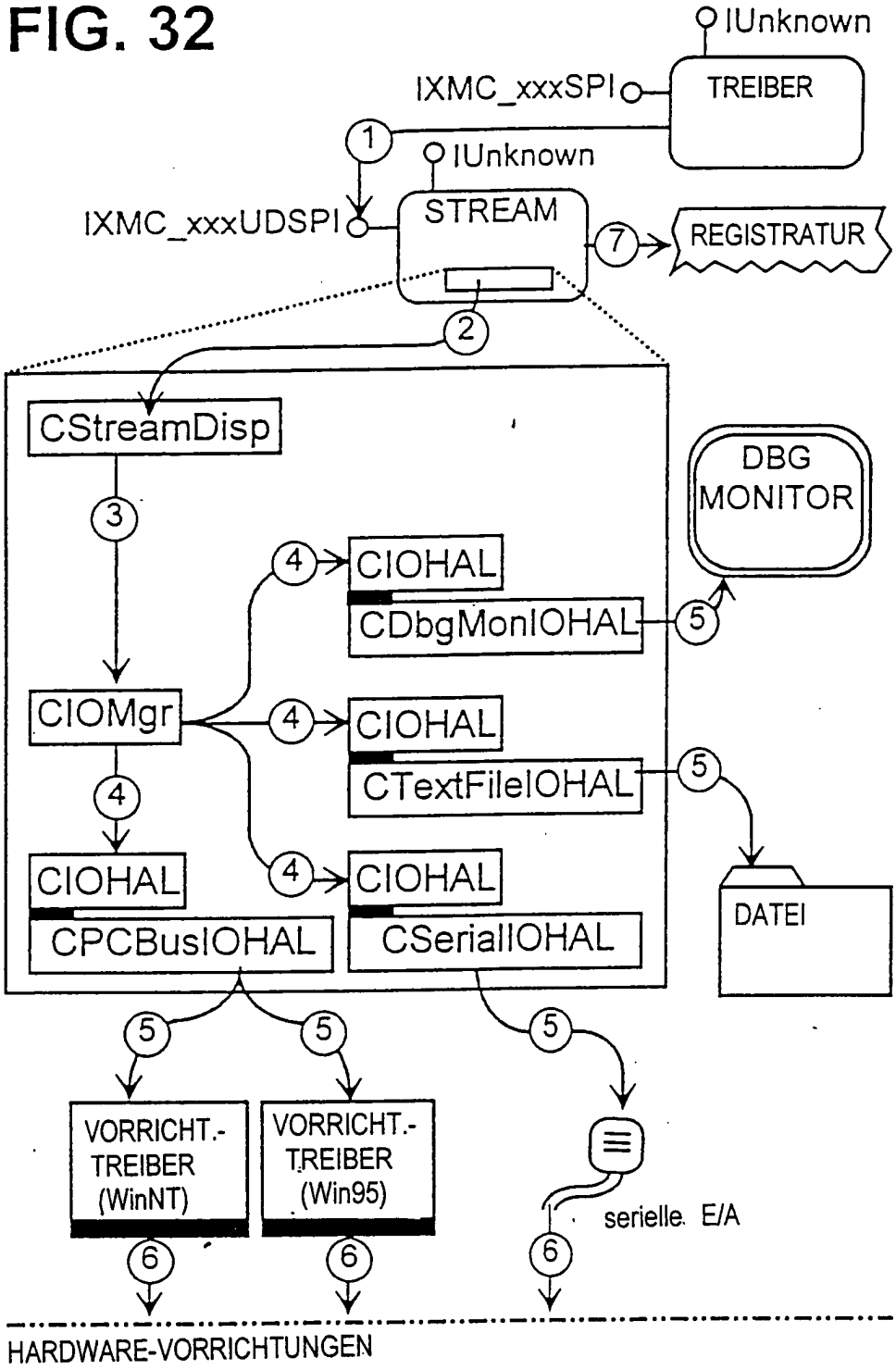

IUnknown
IXMC_xxxSPI
TREIBER
1
IUnknown
IXMC_xxxUDSPI
STREAM
7
REGISTRATUR
2
CStreamDisp
DBG MONITOR
3
4
CIOHAL
CDbgMonIOHAL
5
CIOMgr
4
CIOHAL
CTextFileIOHAL
5
4
CIOHAL
CIOHAL
CPCBusIOHAL
CSerialIOHAL
DATEI
5
5
5
VORRICHT.-TREIBER (WinNT)
VORRICHT.-TREIBER (Win95)
serielle E/A
6
6
6
HARDWARE-VORRICHTUNGEN

FIG. 33

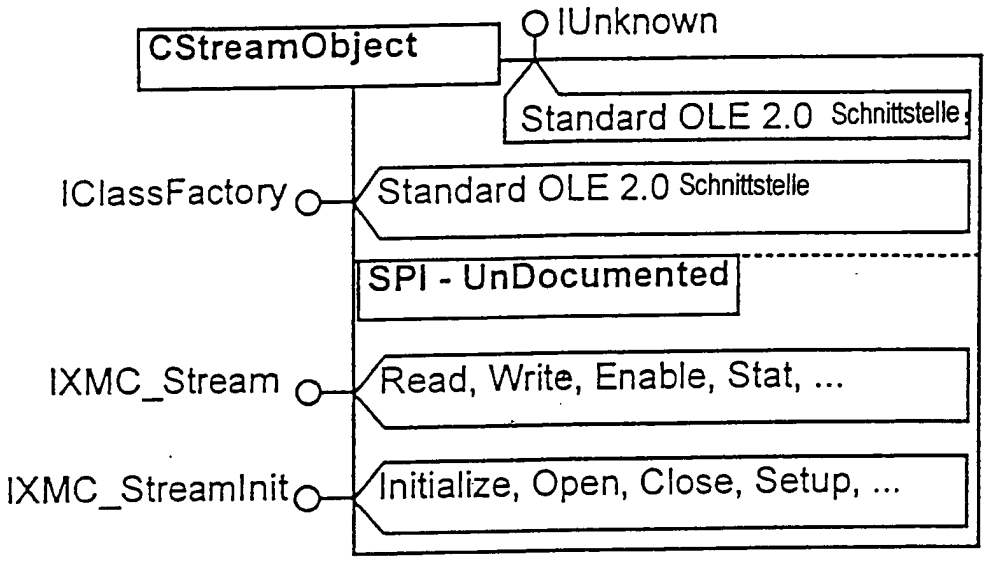

CStreamObject
IUnknown
Standard OLE 2.0 Schnittstelle
IClassFactory
Standard OLE 2.0 Schnittstelle
SPI - UnDocumented
IXMC_Stream
Read, Write, Enable, Stat, ...
IXMC_StreamInit
Initialize, Open, Close, Setup, ...

FIG. 34

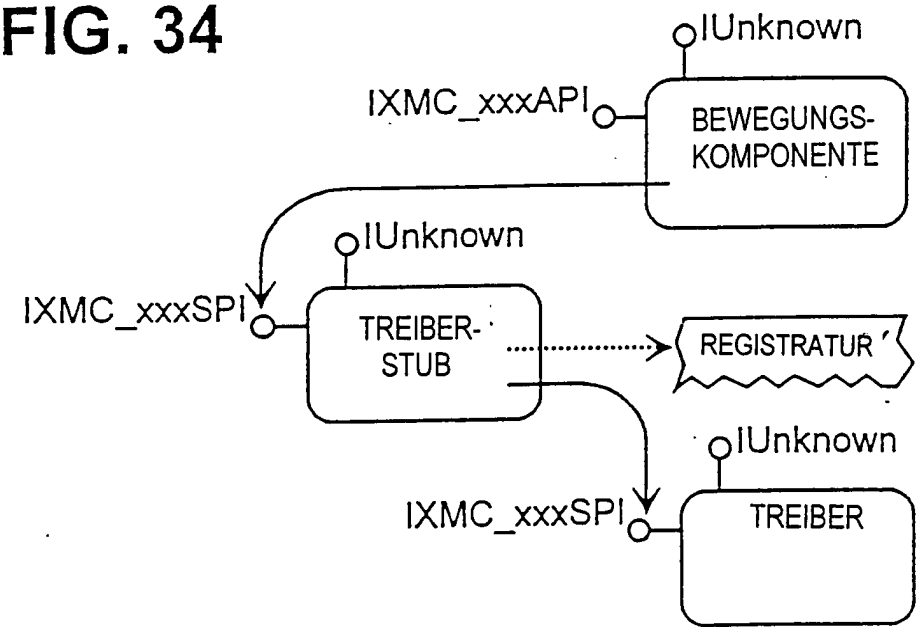

IUnknown
IXMC_xxxAPI
BEWEGUNGS-
KOMPONENTE
IUnknown
IXMC_xxxSPI
TREIBER-
STUB
REGISTRATUR
IUnknown
IXMC_xxxSPI
TREIBER

FIG. 35

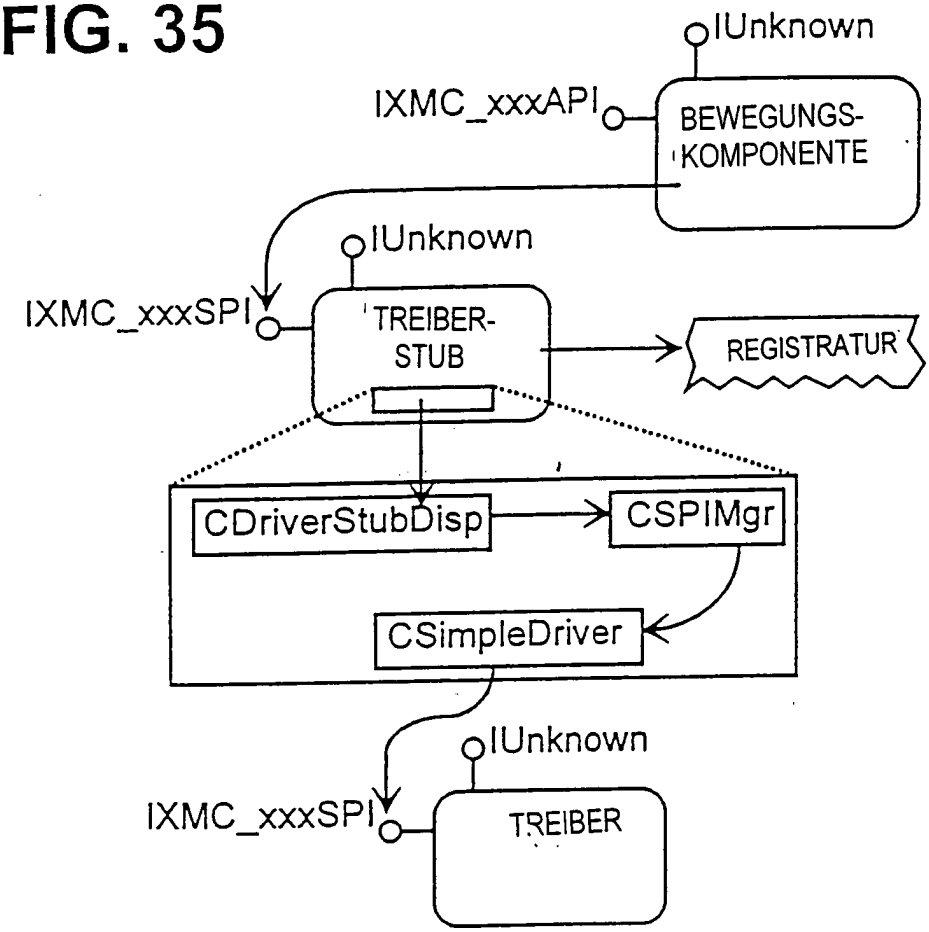

IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
IUnknown
IXMC_xxxSPI
TREIBER-STUB
REGISTRATUR
CDriverStubDisp
CSPIMgr
CSimpleDriver
IUnknown
IXMC_xxxSPI
TREIBER

FIG. 36

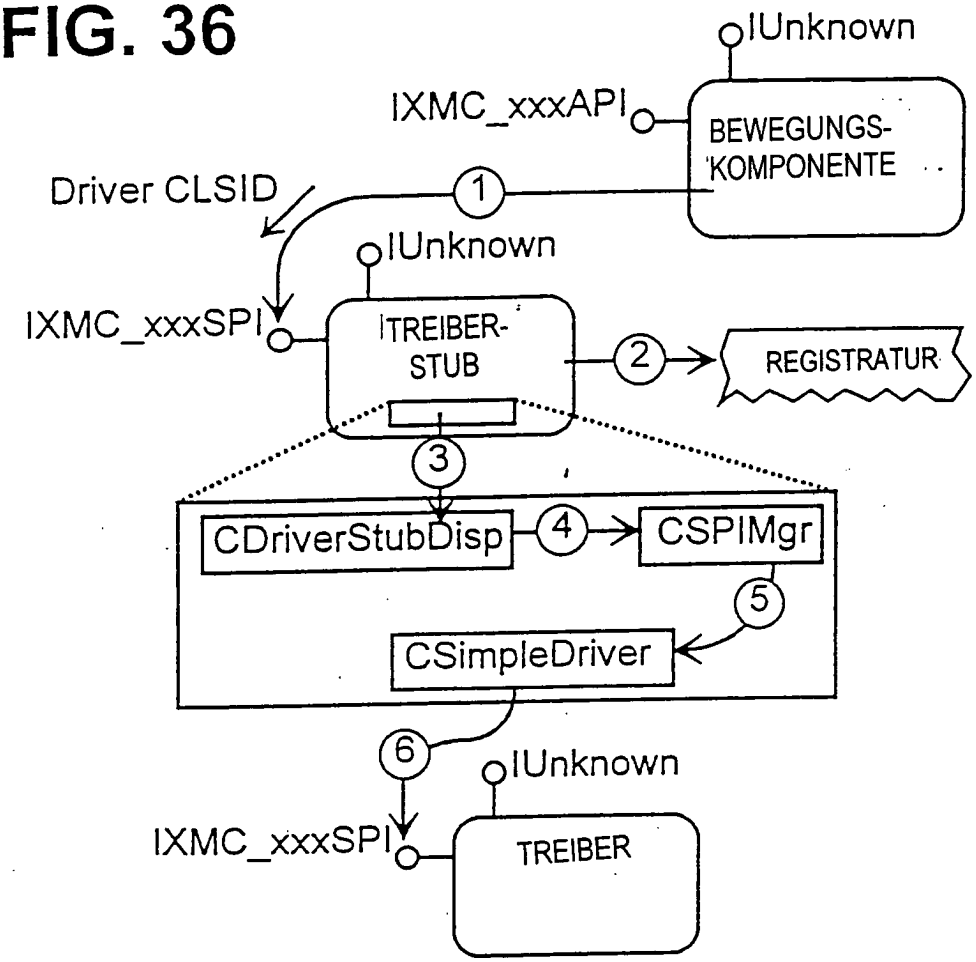

IUnknown
IXMC_xxxAPI
BEWEGUNGS-
KOMPONENTE
Driver CLSID
1
IUnknown
IXMC_xxxSPI
TREIBER-
STUB
2
REGISTRATUR
3
CDriverStubDisp
4
CSPIMgr
5
CSimpleDriver
6
IUnknown
IXMC_xxxSPI
TREIBER

