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

(10) **DE 697 35 351 T2** 2006.11.30

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 0 929 961 B1**
(21) Deutsches Aktenzeichen: **697 35 351.6**
(86) PCT-Aktenzeichen: **PCT/US97/17407**
(96) Europäisches Aktenzeichen: **97 909 883.7**
(87) PCT-Veröffentlichungs-Nr.: **WO 1998/015092**
(86) PCT-Anmeldetag: **02.10.1997**
(87) Veröffentlichungstag der PCT-Anmeldung: **09.04.1998**
(97) Erstveröffentlichung durch das EPA: **21.07.1999**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **01.03.2006**
(47) Veröffentlichungstag im Patentblatt: **30.11.2006**

(51) Int Cl.[8]: ***H04L 29/06*** (2006.01)
***G06F 19/00*** (2006.01)

(30) Unionspriorität:
**720882** **04.10.1996** **US**

(84) Benannte Vertragsstaaten:
**DE**

(73) Patentinhaber:
**Eastman Kodak Co., Rochester, N.Y., US**

(74) Vertreter:
**WAGNER & GEYER Partnerschaft Patent- und Rechtsanwälte, 80538 München**

(72) Erfinder:
**SIEFFERT, J., Kent, Saint Paul, MN 55164-0898, US; IHLENFELDT, R., Andrew, Saint Paul, MN 55164-0898, US**

(54) Bezeichnung: **SYSTEM ZUR ÜBERMITTLUNG VON BILDINFORMATIONEN ÜBER EIN NETZWERK ZWISCHEN BEBILDERUNGSVORRICHTUNGEN, DIE NACH MEHREREN PROTOKOLLEN ARBEITEN**

## Beschreibung

**[0001]** Die vorliegende Erfindung betrifft Abbildungssysteme und insbesondere Systeme zur Übermittlung von Bildinformationen zwischen einer Eingabe-Bebilderungsvorrichtung und einer Ausgabe-Bebildungsvorrichtung in einer Netzwerkumgebung.

**[0002]** Ein Bilderzeugungssystem umfasst üblicherweise eine Eingabe-Bebilderungsvorrichtung, die Bildinformationen erzeugt, und eine Ausgabe-Bebilderungsvorrichtung, die anhand von Bildinformationen eine sichtbare Darstellung des Bildes, abhängig von den Bildinformationen, erzeugt. In einem medizinischen Bebilderungssystem kann die Eingabe-Bebilderungsvorrichtung beispielsweise eine Magnetresonanz- (MR), Computertomographie- (CT), herkömmliche Radiographie- (Röntgen) oder Ultraschallvorrichtung sein. Alternativ hierzu kann die Eingabe-Bebilderungsvorrichtung eine Benutzerschnittstellen-Einrichtung umfassen, beispielsweise eine Tastatur, eine Maus oder einen Trackball, die auch zum Erzeugen medizinischer Bildinformationen in der Lage ist. Als weitere Alternative kann die Eingabe-Bebilderungsvorrichtung eine Bildarchiv-Arbeitsstation zum Abrufen archivierter Bildinformationen umfassen. Die Ausgabe-Bebilderungsvorrichtung in einem medizinischen Bebilderungssystem umfasst üblicherweise einen digitalen Laserbelichter. Der Laserbelichter belichtet ein Bebilderungsmedium abhängig von den Bildinformationen zum Herstellen einer sichtbaren Darstellung.