FIG. 37

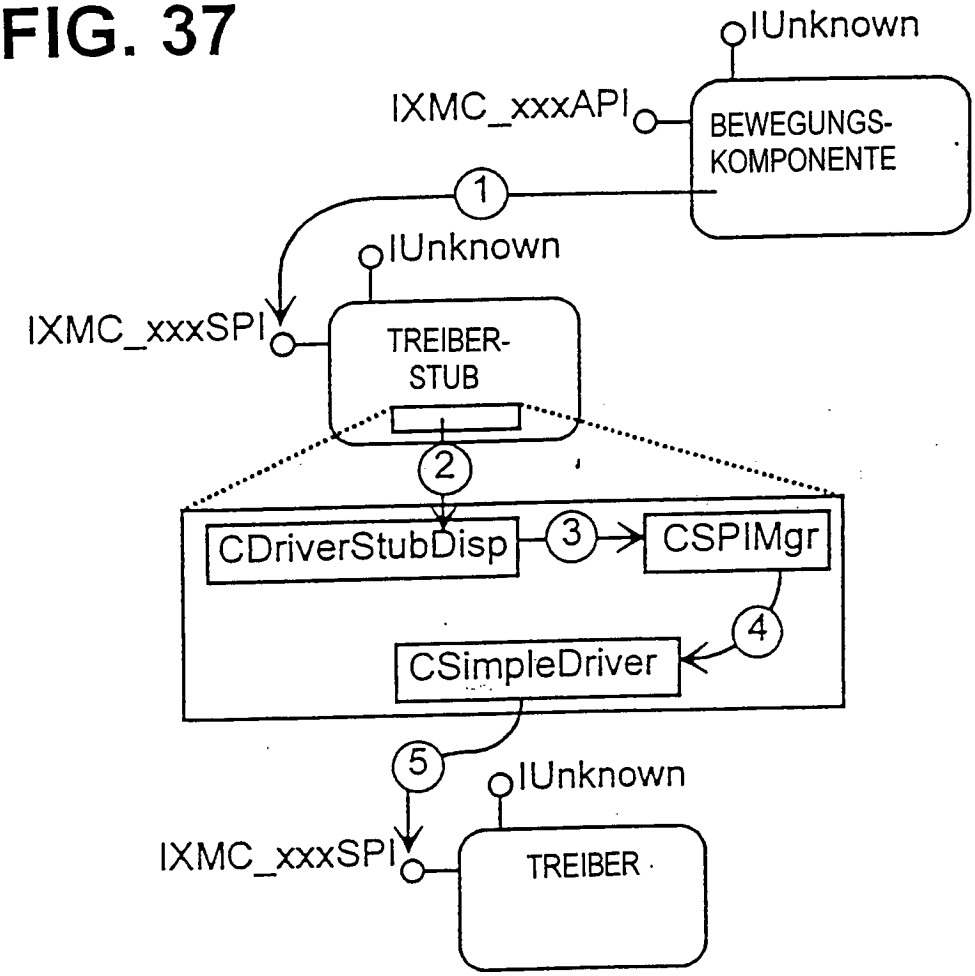

IUnknown
IXMC_xxxAPI
BEWEGUNGS-
KOMPONENTE
1
IUnknown
IXMC_xxxSPI
TREIBER-
STUB
2
CDriverStubDisp
3
CSPIMgr
4
CSimpleDriver
5
IUnknown
IXMC_xxxSPI
TREIBER

FIG. 38

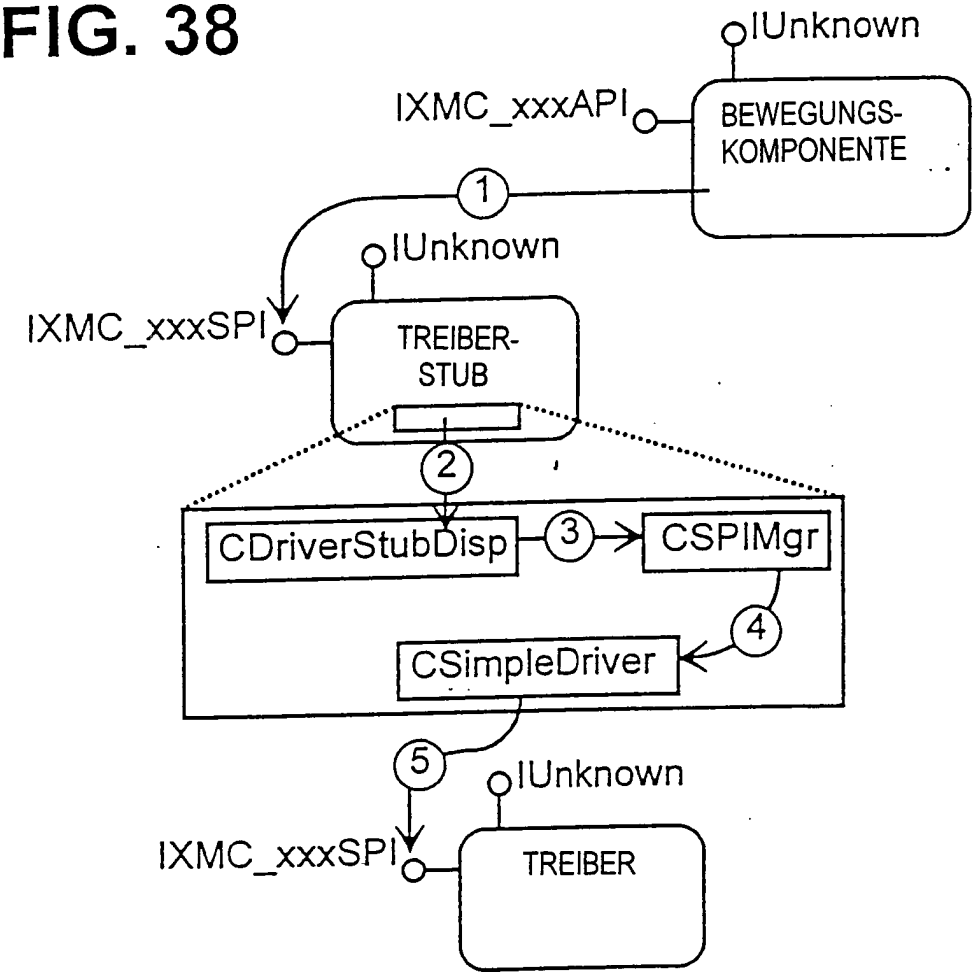

IUnknown
IXMC_xxxAPI
BEWEGUNGS-
KOMPONENTE
1
IUnknown
IXMC_xxxSPI
TREIBER-
STUB
2
CDriverStubDisp
3
CSPIMgr
4
CSimpleDriver
5
IUnknown
IXMC_xxxSPI
TREIBER

FIG. 39

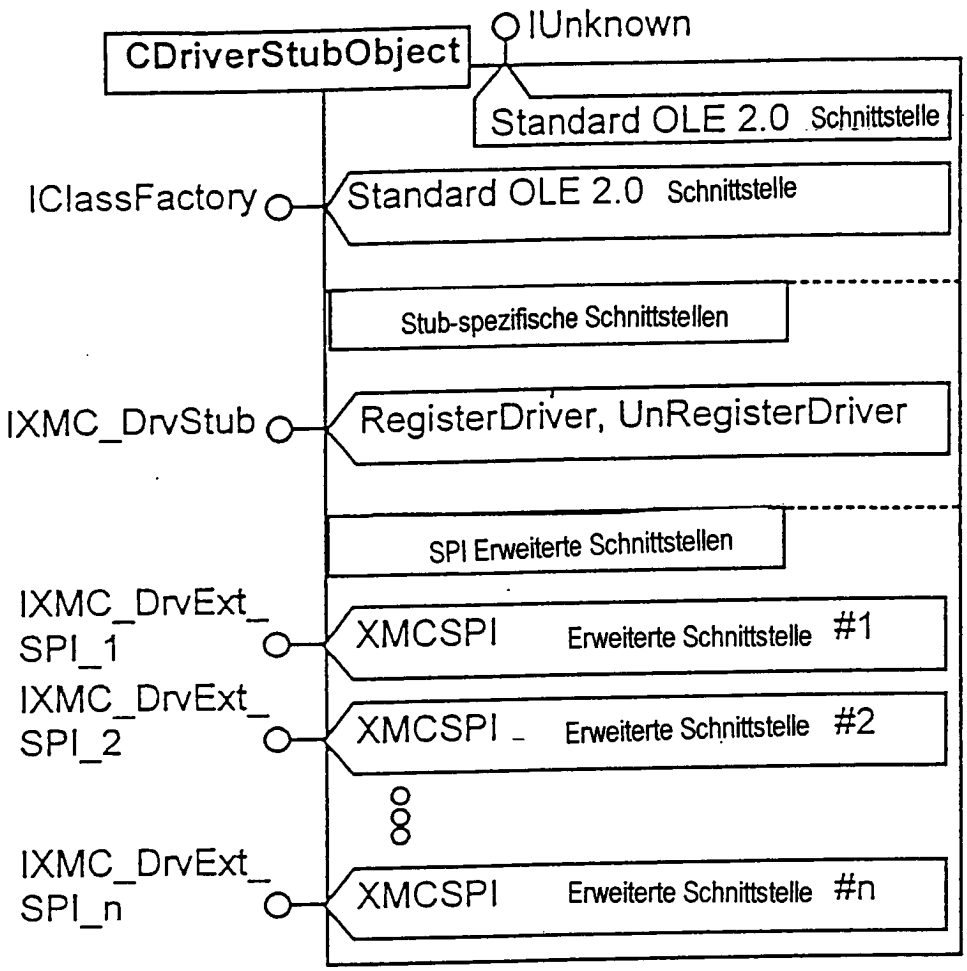

CDriverStubObject
IUnknown
Standard OLE 2.0 Schnittstelle
IClassFactory
Standard OLE 2.0 Schnittstelle
Stub-spezifische Schnittstellen
IXMC_DrvStub
RegisterDriver, UnRegisterDriver
SPI Erweiterte Schnittstellen
IXMC_DrvExt_SPI_1
XMCSPI Erweiterte Schnittstelle #1
IXMC_DrvExt_SPI_2
XMCSPI Erweiterte Schnittstelle #2
IXMC_DrvExt_SPI_n
XMCSPI Erweiterte Schnittstelle #n

FIG. 40

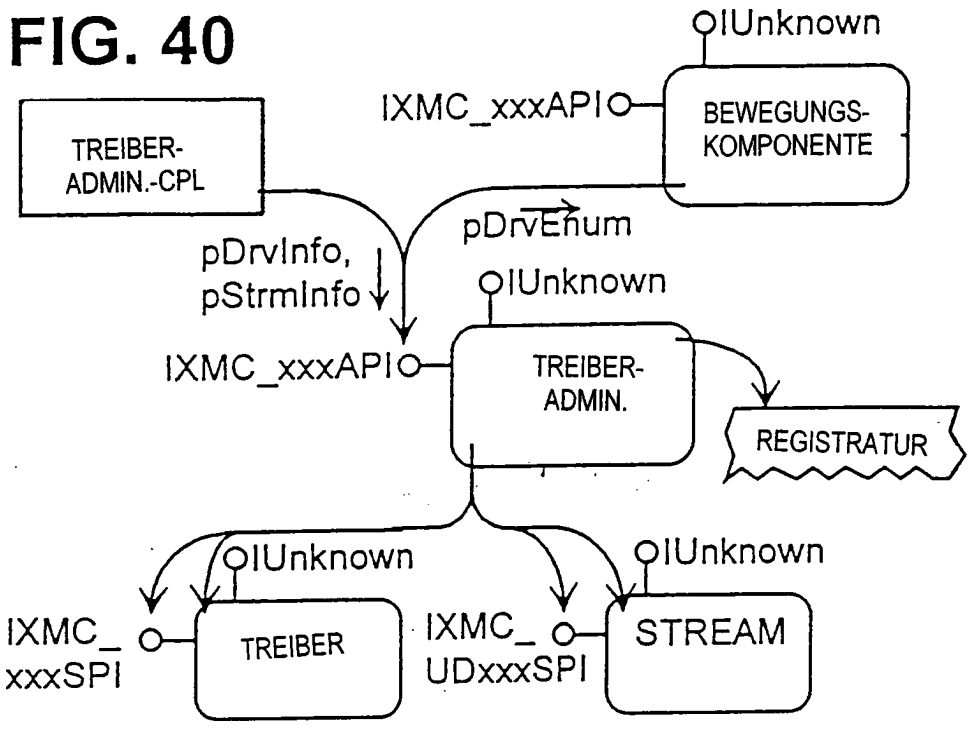

IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
TREIBER-ADMIN.-CPL
pDrvEnum
pDrvInfo,
pStrmInfo
IUnknown
IXMC_xxxAPI
TREIBER-ADMIN.
REGISTRATUR
IUnknown
IUnknown
IXMC_ xxxSPI
TREIBER
IXMC_ UDxxxSPI
STREAM

FIG. 41

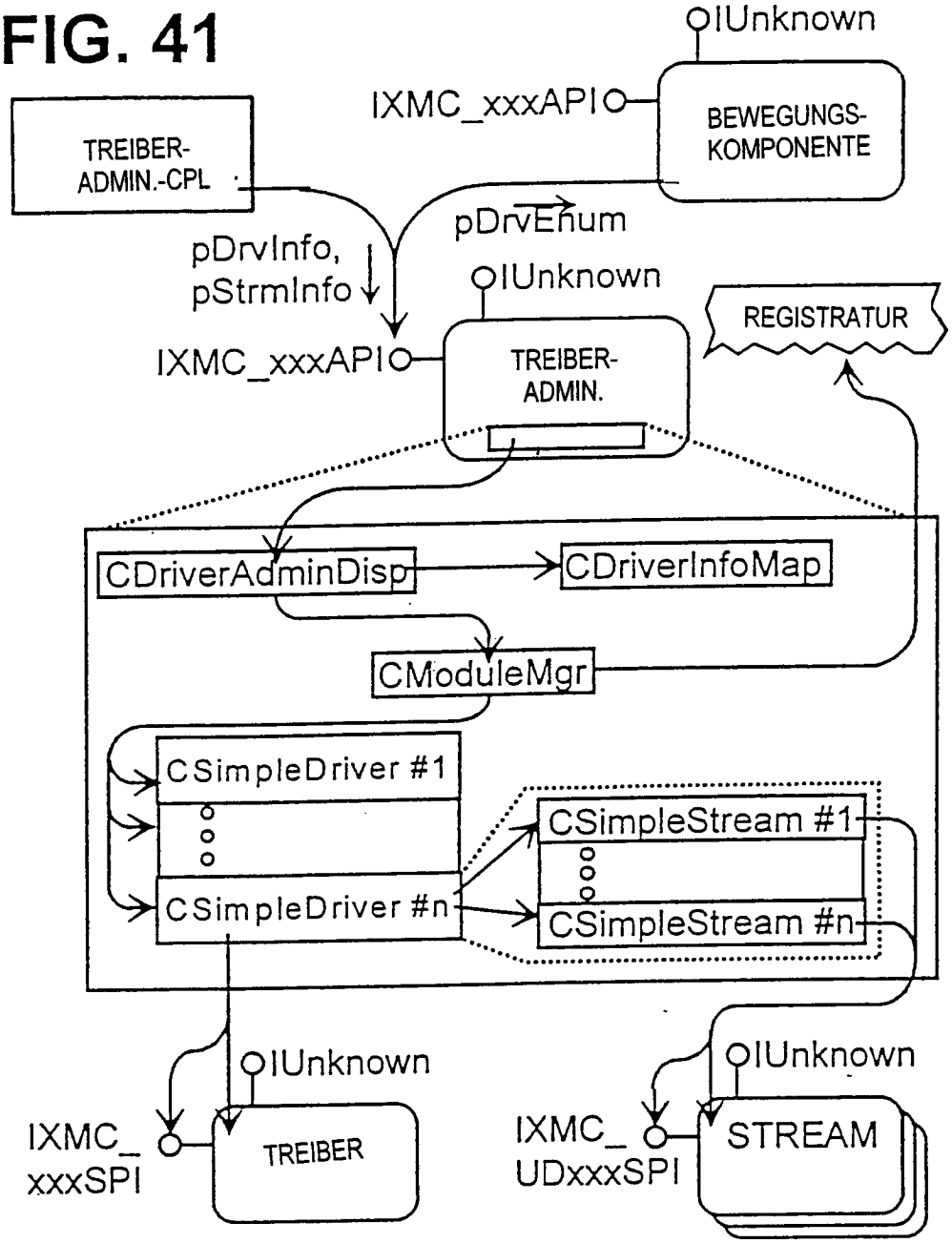

IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
TREIBER-ADMIN.-CPL
pDrvEnum
pDrvInfo,
pStrmInfo
IUnknown
REGISTRATUR
IXMC_xxxAPI
TREIBER-ADMIN.
CDriverAdminDisp
CDriverInfoMap
CModuleMgr
CSimpleDriver #1
CSimpleDriver #n
CSimpleStream #1
CSimpleStream #n
IUnknown
IXMC_ xxxSPI
TREIBER
IUnknown
IXMC_ UDxxxSPI
STREAM