**[0003]** Die von der Eingabe-Bebilderungsvorrichtung erzeugten Bildinformationen umfassen Bilddaten, die digitale, das Bild darstellende Bildwerte enthalten, sowie Bebilderungsanforderungen, die von dem Laserbelichter durchzuführende Operationen bezeichnen. Jeder dieser digitalen Bildwerte entspricht einem Pixel aus einer Vielzahl von Pixeln in dem Originalbild und stellt eine optische Dichte dar, die dem jeweiligen Pixel zugeordnet ist. Abhängig von einer Bebilderungsanforderung wandelt der Laserbelichter die digitalen Bildwerte um, um Laseransteuerungswerte zu erzeugen, die zur Modulation der Intensität eines Abtastlasers dienen. Die Laseransteuerungswerte werden berechnet, um auf dem Bebilderungsmedium Belichtungswerte zu erzeugen, die notwendig sind, um die optischen Dichten zu reproduzieren, die den Pixeln des Originalbildes zugeordnet werden, wenn das Medium entwickelt wird, und zwar entweder durch eine chemische Nassverarbeitung oder durch eine thermische Trockenverarbeitung. Das Laserabbildungsgerät kann eine Anzahl zusätzlicher Operationen in Abhängigkeit von den Bebilderungsanforderungen ausführen, die von der Eingabe-Bebilderungsvorrichtung erzeugt werden. Beispielsweise kann das Laserabbildungsgerät vor dem Erzeugen der Laseransteuerungswerte die Bilddaten manipulieren, um eine Vielzahl verschiedener Formate und/oder Anzeigeeigenschaften zu erstellen.

**[0004]** Die von dem Laserabbildungsgerät verarbeiteten Bildinformationen haben ein Format, das durch ein Eingabeprotokoll bestimmt wird, das der jeweiligen Eingabe-Bebilderungsvorrichtung zugeordnet ist. Medizinische Bebilderungssysteme sind üblicherweise in der Lage, Bildinformationen zu handhaben, die nach verschiedenen, unterschiedlichen Eingabeprotokollen erzeugt worden sind. Ein Eingabeprotokoll kann als Netzwerk-Treiberprotokoll gekennzeichnet sein, das Kommunikationsspezifikationen auf unterer Ebene für eine bestimmte Eingabe-Bebilderungsvorrichtung bereitstellt, und ein Netzwerk-Interpreterprotokoll, das das Format zur Interpretation der von der Eingabe-Bebilderungsvorrichtung erzeugten Bildinformationen ermittelt. Die Zahl der verschiedenen Eingabeprotokolle ergibt sich in gewissem Maße aus den verschiedenen Typen von derzeit verwendeten Eingabe-Bebilderungsvorrichtungen, wie Magnetresonanz- (MR), Computertomographie- (CT), herkömmlichen Radiographie- (Röntgen) oder Ultraschallvorrichtungen, die unter Umständen jeweils Bildinformationen nach einem anderen Protokoll erzeugen. Die Hauptquelle für unterschiedliche Eingabeprotokolle ist jedoch das Vorhandensein einer Reihe von Modalitäten, d.h. Eingabe-Bebilderungsvorrichtungen, die von verschiedenen Herstellern stammen und eigene, herstellerspezifische Eingabeprotokolle aufweisen. Hersteller, wie Siemens, Toshiba, GE und Picker, stellen derzeit CT-Eingabe-Bebilderungsvorrichtungen her, die eine ähnliche Funktionalität bereitstellen, aber die Bildinformationen nach unterschiedlichen, modalitätsspezifischen Eingabeprotokollen erzeugen.

**[0005]** Neben der Fähigkeit, mehrere Eingabeprotokolle zu verarbeiten, sind medizinische Bebilderungssysteme üblicherweise in der Lage, die Kommunikation der Bildinformationen mit Ausgabe-Bebilderungsvorrichtungen nach mehreren Ausgabeprotokollen abzuwickeln. Wie ein Eingabeprotokoll kann auch ein Ausgabeprotokoll dadurch gekennzeichnet sein, dass es ein Ausgabe-Treiberprotokoll umfasst, das die Anforderungen zur Kommunikation mit einer bestimmten Ausgabe-Bebilderungsvorrichtung sowie ein Ausgabe-Interpreterprotokoll umfasst, das das Format zur Übersetzung der Bildinformationen in eine Form ermittelt, die von der Ausgabe-Bebilderungsvorrichtung verstanden wird. Der Hauptgrund für unterschiedliche Ausgabeprotokolle ist die Verfügbarkeit von Laser-Bebilderungsvorrichtungen mit unterschiedlichen Funktionsmengen. Diese unterschiedlichen Funktionsmengen stellen eine wechselnde Komplexität dar, die zu verschiedenen Ausgabeprotokollen führt. Beispielsweise bietet die Imation Enterprise Corp. ("Imation") aus Oakdale, Minnesota, USA,

derzeit Laserabbildungsgeräte an, die über unterschiedliche Funktionsmengen verfügen, die als "831", "952" und als "SuperSet" bezeichnet werden, und denen jeweils ein bestimmtes Ausgabeprotokoll zugeordnet ist.