FIG. 42

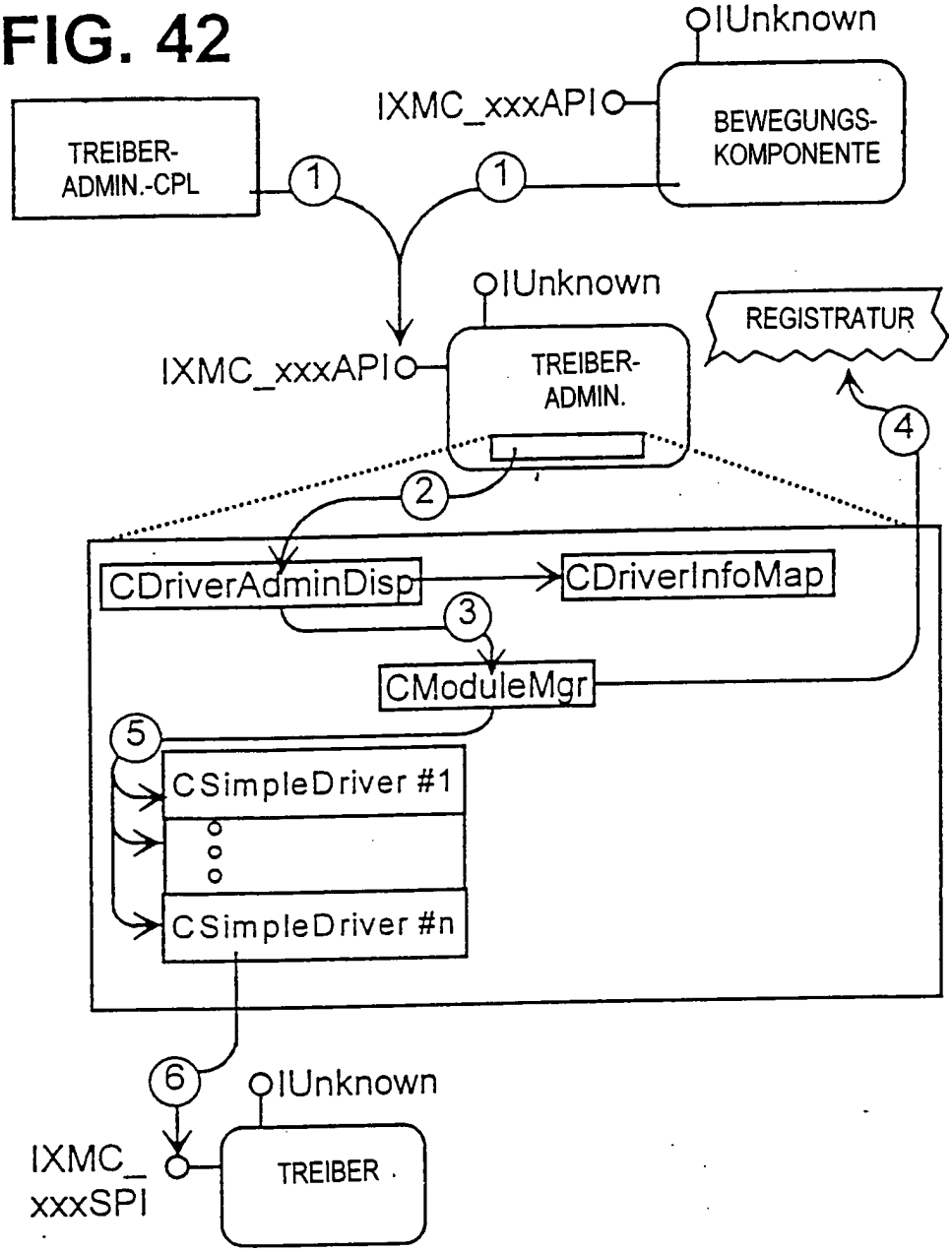

IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
TREIBER-ADMIN.-CPL
1
1
IUnknown
REGISTRATUR
IXMC_xxxAPI
TREIBER-ADMIN.
4
2
CDriverAdminDisp
CDriverInfoMap
3
CModuleMgr
5
CSimpleDriver #1
CSimpleDriver #n
6
IUnknown
IXMC_xxxSPI
TREIBER

FIG. 43

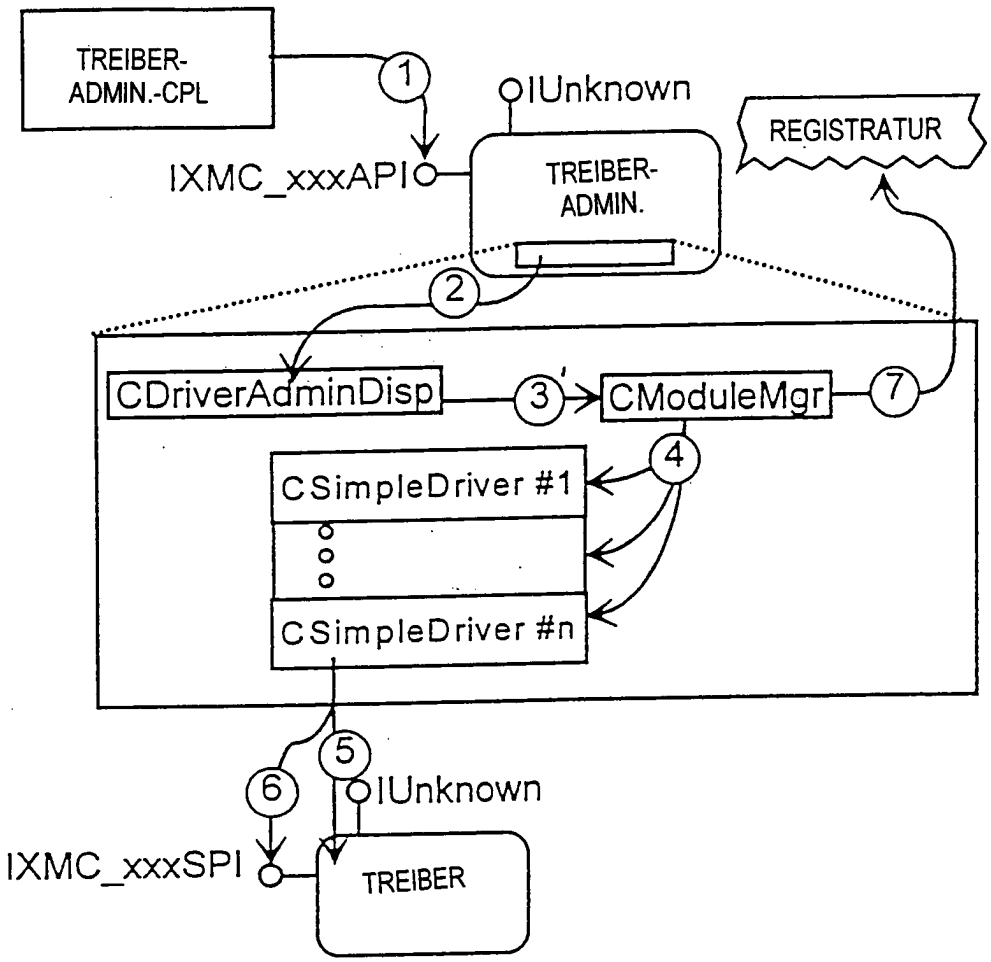

TREIBER-
ADMIN.-CPL
1
IUnknown
REGISTRATUR
IXMC_xxxAPI
TREIBER-
ADMIN.
2
CDriverAdminDisp
3'
CModuleMgr
7
4
CSimpleDriver #1
CSimpleDriver #n
5
6
IUnknown
IXMC_xxxSPI
TREIBER

FIG. 44

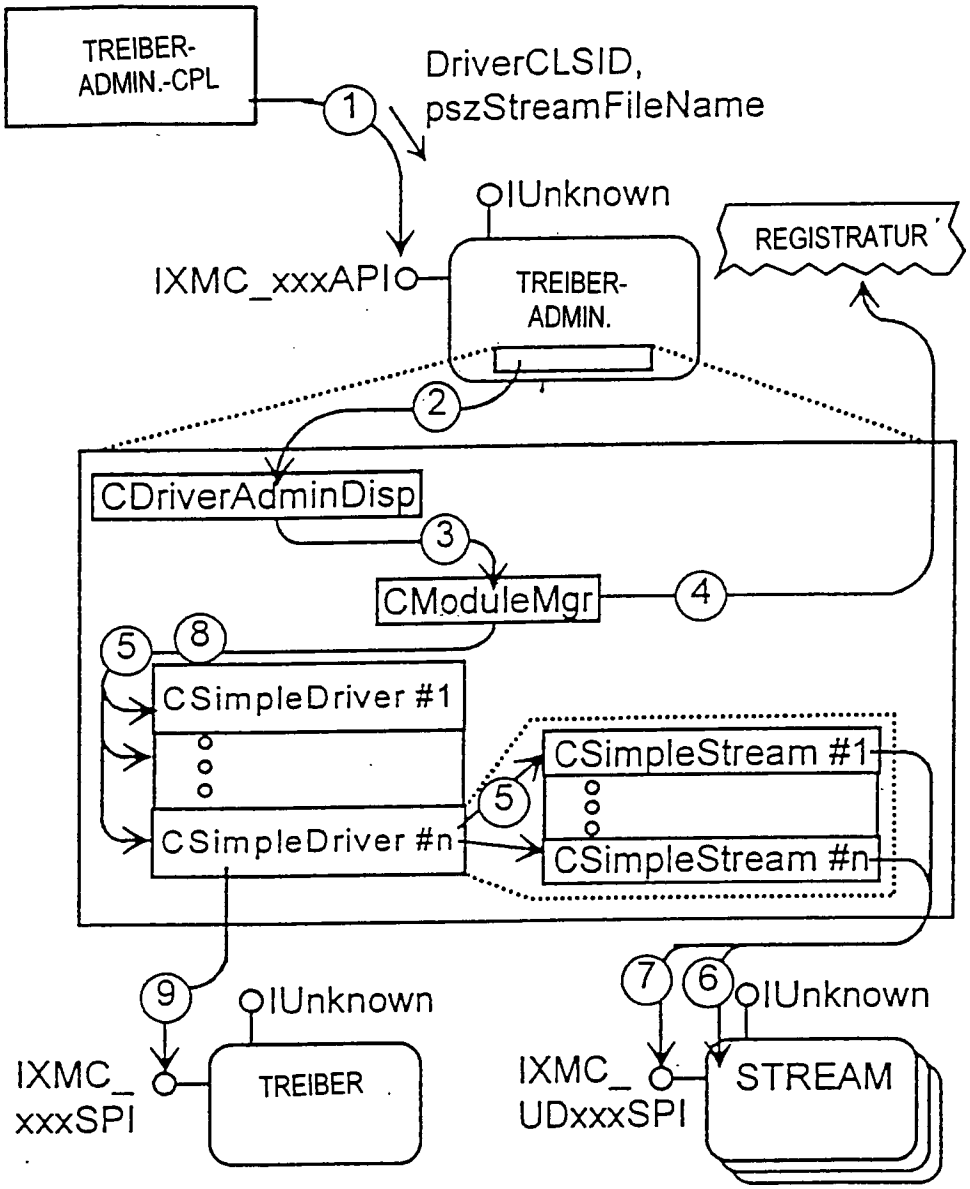

TREIBER-
ADMIN.-CPL
1
DriverCLSID,
pszStreamFileName
IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.
REGISTRATUR
2
CDriverAdminDisp
3
CModuleMgr
4
5
8
CSimpleDriver #1
CSimpleDriver #n
5
CSimpleStream #1
CSimpleStream #n
9
IUnknown
IXMC_
xxxSPI
TREIBER
7
6
IUnknown
IXMC_
UDxxxSPI
STREAM

FIG. 45

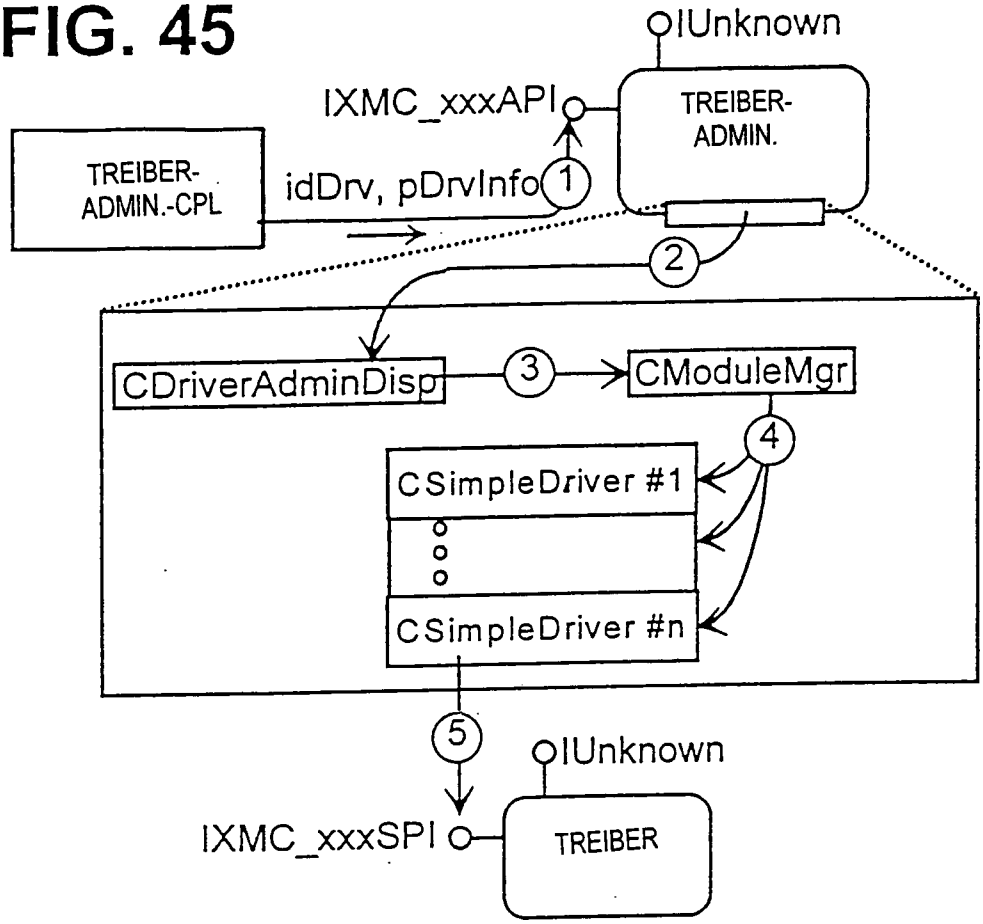

IUnknown
IXMC_xxxAPI
TREIBER-ADMIN.
TREIBER-ADMIN.-CPL
idDrv, pDrvInfo
1
2
CDriverAdminDisp
3
CModuleMgr
4
CSimpleDriver #1
CSimpleDriver #n
5
IUnknown
IXMC_xxxSPI
TREIBER

FIG. 46

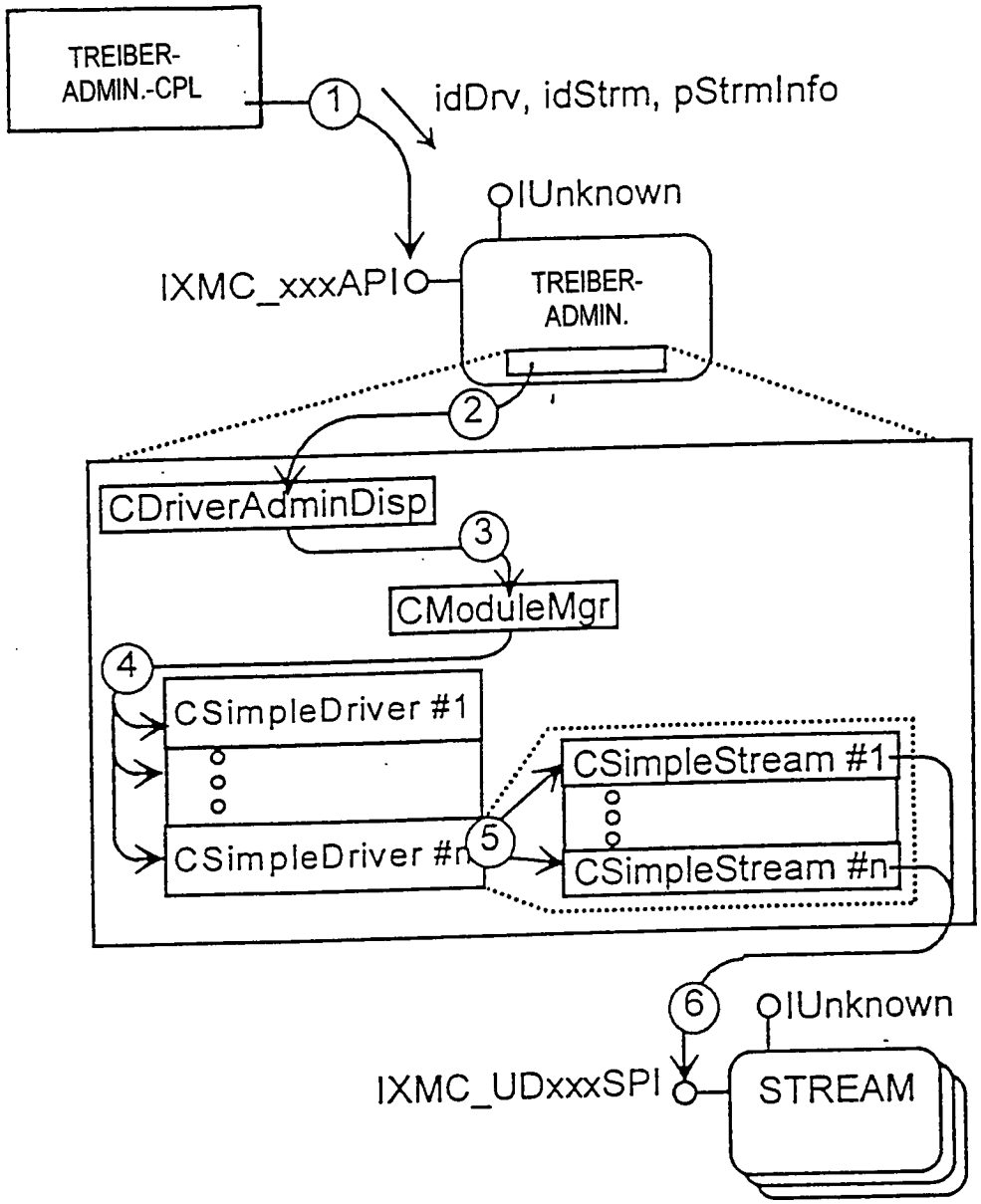

TREIBER-
ADMIN.-CPL
1
idDrv, idStrm, pStrmInfo
IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.
2
CDriverAdminDisp
3
CModuleMgr
4
CSimpleDriver #1
CSimpleDriver #n
5
CSimpleStream #1
CSimpleStream #n
6
IUnknown
IXMC_UDxxxSPI
STREAM