**[0006]** Bestehende medizinische Bebilderungssysteme beinhalten derzeit mehrere Eingabe- und Ausgabeprotokolle auf Ad-hoc-Basis durch Konstruktion von Punkt-zu-Punkt-Hardware- und/oder Softwareschnittstellen, die speziell für ein bestimmtes Eingabeprotokoll und ein bestimmtes Ausgabeprotokoll konfiguriert sind. Die Verwendung einer speziell hergestellten Schnittstelle ist äußerst inflexibel. Wenn später die Kommunikation mit einer anderen Eingabe-Bebilderungsvorrichtung benötigt wird, muss die gesamte Schnittstelle neu konstruiert werden, um die Beziehung zwischen dem neuen Eingabeprotokoll und dem alten Ausgabeprotokoll abzuwickeln. Eine Änderung der Ausgabe-Bebilderungsvorrichtung erfordert ebenfalls die Neukonstruktion der Schnittstelle zur Handhabung der Beziehung zwischen dem neuen Ausgabeprotokoll und dem alten Eingabeprotokoll. Leider ist die Neukonstruktion der Schnittstelle eine umständliche Aufgabe, die oft erhebliche Investitionen in Hardware- und/oder Softwareentwicklungszeit erfordert. Auch anscheinend geringfügige Änderungen an der Funktionalität einer Eingabe- oder Ausgabe-Bebilderungsvorrichtung können zahlreiche, kostspielige konstruktive Änderungen erforderlich machen, die sich durch die gesamte Schnittstelle ziehen.

**[0007]** Eine Lösung dieser Probleme wird in der Hauptanmeldung US-A-5,630,101 mit dem Titel "System for Communication of Image Information Between Multiple-Protocol Imaging Device" beschrieben. Das in dieser Patentanmeldung beschriebene System verfolgt eine objektorientierte, modulare Konstruktion, um eine softwaregestützte Architektur mit direkter Verbindung vorzusehen, die in Bezug auf die Kommunikation mit dem Laserabbildungsgerät eine erhebliche Flexibilität ermöglicht. Eine Schnittstellenausführungskomponente instanziiert das benötigte Paar aus Eingabetreiber und Eingabeinterpreter sowie das benötigte Paar aus Ausgabeinterpreter und Ausgabetreiber, um eine Pipeline zu erzeugen, so dass eine bestimmte Hostmodalität mit einem bestimmten Laserabbildungsgerät kommunizieren kann. Die jeweiligen Komponenten aus Eingabetreiber, Eingabeinterpreter, Ausgabeinterpreter und Ausgabetreiber stellen ein diskretes Softwareobjekt oder eine „Blackbox" dar. Auf diese Weise kann jede Komponente modifiziert oder durch ein neues Objekt ersetzt werden, ohne die Leistung der übrigen Komponenten oder der gesamten Pipeline zu beeinträchtigen. Beispielsweise kann das Paar aus Eingabetreiber und Eingabeinterpreter speziell für eine Siemens-Hostmodalität vorgesehen sein, während das Paar aus Ausgabeinterpreter und Ausgabetreiber speziell für einen Imation-Laserbelichter vorgesehen sein kann, der das Protokoll 831 verwendet. Wenn das letztgenannte Paar durch ein Paar ersetzt wird, das für ein Imation-Laserabbildungsgerät gedacht ist, das mit dem SuperSet-Protokoll arbeitet, ist die Konstruktion der Komponenten derart beschaffen, dass das Paar aus Eingabetreiber und Eingabeinterpreter nicht ebenfalls ersetzt zu werden braucht.

**[0008]** Obwohl US-A-5,630,101 mehr Flexibilität in der Architektur von Laserabbildungsgeräten fördert, beschreibt auch diese Anmeldung nur eine direktverbundene Punkt-zu-Punkt-Architektur. Für jedes Eingabe-Ausgabe-Paar muss die Schnittstellenausführungskomponente ein separates Paar aus Eingabetreiber und Eingabeinterpreter sowie ein Paar aus Ausgabeinterpreter und Ausgabetreiber instanziieren. Die Schnittstellenausführungskomponente muss daher eine separate Pipeline zwischen jeder Hostmodalität und jedem Laserabbildungsgerät herstellen. Zwar ist dies in einem System mit einer relativ kleinen Zahl von Hostmodalitäten nicht unbedingt bedenklich, aber es kann in Umgebungen problematisch sein, in denen eine erhebliche Anzahl von Hostmodalitäten mit einer Vielzahl unterschiedlicher Laserabbildungsgeräte kommuniziert. Dies gilt insbesondere in einer Netzwerkumgebung, in der üblicherweise eine Reihe von Netzwerk-Clients das gleiche Protokoll verwenden. In einer derartigen Situation ist es wünschenswert, dass keine redundanten Paare aus Eingabetreiber und Eingabeinterpreter für jeden Client vorhanden sind. Neben der Beanspruchung von Ressourcen belastet diese Architektur die Schnittstellenausführungskomponente zudem mit einem hohen Overhead.

**[0009]** Es besteht somit zunehmend Bedarf nach flexibleren medizinischen Bebilderungssystemen, die in der Lage sind, die Kommunikation zwischen einer Vielzahl von Eingabe- und Ausgabe-Bebilderungsvorrichtungen mit mehreren Protokollen abzuwickeln. Es ist wünschenswert, dass diese medizinischen Bebilderungssysteme nicht nur in Bezug auf die vorhandenen Protokolle flexibel sind, sondern auch zukünftige Protokolle in kostengünstiger Weise nutzen können. Es besteht zudem zunehmender Bedarf nach der Netzwerkübermittlung von Bildinformationen zwischen Eingabe- und Ausgabe-Bebilderungsvorrichtungen. Im Bereich der medizinischen Bebilderung haben beispielsweise das American College of Radiology (ACR) und die National Electrical Manufacturers Association (NEMA) einen gemeinsamen Ausschuss zur Entwicklung eines Standards für die digitale Bebilderung und Kommunikation in der Medizin gegründet, der als DICOM-Protokoll bekannt ist. Das DICOM-Protokoll wurde entworfen, um die Connectivity unter medizinischen Geräten zu ermöglichen, insbesondere mit Blick auf den Entwicklungstrend in Krankenhäusern, der eine Abkehr von Punkt-zu-Punkt-Umgebungen und eine Hinwendung zu Netzwerkumgebungen vorsieht. Hersteller medizinischer Geräte beginnen jetzt branchenweit mit der Implementierung des DICOM-Kommunikationsprotokolls. Das DICOM-Protokoll setzt ei-

nen Standard für die Netzwerkkommunikation von Bildinformationen. Doch auch andere Netzwerkprotokolle sind vorhanden und werden weiterhin entwickelt werden. Es besteht somit weiterhin Bedarf nach einer Protokollübersetzung in Netzwerksystemen. Der Bedarf nach Protokollübersetzung in Netzwerksystemen begründet Probleme, die mit denen in Punkt-zu-Punkt-Systemen vergleichbar sind. Insbesondere sind Flexibilität und einfache Anpassung an mehrere Protokolle weiterhin kritisch. Es besteht daher Bedarf nach einem System, das in der Lage ist, Bildinformationen zwischen Bebilderungsvorrichtungen gemäß mehreren Kommunikationsprotokollen zu übermitteln.