FIG. 47

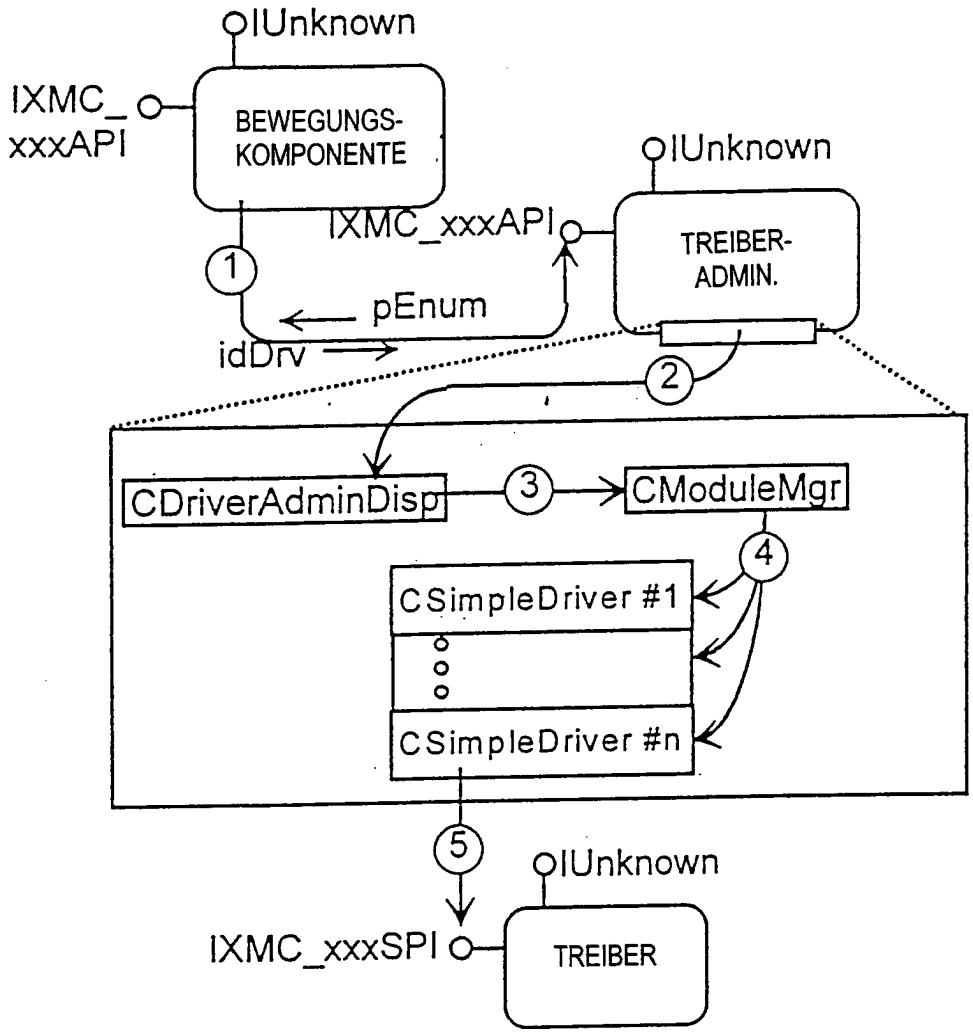

IUnknown
IXMC_
xxxAPI
BEWEGUNGS-
KOMPONENTE
IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.
1
pEnum
idDrv
2
CDriverAdminDisp
3
CModuleMgr
4
CSimpleDriver #1
CSimpleDriver #n
5
IUnknown
IXMC_xxxSPI
TREIBER

FIG. 48

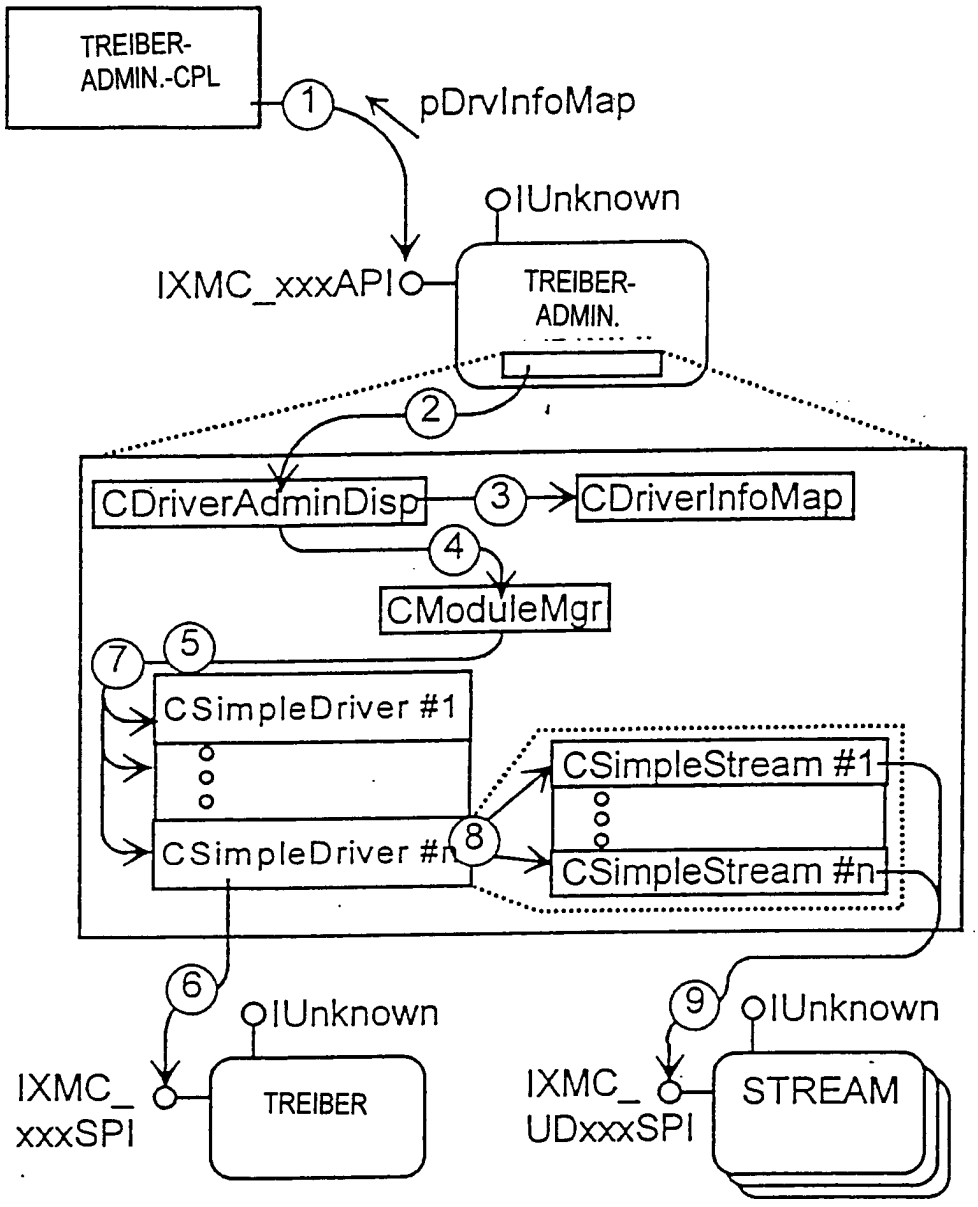

TREIBER-
ADMIN.-CPL
1
pDrvInfoMap
IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.
2
CDriverAdminDisp
3
CDriverInfoMap
4
CModuleMgr
7
5
CSimpleDriver #1
CSimpleDriver #n
8
CSimpleStream #1
CSimpleStream #n
6
IUnknown
IXMC_
xxxSPI
TREIBER
9
IUnknown
IXMC_
UDxxxSPI
STREAM

FIG. 49

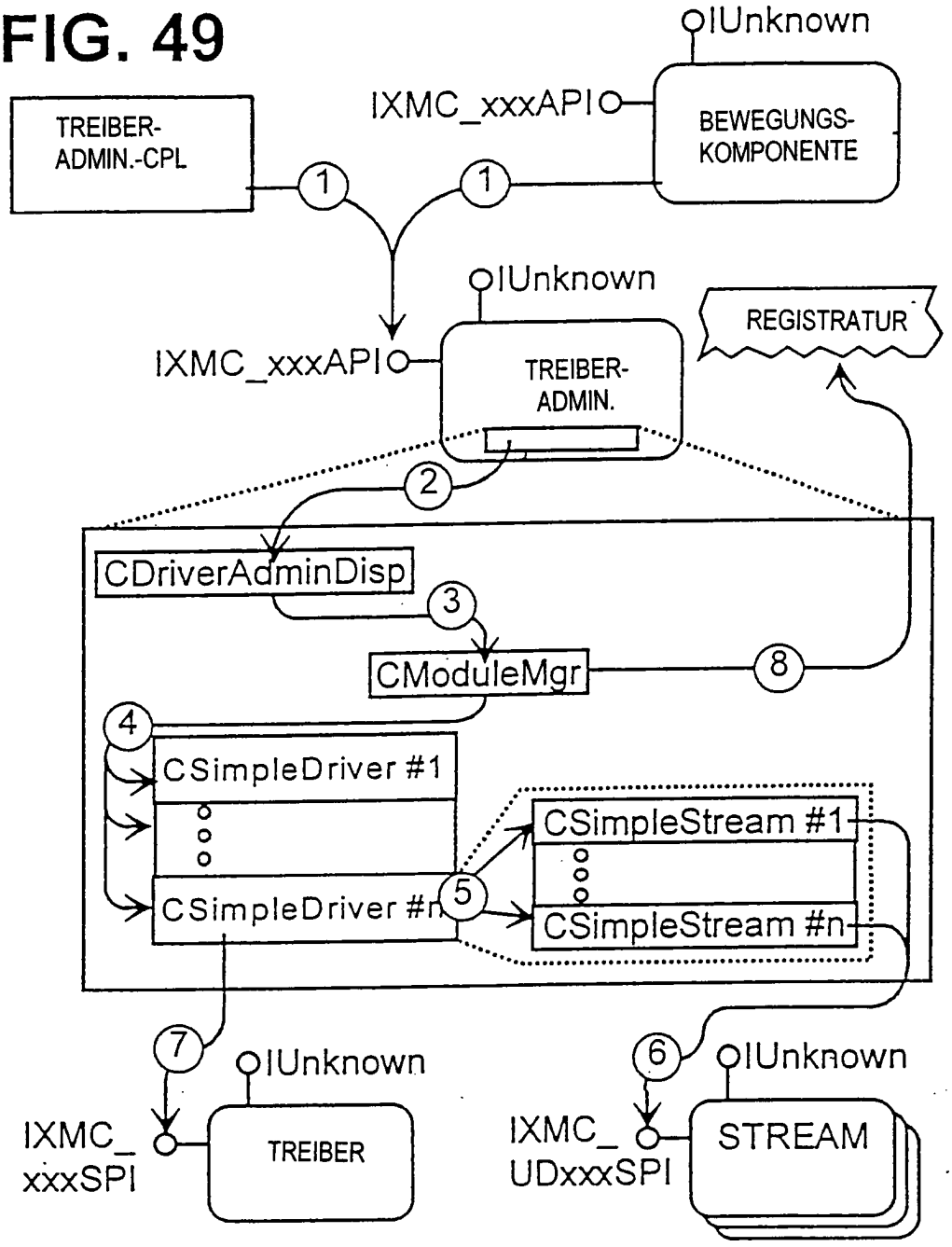

IUnknown
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
TREIBER-ADMIN.-CPL
1
1
IUnknown
IXMC_xxxAPI
TREIBER-ADMIN.
REGISTRATUR
2
CDriverAdminDisp
3
CModuleMgr
8
4
CSimpleDriver #1
CSimpleDriver #n
5
CSimpleStream #1
CSimpleStream #n
7
6
IUnknown
IXMC_xxxSPI
TREIBER
IUnknown
IXMC_UDxxxSPI
STREAM

FIG. 50

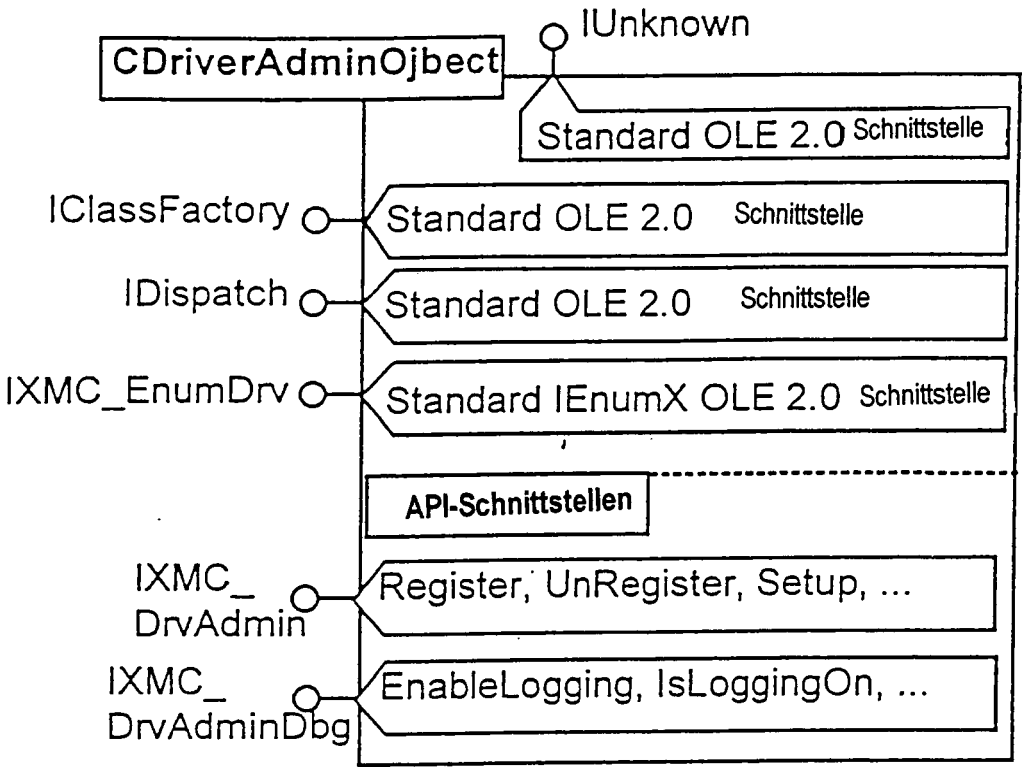

IUnknown
CDriverAdminOjbect
Standard OLE 2.0 Schnittstelle
IClassFactory
Standard OLE 2.0 Schnittstelle
IDispatch
Standard OLE 2.0 Schnittstelle
IXMC_EnumDrv
Standard IEnumX OLE 2.0 Schnittstelle
API-Schnittstellen
IXMC_ DrvAdmin
Register, UnRegister, Setup, ...
IXMC_ DrvAdminDbg
EnableLogging, IsLoggingOn, ...