**[0010]** US-A-5,060,140 beschreibt ein universell programmierbares Datenkommunikations-Verbindungssystem, das benutzerseitig programmierbar ist, um einen ausgewählten Datenweg zwischen einer oder mehreren Datenquellen und einem oder mehreren Datenzielen bereitzustellen. Das Datenkommunikationssystem ermöglicht dem Benutzer, Signale von der Quelle zum Ziel mithilfe einfacher Befehle zu "verbinden". Das beschriebene System betrifft nicht die Lösung von Problemen, die die Übermittlung medizinischer Informationen zwischen unterschiedlichen medizinischen Bebilderungsmodalitäten und mindestens einem Abbildungsgerät aus einer Vielzahl von Abbildungsgeräten betreffen und insbesondere das Problem mehrerer medizinischer Bebilderungsmodalitäten unter Verwendung des gleichen Netzwerkschnittstellenprotokolls zur Kommunikation mit einem der Abbildungsgeräte über eine einzelne Kommunikationspipeline.

**[0011]** Die vorliegende Erfindung betrifft ein System zum Übermitteln medizinischer Bildinformationen zwischen verschiedenen medizinischen Abbildungsmodalitäten und mindestens einem aus einer Vielzahl von unterschiedlichen Abbildungsgeräten über eine Netzwerkschnittstelle. Das System umfasst eine Netzwerkausführungskomponente, eine oder mehrere Ausgabeschnittstellenkomponenten und eine Schnittstellenausführungskomponente.

**[0012]** Die Netzwerkausführungskomponente instanziiert eine oder mehrere Netzwerkschnittstellenkomponenten gemäß ausgewählten Netzwerkschnittstellenprotokollen. Jede Netzwerkschnittstellenkomponente ist derart ausgebildet, dass sie medizinische Bildinformationen von einer der medizinischen Abbildungsmodalitäten über die Netzwerkschnittstelle empfängt, wobei die medizinischen Bildinformationen gemäß dem ausgewählten Netzwerk-Schnittstellenprotokoll empfangen werden. Jedes Netzwerkschnittstellenprotokoll ist ausgewählten medizinischen Abbildungsmodalitäten speziell zugeordnet. Erste Abbildungsanforderungen werden auf der Grundlage der empfangenen medizinischen Bildinformationen und gemäß dem ausgewählten Netzwerkschnittstellenprotokoll erzeugt.

**[0013]** Jede der einen oder mehreren Ausgabeschnittstellenkomponenten ist derart ausgebildet, dass sie zweite Abbildungs- oder Bebilderungsanforderungen auf der Grundlage der ersten, von einer der Netzwerkschnittstellenkomponenten erzeugten Bebilderungsanforderungen erzeugt, Die zweiten Bebilderungsanforderungen werden gemäß einem aus einer Vielzahl unterschiedlicher Ausgabeschnittstellenprotokolle erzeugt. Jedes der Ausgabeschnittstellenprotokolle ist einem der Abbildungsgeräte speziell zugeordnet. Die zweiten, von einer der Ausgabeschnittstellenkomponenten erzeugten Bebilderungsanforderungen werden zu einem der Abbildungsgeräte übermittelt, und die zweiten Bebilderungsanforderungen werden gemäß dem einen der Ausgabeschnittstellenprotokolle übermittelt.

**[0014]** Eine Schnittstellenausführungskomponente bildet eine oder mehrere Übermittlungsleitungen, von denen jede Leitung eine oder mehrere medizinische Abbildungsmodalitäten mit einer der Netzwerkschnittstellenkomponenten kommunikativ verbindet unter Verwendung des gleichen Netzwerkschnittstellenprotokolls, einer der Ausgabeschnittstellenkomponenten und eines der Abbildungsgeräte. Dadurch können mehrere medizinische Abbildungsmodalitäten unter Verwendung des gleichen Netzwerkschnittstellenprotokolls mit einem der Abbildungsgeräte über eine einzelne Übermittlungsleitung kommunizieren.