FIG. 51

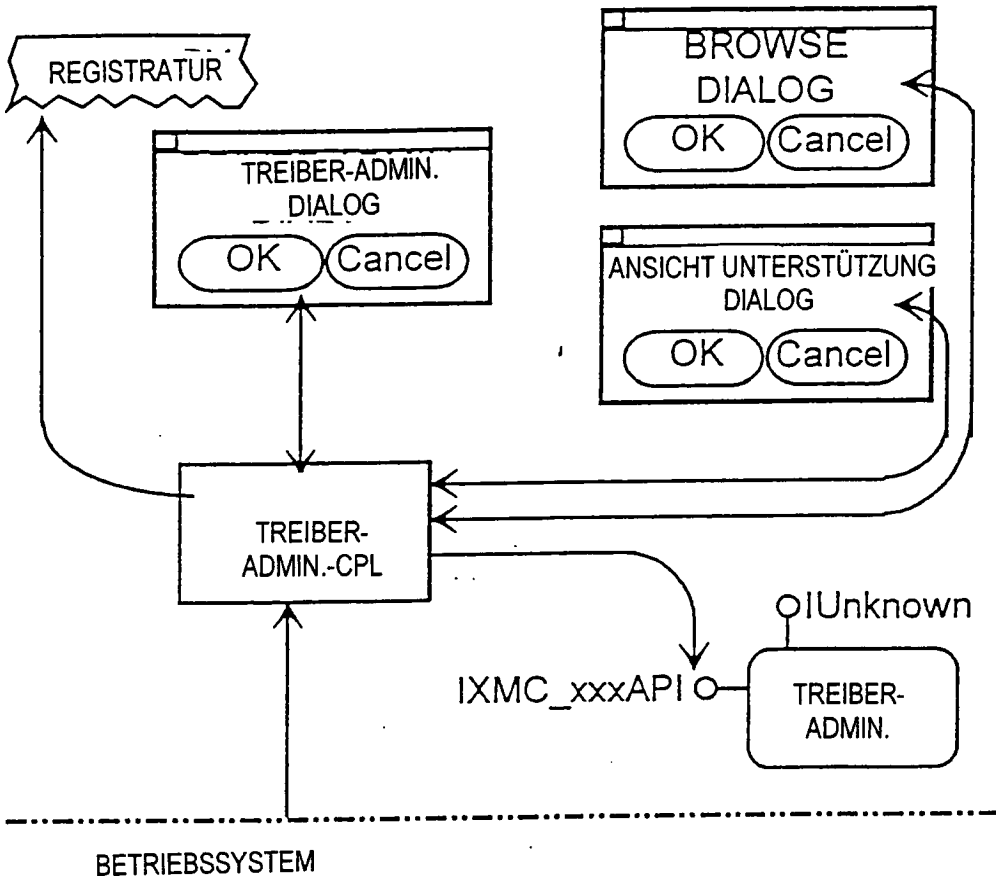

REGISTRATUR
BROWSE DIALOG
OK
Cancel
TREIBER-ADMIN. DIALOG
OK
Cancel
ANSICHT UNTERSTÜTZUNG DIALOG
OK
Cancel
TREIBER-ADMIN.-CPL
IUnknown
IXMC_xxxAPI
TREIBER-ADMIN.
BETRIEBSSYSTEM

FIG. 52

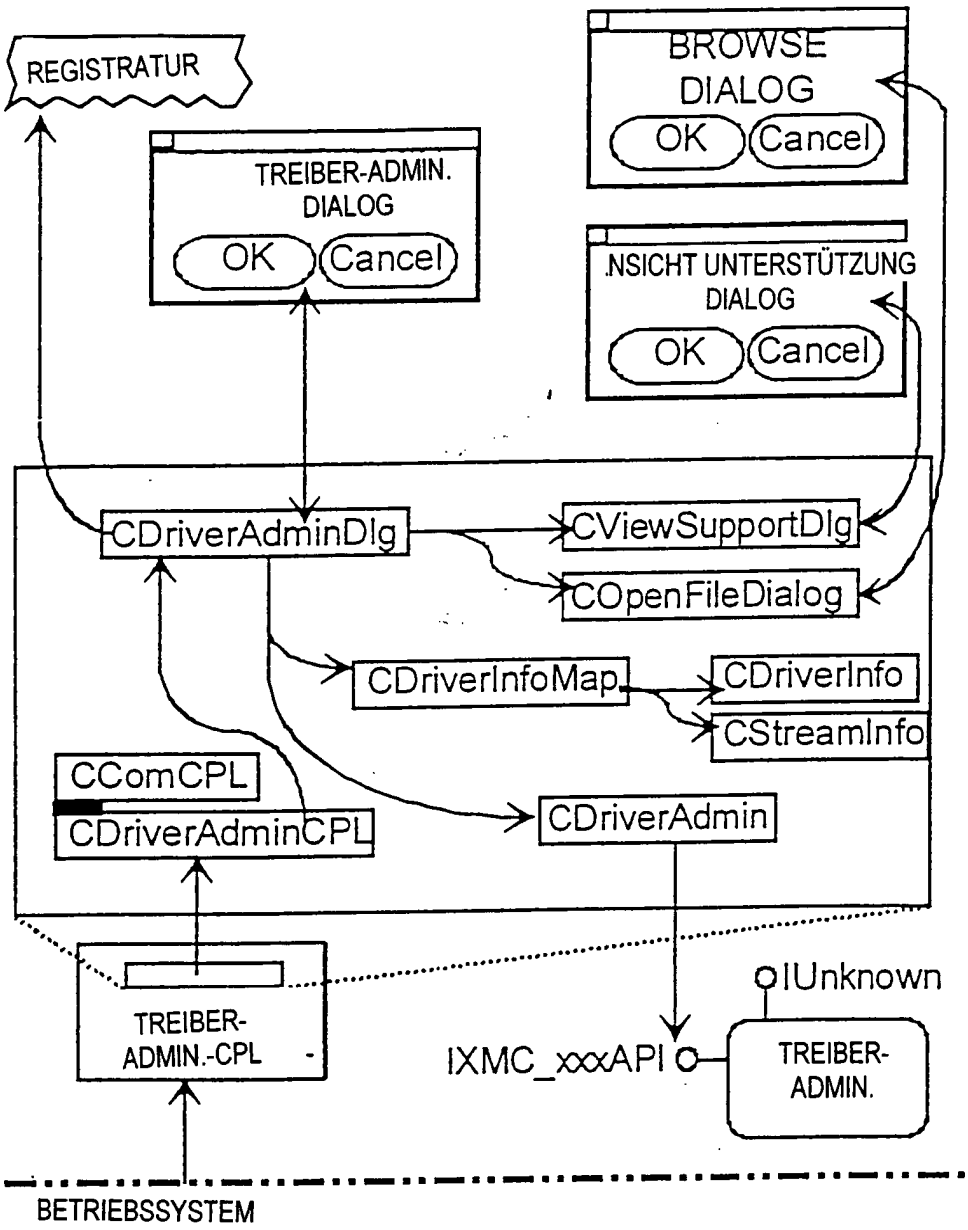

REGISTRATUR
BROWSE DIALOG
OK
Cancel
TREIBER-ADMIN. DIALOG
OK
Cancel
.NSICHT UNTERSTÜTZUNG DIALOG
OK
Cancel
CDriverAdminDlg
CViewSupportDlg
COpenFileDialog
CDriverInfoMap
CDriverInfo
CStreamInfo
CComCPL
CDriverAdminCPL
CDriverAdmin
IUnknown
TREIBER-ADMIN.-CPL
IXMC_xxxAPI
TREIBER-ADMIN.
BETRIEBSSYSTEM

FIG. 53

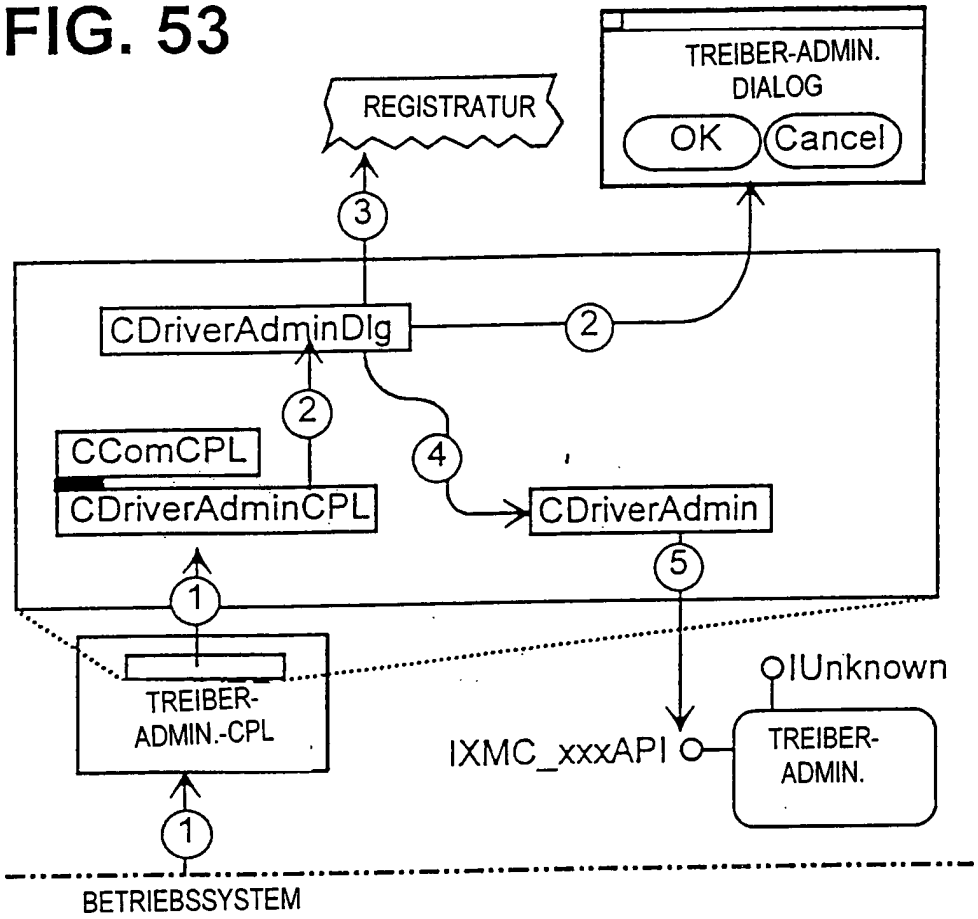

TREIBER-ADMIN.
DIALOG
OK
Cancel
REGISTRATUR
3
CDriverAdminDlg
2
2
CComCPL
CDriverAdminCPL
4
CDriverAdmin
5
1
TREIBER-
ADMIN.-CPL
IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.
1
BETRIEBSSYSTEM

FIG. 54

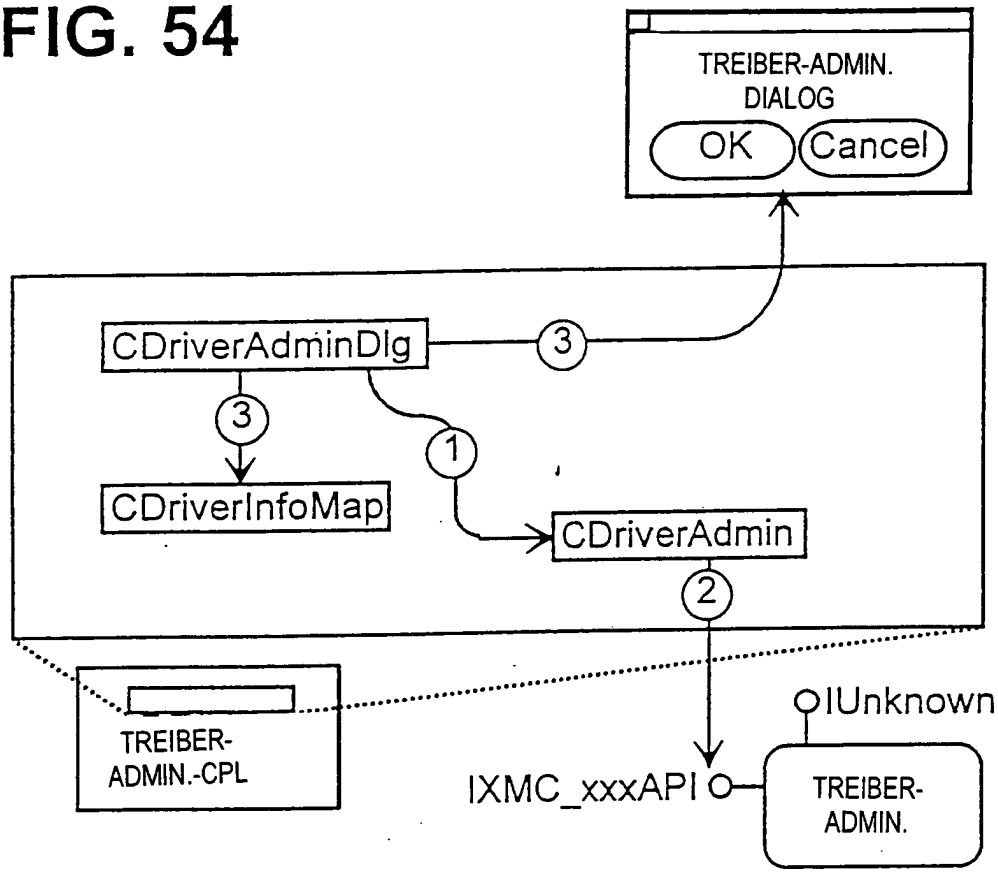

TREIBER-ADMIN.
DIALOG
OK
Cancel
CDriverAdminDlg
3
3
1
CDriverInfoMap
CDriverAdmin
2
IUnknown
TREIBER-
ADMIN.-CPL
IXMC_xxxAPI
TREIBER-
ADMIN.

FIG. 55

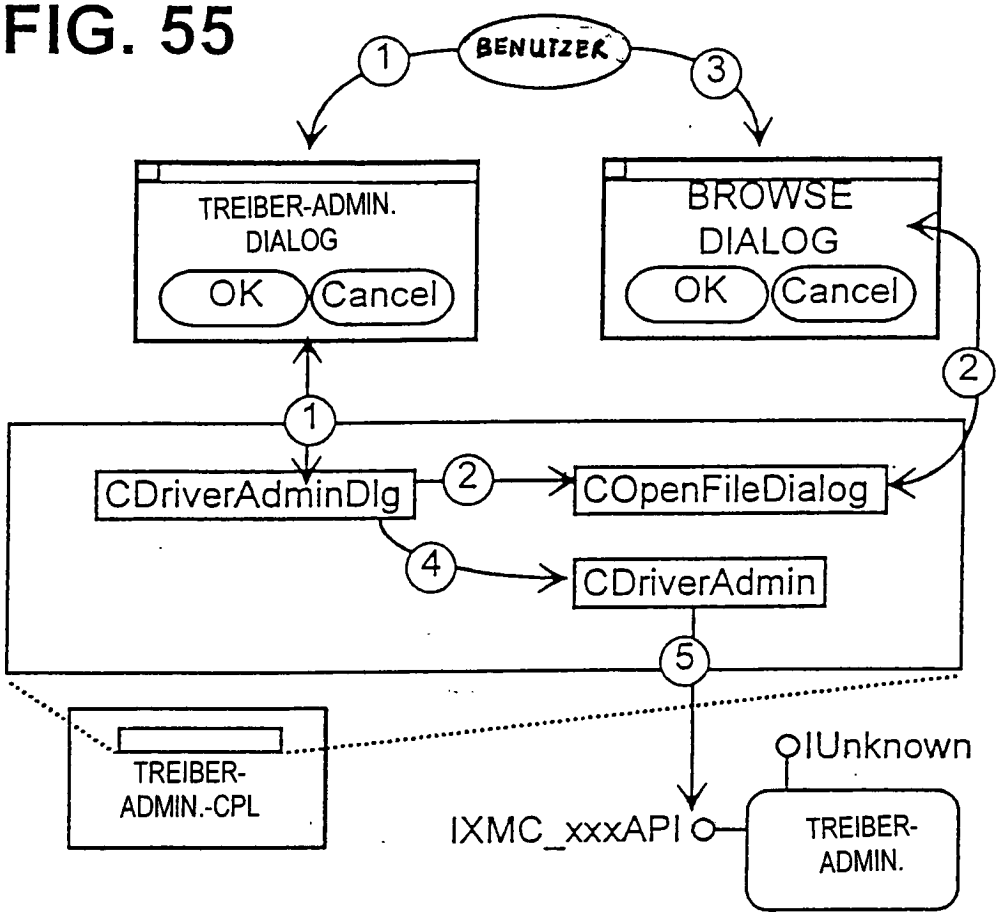

BENUTZER
1
3
TREIBER-ADMIN.
DIALOG
OK
Cancel
BROWSE
DIALOG
OK
Cancel
2
1
CDriverAdminDlg
2
COpenFileDialog
4
CDriverAdmin
5
TREIBER-
ADMIN.-CPL
IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.