**[0015]** Die vorliegende Erfindung weist eine Reihe von Vorteilen in Bezug auf die Bereitstellung der Kommunikation zwischen den Eingabe-Bebilderungsvorrichtungen und den Laserabbildungsgeräten auf. Weil die Netzwerkausführungskomponenten jeweils die Kommunikation mit einer Reihe von Eingabe-Bebilderungsvorrichtungen ermöglichen können, ist keine separate Leitung für jede medizinische Abbildungsmodalität erforderlich, wodurch Ressourcen geschont werden. Die Netzwerkausführungskomponenten ermöglichen zudem die erfindungsgemäße Kommunikation zwischen medizinischen Abbildungsmodalitäten und Abbildungsgeräten auf Netzwerkebene im Unterschied zu einer direkten Anschlussweise. Den Netzwerkausführungskomponenten wird von der Schnittstellenausführungskomponente zudem die Zuständigkeit bezüglich der Überwachung der Übermittlung von den medizinischen Abbildungsmodalitäten übertragen. Dadurch wird die Schnittstellenausführungskomponente von der diesbezüglichen Zuständigkeit entlastet.

**[0016]** Andere und weitere Ausführungsbeispiele, Aspekte und Vorteile der vorliegenden Erfindung werden anhand der folgenden Beschreibung und unter Bezug auf die anliegenden Zeichnungen deutlich.

**[0017]** Die Erfindung wird im folgenden anhand in der Zeichnung dargestellter Ausführungsbeispiele näher erläutert.

**[0018]** Es zeigen

**[0019]** Fig. 1 ein Funktionsblockdiagramm eines medizinischen Bebilderungssystems zur Übermittlung von Bildinformationen zwischen Mehrprotokoll-Bebilderungsvorrichtungen in einer Netzwerkkommunikationsumgebung gemäß der vorliegenden Erfindung;

**[0020]** Fig. 2 ein Funktionsblockdiagramm eines alternativen medizinischen Bebilderungssystems zur Übermittlung von Bildinformationen zwischen Mehrprotokoll-Bebilderungsvorrichtungen in einer Netzwerkkommunikationsumgebung gemäß einem weiteren Ausführungsbeispiel der vorliegenden Erfindung;

**[0021]** Fig. 3 ein Funktionsblockdiagramm zur Darstellung eines Subsystems des medizinischen Bebilderungssystems aus Fig. 1;

**[0022]** Fig. 4 ein Diagramm zur Darstellung der objektorientierten Protokollhierarchie, die die Austauschbarkeit der Netzwerkprotokollkomponenten ermöglicht, einschließlich der Netzwerktreiberkomponente und der Netzwerkinterpreterkomponente;

**[0023]** Fig. 5 ein Diagramm zur Darstellung der objektorientierten Protokollhierarchie, die die Austauschbarkeit der Ausgabeinterpreterkomponente und der Ausgabetreiberkomponente ermöglicht, und;

**[0024]** Fig. 6 ein Funktionsblockdiagramm einer erfindungsgemäßen Client-Server-Beziehung, die auf das in Fig. 1 gezeigte medizinische Bebilderungssystem anwendbar ist;

**[0025]** Die vorliegende Erfindung betrifft eine skalierbare Softwarearchitektur zur simultanen Übersetzung mehrerer medizinischer Bebilderungsprotokolle innerhalb eines Netzwerkparadigmas. Fig. 1 zeigt ein Funktionsblockdiagramm eines medizinischen Bebilderungssystems zur Übermittlung von Bildinformationen zwischen Mehrprotokoll-Bebilderungsvorrichtungen in einer Netzwerkkommunikationsumgebung gemäß der vorliegenden Erfindung. Das System **10** umfasst eine Vielzahl von Eingabe-Bebilderungsvorrichtungen in Form von Netzwerk-Clients **12**, eine oder mehrere Netzwerkausführungskomponenten **14**, eine oder mehrere Ausgabeschnittstellenkomponenten **16**, eine Ausgabe-Bebilderungsvorrichtung **18** und eine Schnittstellenausführungskomponente **20**. Jede Ausgabeschnittstellenkomponente **16** umfasst eine Ausgabeinterpreterkomponente **22** und eine Ausgabetreiberkomponente **24**.