FIG. 56

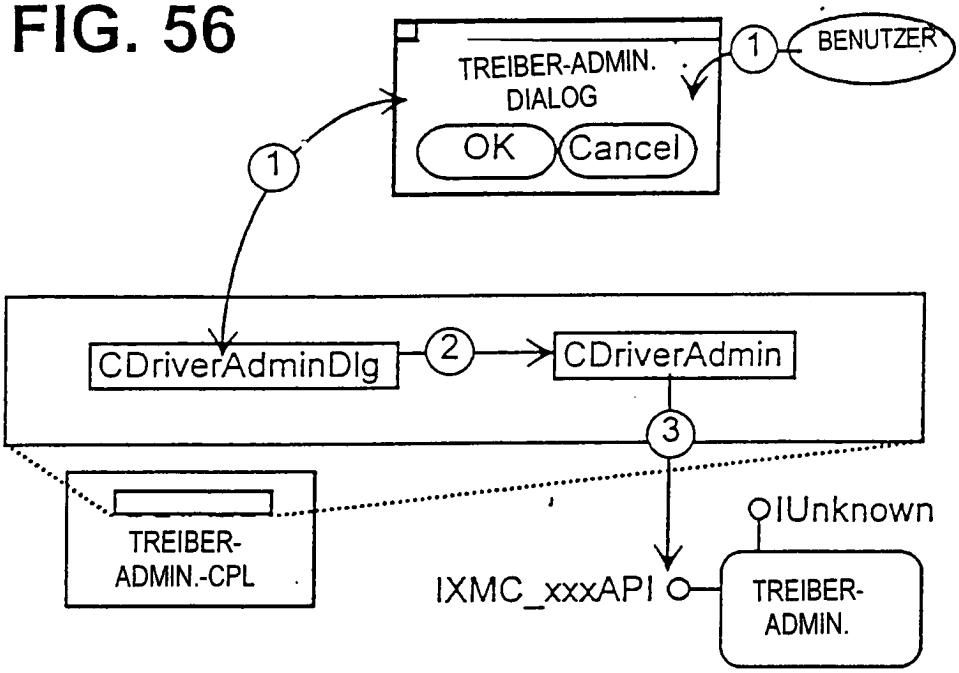

TREIBER-ADMIN.
DIALOG
OK
Cancel
1
BENUTZER
1
CDriverAdminDlg
2
CDriverAdmin
3
TREIBER-
ADMIN.-CPL
IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.

FIG. 57

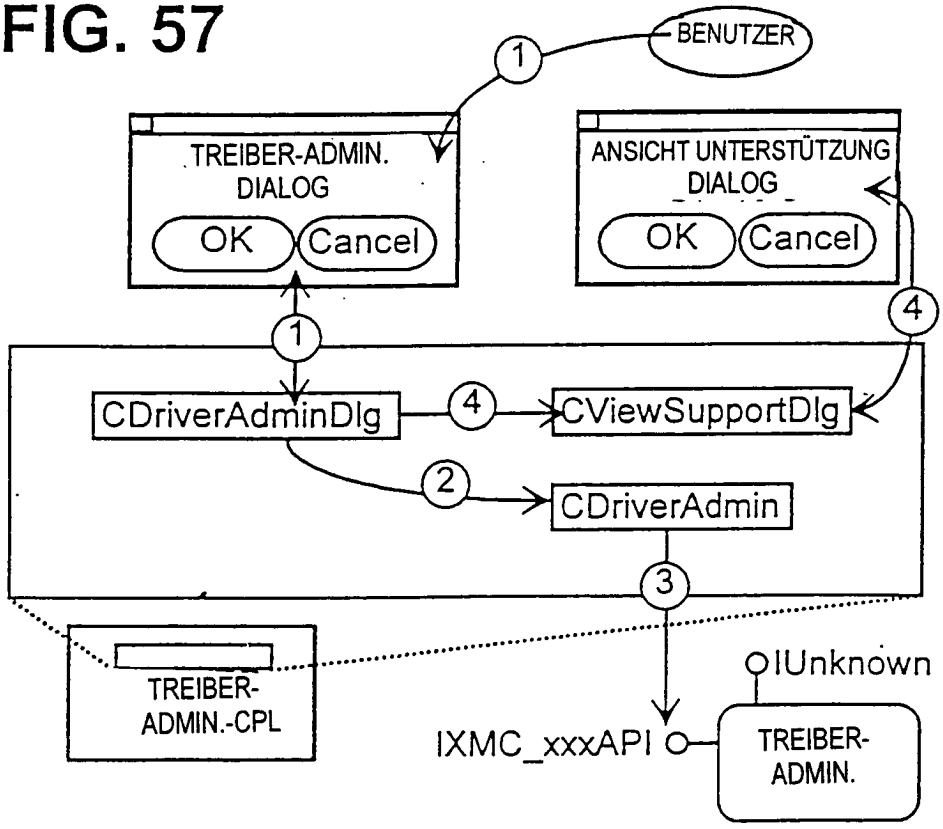

BENUTZER
1
TREIBER-ADMIN.
DIALOG
OK
Cancel
ANSICHT UNTERSTÜTZUNG
DIALOG
OK
Cancel
4
1
CDriverAdminDlg
4
CViewSupportDlg
2
CDriverAdmin
3
TREIBER-
ADMIN.-CPL
IUnknown
IXMC_xxxAPI
TREIBER-
ADMIN.

FIG. 58

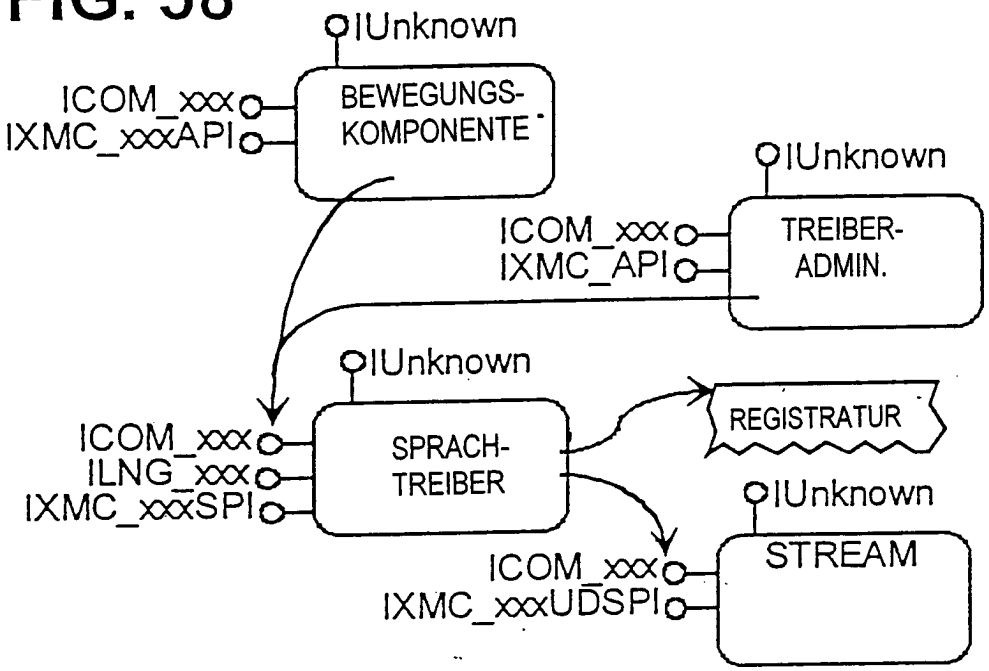

IUnknown
ICOM_xxx
IXMC_xxxAPI
BEWEGUNGS-
KOMPONENTE
IUnknown
ICOM_xxx
IXMC_API
TREIBER-
ADMIN.
IUnknown
ICOM_xxx
ILNG_xxx
IXMC_xxxSPI
SPRACH-
TREIBER
REGISTRATUR
IUnknown
ICOM_xxx
IXMC_xxxUDSPI
STREAM

FIG. 59

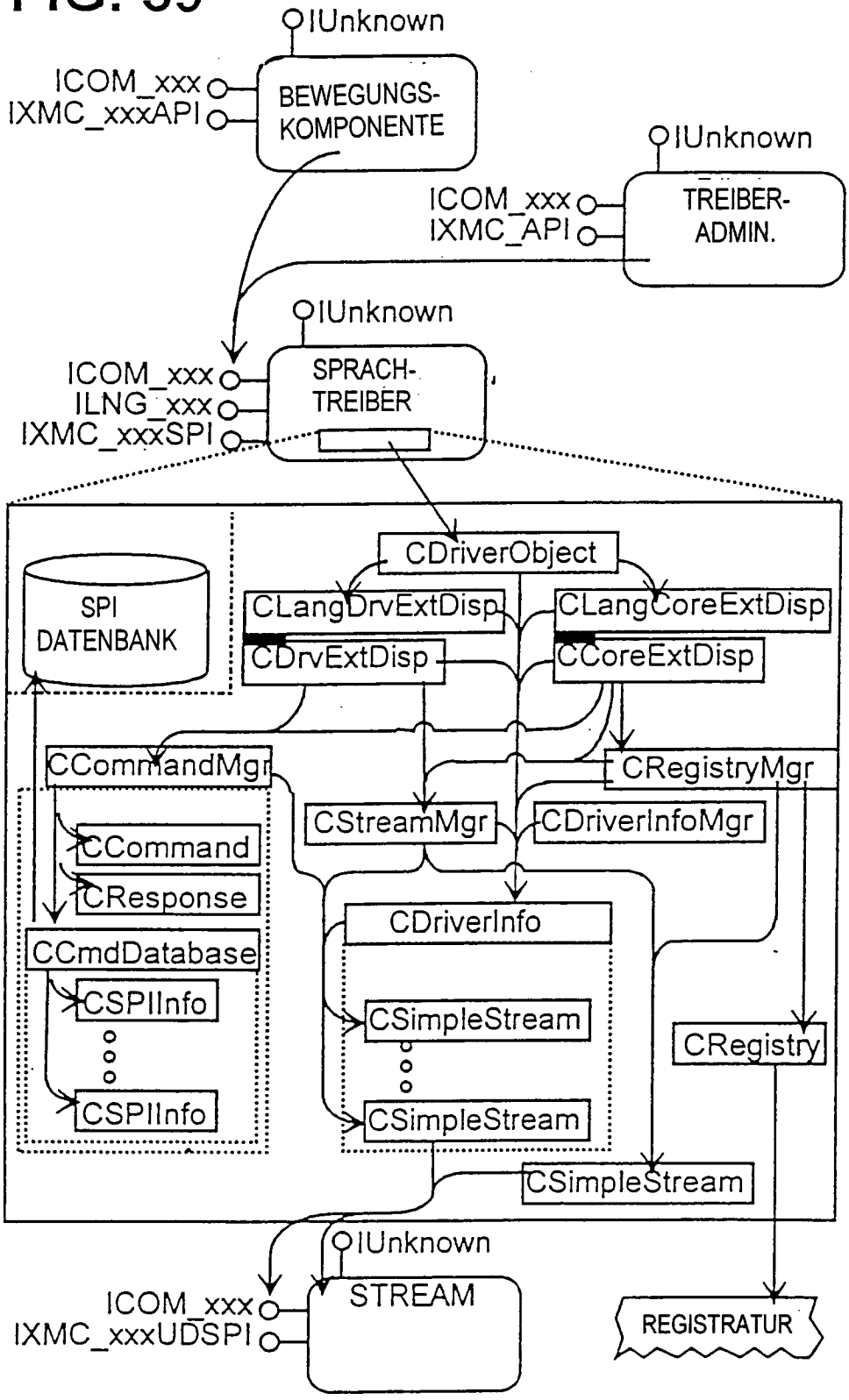

IUnknown
ICOM_xxx
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
IUnknown
ICOM_xxx
IXMC_API
TREIBER-ADMIN.
IUnknown
ICOM_xxx
ILNG_xxx
IXMC_xxxSPI
SPRACH-TREIBER
SPI DATENBANK
CDriverObject
CLangDrvExtDisp
CLangCoreExtDisp
CDrvExtDisp
CCoreExtDisp
CCommandMgr
CRegistryMgr
CStreamMgr
CDriverInfoMgr
CCommand
CResponse
CDriverInfo
CCmdDatabase
CSPIInfo
CSimpleStream
CRegistry
CSPIInfo
CSimpleStream
CSimpleStream
IUnknown
ICOM_xxx
IXMC_xxxUDSPI
STREAM
REGISTRATUR

FIG. 60

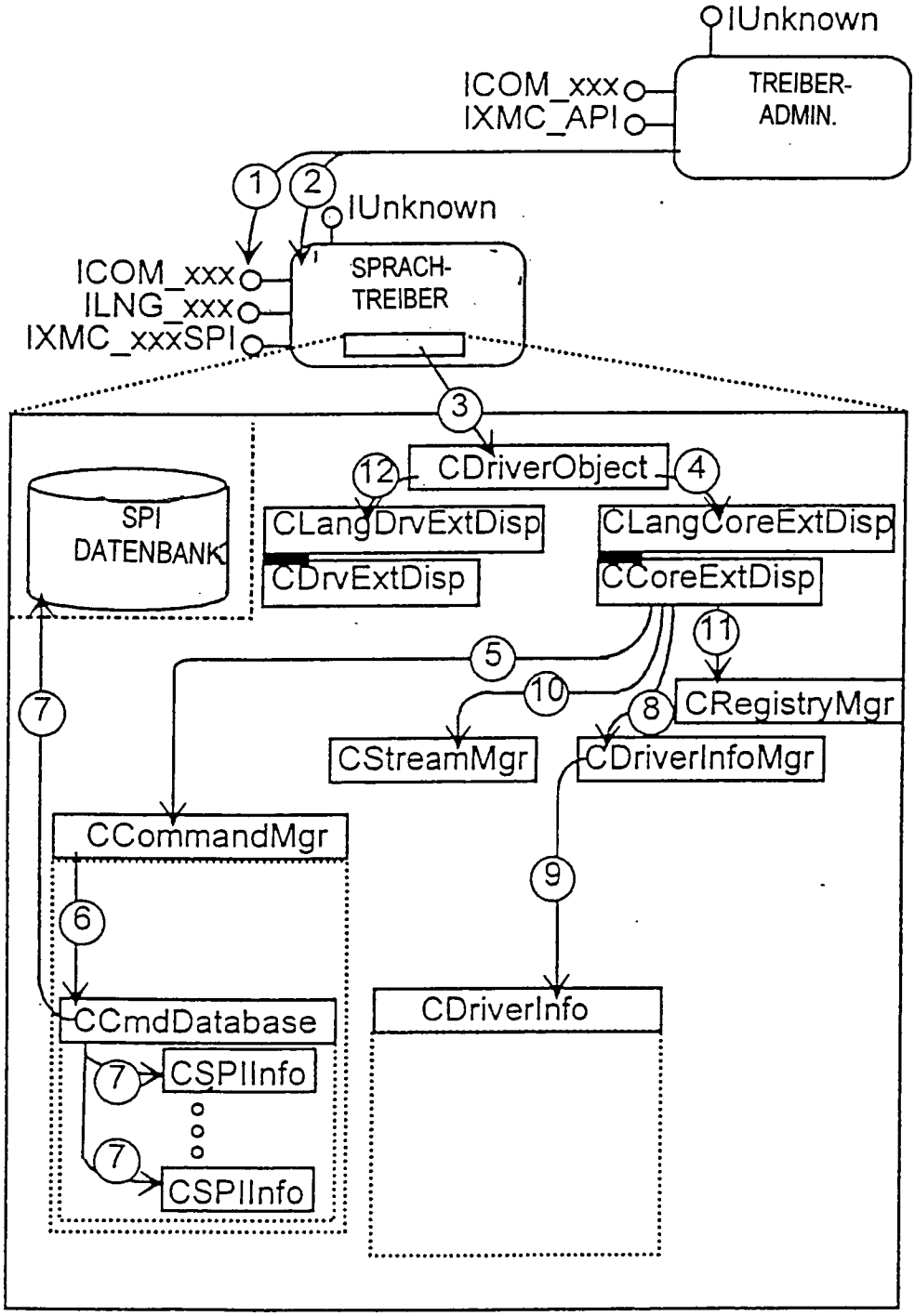

IUnknown
ICOM_xxx
IXMC_API
TREIBER-ADMIN.
1
2
IUnknown
ICOM_xxx
ILNG_xxx
IXMC_xxxSPI
SPRACH-TREIBER
3
12
CDriverObject
4
SPI DATENBANK
CLangDrvExtDisp
CDrvExtDisp
CLangCoreExtDisp
CCoreExtDisp
11
5
10
8
CRegistryMgr
7
CStreamMgr
CDriverInfoMgr
CCommandMgr
9
6
CCmdDatabase
CDriverInfo
7
CSPIInfo
7
CSPIInfo

FIG. 61

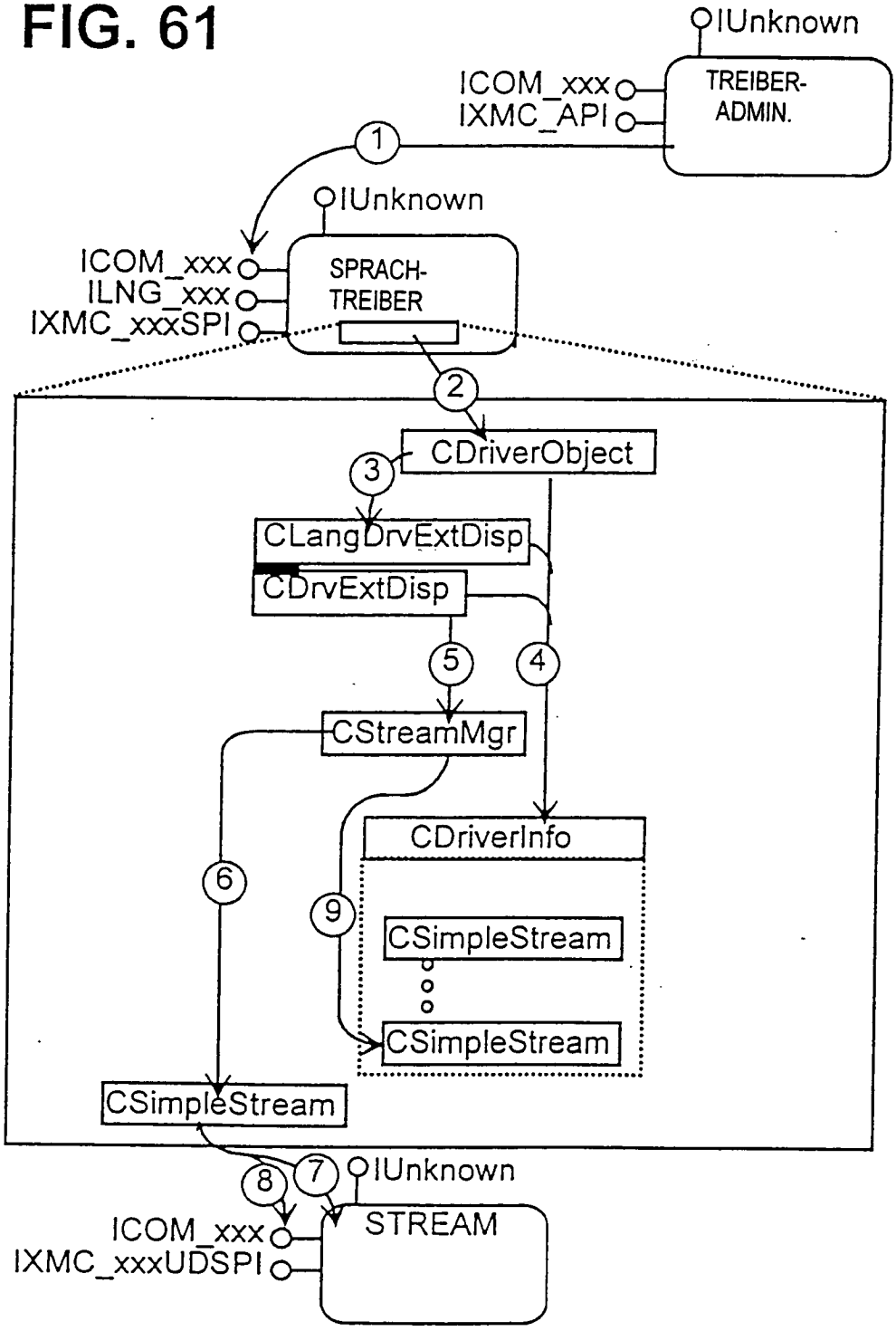

IUnknown
ICOM_xxx
IXMC_API
TREIBER-
ADMIN.
1
IUnknown
ICOM_xxx
ILNG_xxx
IXMC_xxxSPI
SPRACH-
TREIBER
2
CDriverObject
3
CLangDrvExtDisp
CDrvExtDisp
5
4
CStreamMgr
CDriverInfo
6
9
CSimpleStream
CSimpleStream
CSimpleStream
8
7
IUnknown
ICOM_xxx
IXMC_xxxUDSPI
STREAM

FIG. 62

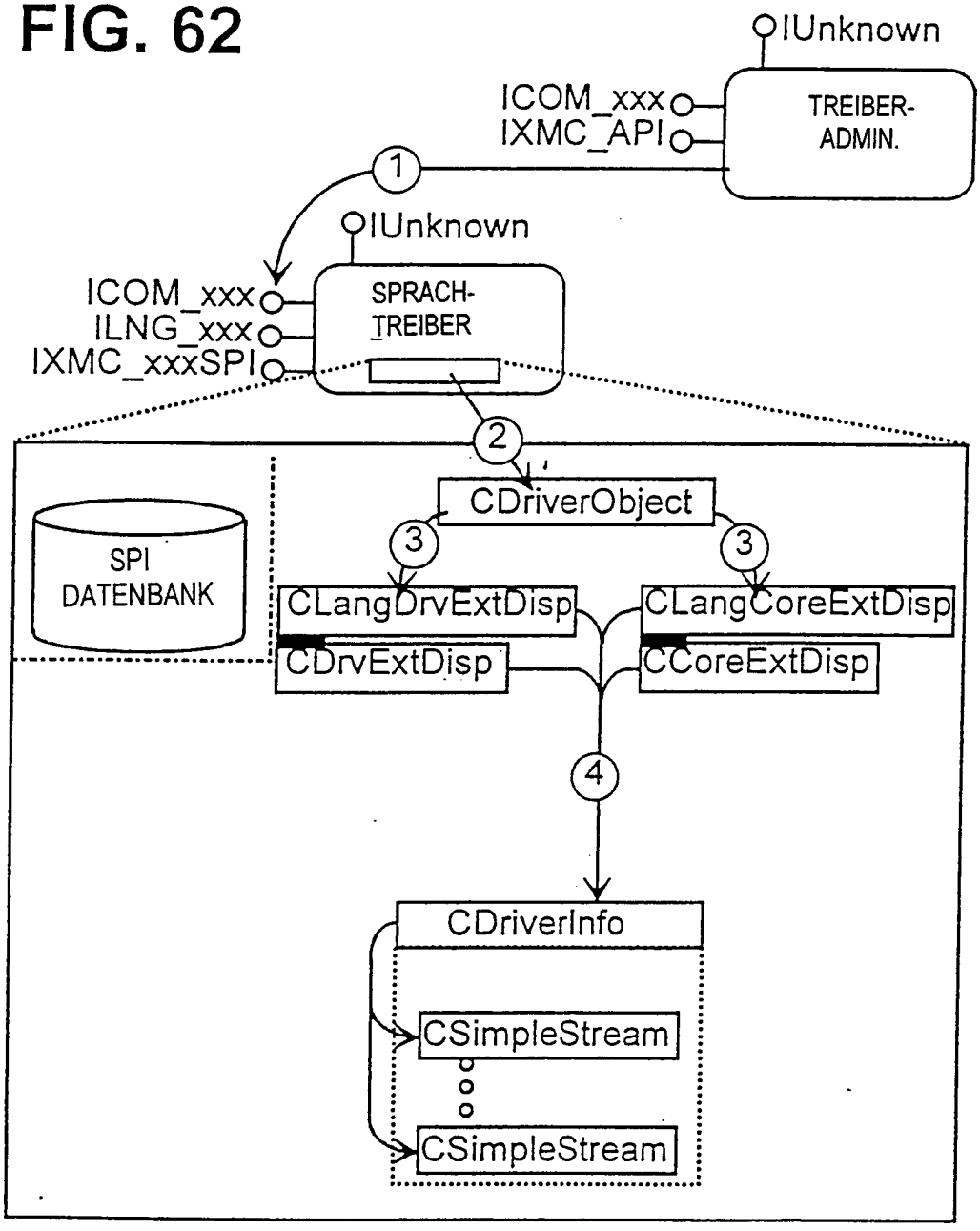

IUnknown
ICOM_xxx
IXMC_API
TREIBER-
ADMIN.
1
IUnknown
ICOM_xxx
ILNG_xxx
IXMC_xxxSPI
SPRACH-
TREIBER
2
CDriverObject
3
3
SPI
DATENBANK
CLangDrvExtDisp
CLangCoreExtDisp
CDrvExtDisp
CCoreExtDisp
4
CDriverInfo
CSimpleStream
CSimpleStream

FIG. 63

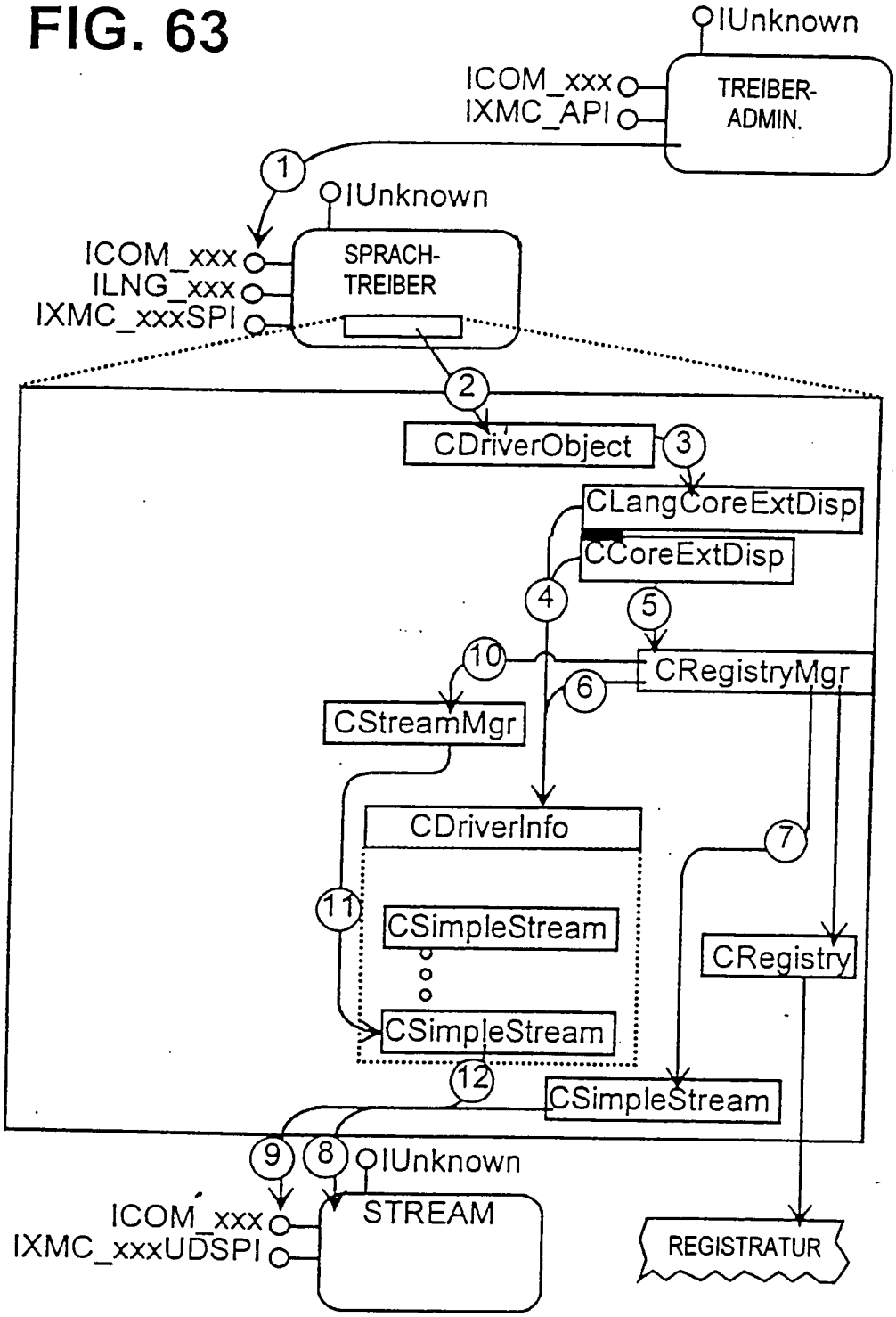

IUnknown
ICOM_xxx
IXMC_API
TREIBER-ADMIN.
1
IUnknown
ICOM_xxx
ILNG_xxx
IXMC_xxxSPI
SPRACH-TREIBER
2
CDriverObject
3
CLangCoreExtDisp
CCoreExtDisp
4
5
10
CRegistryMgr
6
CStreamMgr
CDriverInfo
7
11
CSimpleStream
CRegistry
CSimpleStream
12
CSimpleStream
9
8
IUnknown
ICOM_xxx
IXMC_xxxUDSPI
STREAM
REGISTRATUR

FIG. 64

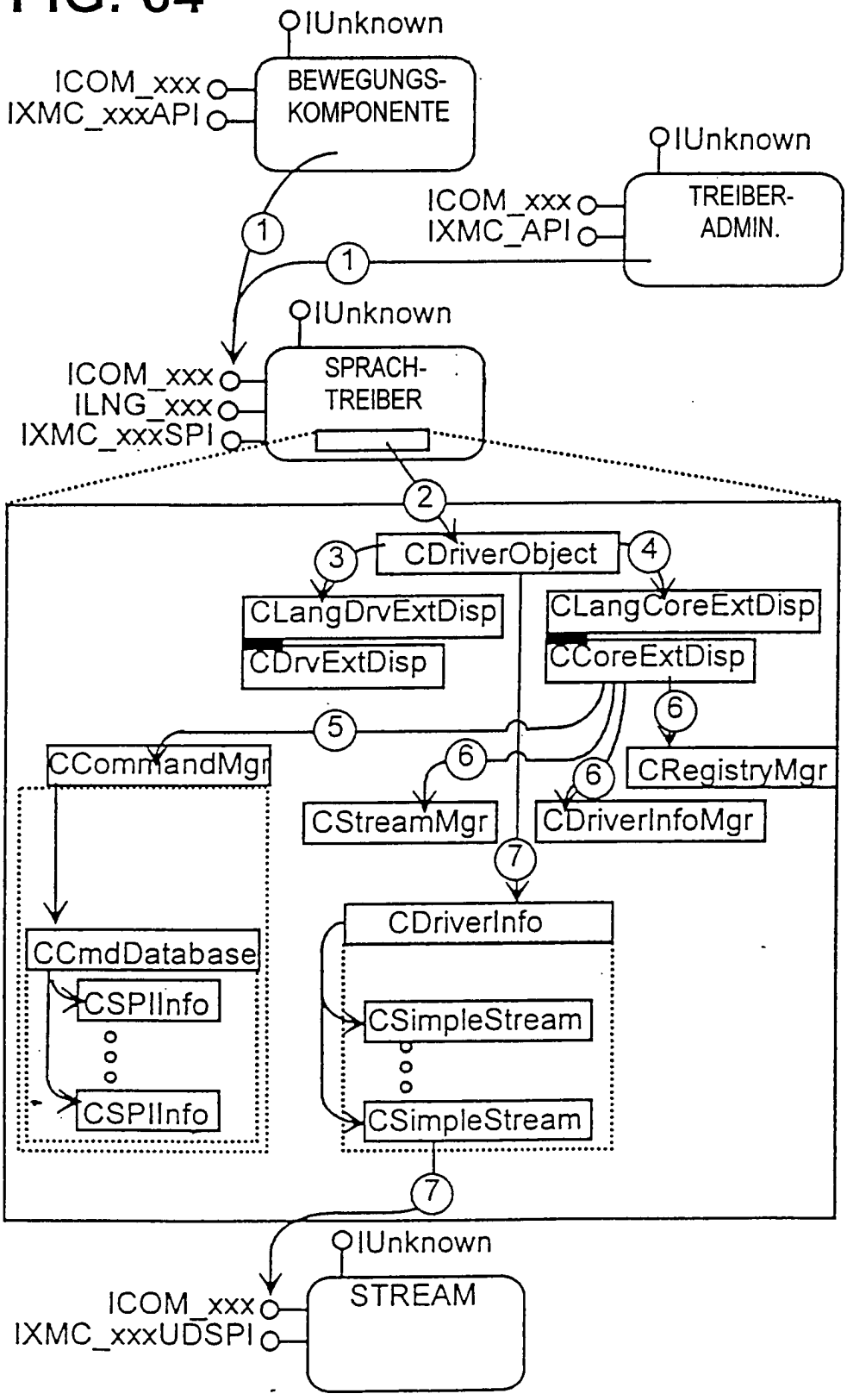

IUnknown
ICOM_xxx
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
IUnknown
ICOM_xxx
IXMC_API
TREIBER-ADMIN.
1
1
IUnknown
ICOM_xxx
ILNG_xxx
IXMC_xxxSPI
SPRACH-TREIBER
2
3
CDriverObject
4
CLangDrvExtDisp
CDrvExtDisp
CLangCoreExtDisp
CCoreExtDisp
5
6
6
6
CCommandMgr
CRegistryMgr
CStreamMgr
CDriverInfoMgr
7
CDriverInfo
CCmdDatabase
CSPIInfo
CSPIInfo
CSimpleStream
CSimpleStream
7
IUnknown
ICOM_xxx
IXMC_xxxUDSPI
STREAM