**[0026]** Wie in Fig. 1 gezeigt, kommuniziert jeder Client **12** mit einer Ausgabe-Bebilderungsvorrichtung **18** über eine spezielle Leitung **26** gemäß einem bestimmten Protokoll. Sofern jeder Client **12** dasselbe Protokoll verwendet, wird also nur eine Leitung **26** benötigt, um die Kommunikation zwischen den Clients **12** und der Ausgabe-Bebilderungsvorrichtung **18** zu ermöglichen. Wenn jeder Client **12** ein oder zwei Protokolle von zwei verschiedenen Protokollen verwendet, werden zwei verschiedene Leitungen benötigt usw. Auf diese Weise ermöglicht die vorliegende Erfindung N unterschiedliche Leitungen für N unterschiedliche Protokolle, wobei jede Leitung in der Lage ist, M unterschiedliche Clients zu handhaben, die dieses jeweilige Protokoll verwenden. Daher ist eine separate Leitung für jeden Client nicht erforderlich, sondern nur für jedes unterschiedliche Protokoll.

**[0027]** Jede Leitung **26** umfasst drei primäre Komponenten: eine Netzwerkausführungskomponente **14**, eine Ausgabeinterpreterkomponente **22** und eine Ausgabetreiberkomponente **24**, wobei die beiden letztgenannten als eine einzelne Ausgabeschnittstellenkomponente **16** zusammengefasst sind. Allgemein gesagt ist das in Fig. 1 gezeigte System folgendermaßen aufgebaut. Für jede Ausgabe-Bebilderungsvorrichtung **18** erstellt die Netzwerkausführungskomponente **14** eine separate Leitung **26** für jedes separate Protokoll, das von mindestens einem Netzwerk-Client **12** verwendet wird, der mit der Ausgabe-Bebilderungsvorrichtung **18** ggf. kommuniziert. Die Netzwerkausführungskomponente **14** erreicht dies, indem sie eine Netzwerkausführungskomponente **14** speziell für das Protokoll instanziiert, das von einem oder mehreren Netzwerk-Clients **12** verwendet wird, sowie eine Ausgabeschnittstellenkomponente **16**, die speziell der Ausgabe-Bebilderungsvorrichtung **18** zugeordnet ist, zwei spezielle Netzwerkausführungskomponenten **14** und **16**, wodurch eine spezielle Leitung **26** entsteht. Die Erstellung der Leitungen **26** kann entweder „spontan" erfolgen, wenn Clients, die unterschied-

liche Protokolle verwenden, in das Netzwerk des Systems **10** eintreten oder dieses verlassen, oder sie kann erfolgen, wenn das Netzwerk erstmals instanziiert wird. Die vorliegende Erfindung ist in beiden Fällen nicht eingeschränkt.

**[0028]** Bei Erstellung der Leitungen **26** kommuniziert ein Client **12** mit der Ausgabe-Bebilderungsvorrichtung **18** allgemein auf folgende Weise. Die Netzwerkausführungskomponente **14** filtert und interpretiert die von einem Client **12** erhaltenen Anforderungen gemäß ersten Anforderungen, die die Ausgabeschnittstellenkomponente **16** versteht. Bei Übertragung an die Ausgabeschnittstellenkomponente **16** werden die ersten Anforderungen weiter gefiltert und in die entsprechenden zweiten Anforderungen interpretiert, die die Ausgabe-Bebilderungsvorrichtung **18** versteht. Auf diese Weise nimmt die vorliegende Erfindung Anforderungen entgegen, die für ein bestimmtes Protokoll bestimmt sind, übersetzt diese in erste Anforderungen und übersetzt diese dann weiter in zweite Anforderungen für eine bestimmte Bebilderungsvorrichtung. Somit können die Komponente **14** und die Komponente **16** unabhängig voneinander ausgetauscht werden, weil beide miteinander über erste Anforderungen kommunizieren. Anders ausgedrückt, ist die Implementierung einer Netzwerkausführungskomponente **14** für ein bestimmtes Protokoll unabhängig von einer Ausgabe-Bebilderungsvorrichtung **18**, während die Implementierung der Ausgabeschnittstellenkomponente **16** von einem gegebenen Protokoll unabhängig ist, das von einem bestimmten Client **12** verwendet wird. Es sei darauf hingewiesen, dass der beschriebene Vorgang auch in umgekehrter Richtung erfolgen kann, so dass Anforderungen von der Ausgabe-Bebilderungsvorrichtung **18** an den Client **12** gesendet werden können.