FIG. 65

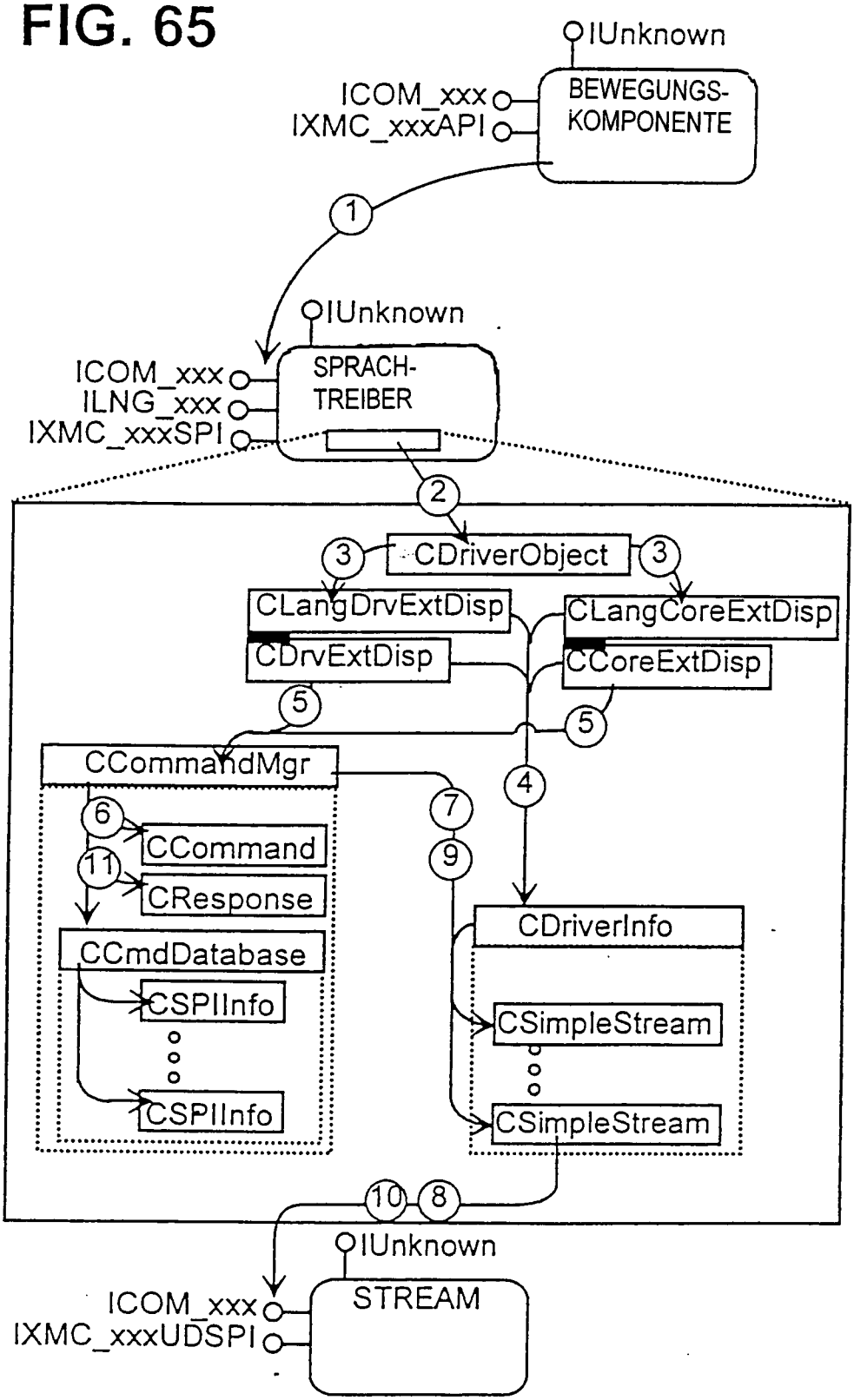
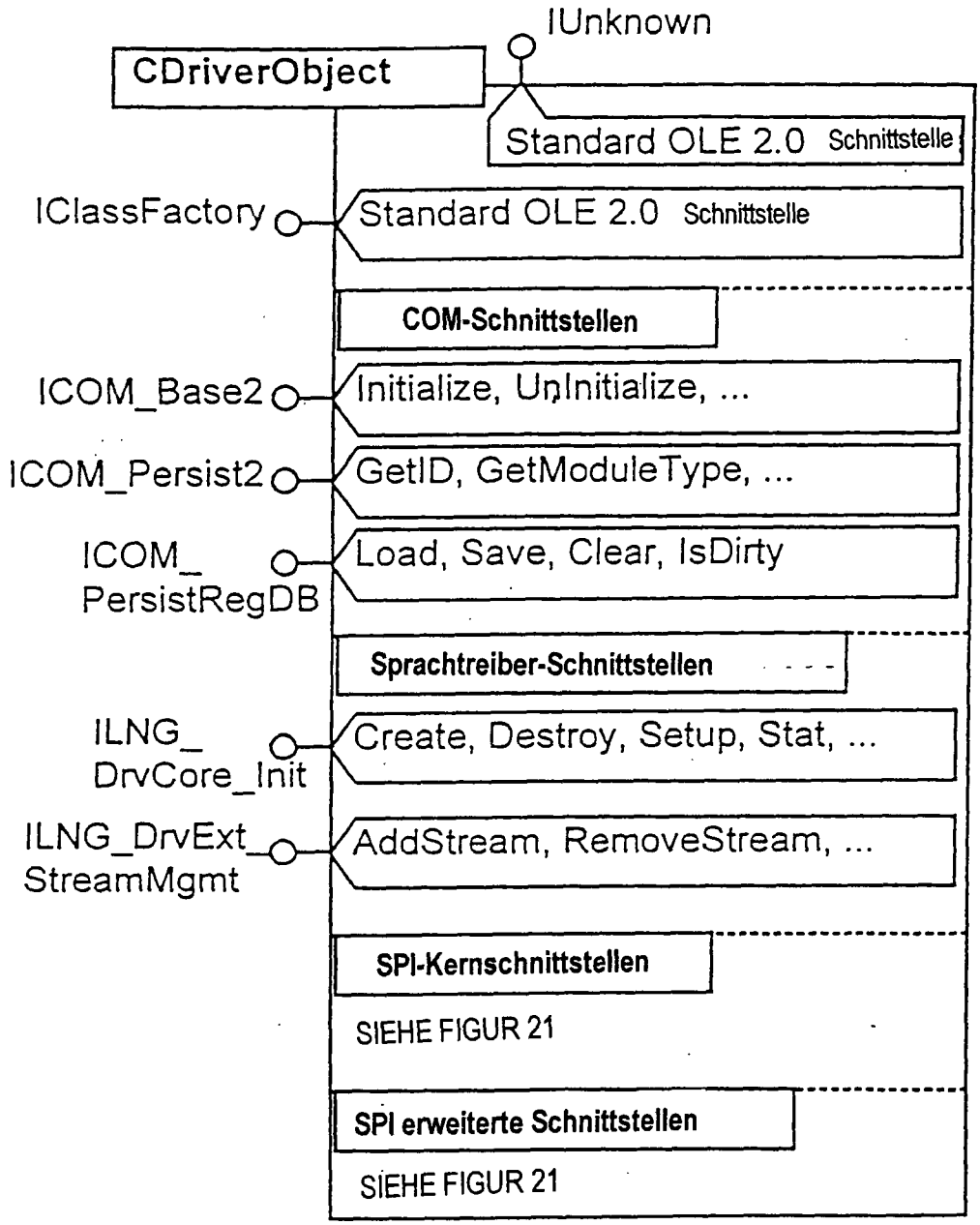
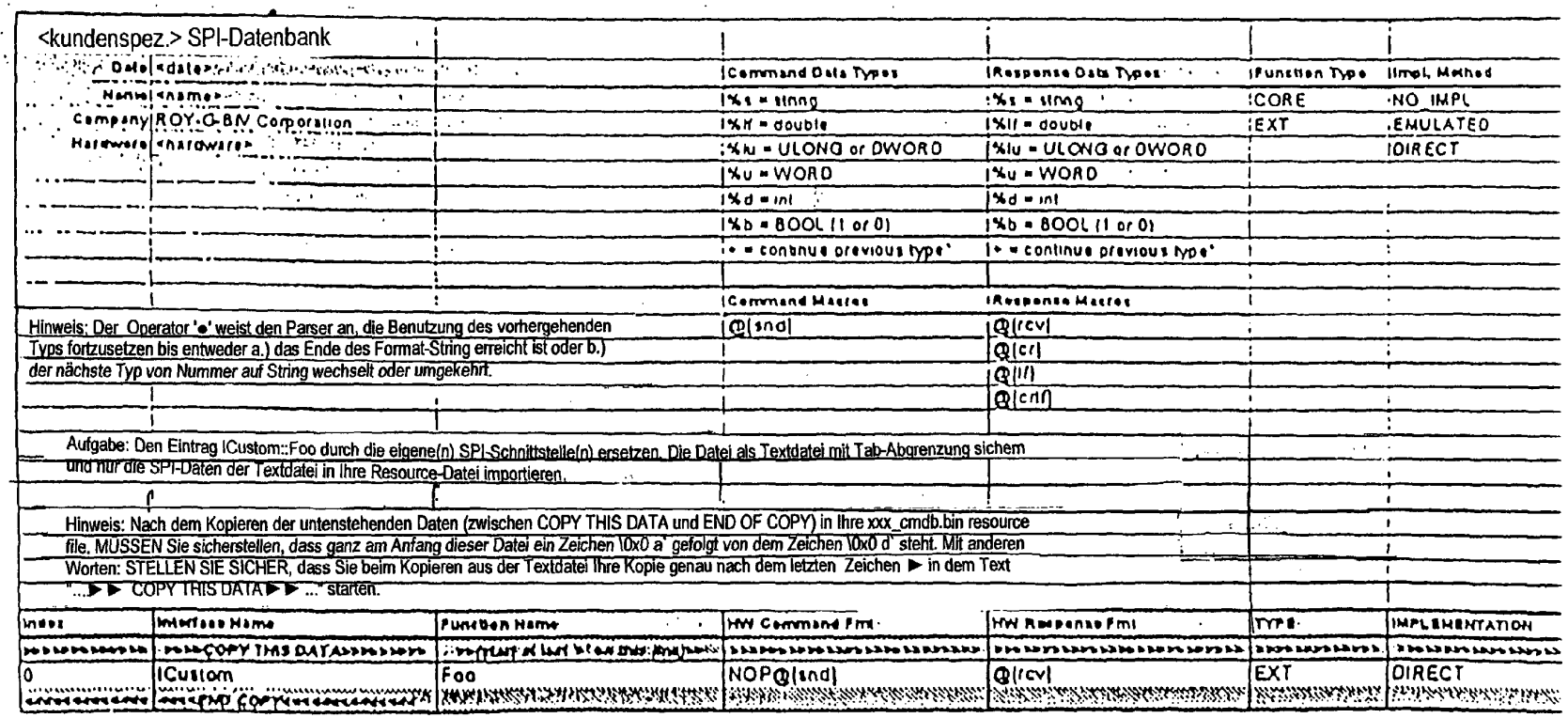

IUnknown
ICOM_xxx
IXMC_xxxAPI
BEWEGUNGS-KOMPONENTE
1
IUnknown
ICOM_xxx
ILNG_xxx
IXMC_xxxSPI
SPRACH-TREIBER
2
3
CDriverObject
3
CLangDrvExtDisp
CLangCoreExtDisp
CDrvExtDisp
CCoreExtDisp
5
5
CCommandMgr
4
6
7
CCommand
11
9
CResponse
CDriverInfo
CCmdDatabase
CSPIInfo
CSimpleStream
CSPIInfo
CSimpleStream
10
8
IUnknown
ICOM_xxx
IXMC_xxxUDSPI
STREAM

FIG. 66


IUnknown
CDriverObject
Standard OLE 2.0 Schnittstelle
IClassFactory
Standard OLE 2.0 Schnittstelle
COM-Schnittstellen
ICOM_Base2
Initialize, UnInitialize, ...
ICOM_Persist2
GetID, GetModuleType, ...
ICOM_ PersistRegDB
Load, Save, Clear, IsDirty
Sprachtreiber-Schnittstellen
ILNG_ DrvCore_Init
Create, Destroy, Setup, Stat, ...
ILNG_DrvExt_ StreamMgmt
AddStream, RemoveStream, ...
SPI-Kernschnittstellen
SIEHE FIGUR 21
SPI erweiterte Schnittstellen
SIEHE FIGUR 21

DE 696 35 094 T2 2006.04.20



<kundenspez.> SPI-Datenbank						
Date	<date>		Command Data Types	Response Data Types	Function Type	Impl. Method
Name	<name>		%s = string	%s = string	CORE	NO_IMPL
Company	ROY-G-BIV Corporation		%lf = double	%lf = double	EXT	EMULATED
Hardware	<hardware>		%lu = ULONG or DWORD	%lu = ULONG or DWORD		DIRECT
			%u = WORD	%u = WORD		
			%d = int	%d = int		
			%b = BOOL (1 or 0)	%b = BOOL (1 or 0)		
			+ = continue previous type*	+ = continue previous type*		
			Command Macros	Response Macros		
Hinweis: Der Operator '+' weist den Parser an, die Benutzung des vorhergehenden			@[snd]	@[rcv]		
Typs fortzusetzen bis entweder a.) das Ende des Format-String erreicht ist oder b.)				@[cr]		
der nächste Typ von Nummer auf String wechselt oder umgekehrt.				@[lf]		
				@[crlf]		
Aufgabe: Den Eintrag !Custom::Foo durch die eigene(n) SPI-Schnittstelle(n) ersetzen. Die Datei als Textdatei mit Tab-Abgrenzung sichern						
und nur die SPI-Daten der Textdatei in Ihre Resource-Datei importieren.						
Hinweis: Nach dem Kopieren der untenstehenden Daten (zwischen COPY THIS DATA und END OF COPY) in Ihre xxx_cmdb.bin resource						
file, MÜSSEN Sie sicherstellen, dass ganz am Anfang dieser Datei ein Zeichen \0x0 a` gefolgt von dem Zeichen \0x0 d` steht. Mit anderen						
Worten: STELLEN SIE SICHER, dass Sie beim Kopieren aus der Textdatei Ihre Kopie genau nach dem letzten Zeichen ▶ in dem Text						
"...▶▶ COPY THIS DATA▶▶ ..." starten.						
Index	Interface Name	Function Name	HW Command Fmt	HW Response Fmt	TYPE	IMPLEMENTATION
0	!Custom	Foo	NOP@[snd]	@[rcv]	EXT	DIRECT

Tabellenblatt einer Excel Befehlsdatenbank

FIG. 67