**[0029]** Die vorliegende Erfindung sieht somit ein Leitungsmodell vor, um die Kommunikation zwischen M Clients mit einer Bebilderungsvorrichtung zu ermöglichen, die N Protokolle verwenden. Die Schnittstellenausführungskomponente verwaltet die Erstellung dieser Leitungen. Eine Leitung wird für jedes spezielle Protokoll erstellt, das von mindestens einem von M Clients im Netzwerk verwendet wird. Da typischerweise N << M ist, schont die vorliegende Erfindung Ressourcen in einem System, in dem eine separate Leitung für jeden Client, jedoch kein separates Protokoll notwendig ist. Dies stellt einen wesentlichen Vorteil der vorliegenden Erfindung dar.

**[0030]** Fig. 2 zeigt ein Funktionsblockdiagramm eines weiteren Ausführungsbeispiels der vorliegenden Erfindung. Elemente aus Fig. 2 mit gleichen Bezugsziffern wie in Fig. 1 weisen darauf hin, dass die Elemente identisch sind, und dass die Beschreibung in Verbindung mit Fig. 1 gleichermaßen auf Fig. 2 anwendbar ist. Alternativ zur Instanziierung von N vollständigen Übersetzungsleitungen kann die Schnittstellenausführungskomponente so konfiguriert werden, dass sie eine Übersetzungsleitung mit N Netzwerkausführungskomponenten instanziiert, die unabhängig oder parallel arbeiten. Auf diese Weise können N × M Clients unterstützt werden, ohne N – 1 Ausgabeinterpreterkomponenten und N – 1 Ausgabetreiberkomponenten ineffizient bereitstellen zu müssen. Das System **58** aus Fig. 2 unterstützt N Netzwerkprotokolle und N × M Netzwerk-Clients mit der Implementierung nur einer Kommunikationsleitung. System **58** umfasst eine Vielzahl von Netzwerkausführungskomponenten **14**, die auf Netzwerk-Clients **12** achten, die bestimmte Netzwerkprotokolle verwenden. Die Schnittstellenausführungskomponente **20** verbindet jede Netzwerkausführungskomponente **14** kommunikativ mit einer einzelnen Ausgabeinterpreterkomponente **22**, einer einzelnen Ausgabetreiberkomponente **24** und einer einzelnen Ausgabe-Bebilderungsvorrichtung **18**, um eine einzelne Kommunikationsleitung mit mehreren, protokollspezifischen Netzwerkeingaben bereitzustellen.

**[0031]** Das in Fig. 2 gezeigte Ausführungsbeispiel der vorliegenden Erfindung unterscheidet sich von dem in Fig. 1 gezeigten insofern, als dass das erste Ausführungsbeispiel noch mehr Ressourcen schont als das letztere. Für den Fall, dass eine Ausgabe-Bebilderungsvorrichtung, jedoch mehrere Netzwerkprotokolle vorhanden sind, vergeudet das in Fig. 1 gezeigte Ausführungsbeispiel einige Ressourcen, indem redundante Ausgabeschnittstellenkomponenten **16** für jede Leitung **26** bereitgestellt werden, welche alle aufgrund der Tatsache redundant sind, dass nur eine Ausgabe-Bebilderungsvorrichtung vorhanden ist. Diese Redundanz und die entsprechende Vergeudung von Ressourcen wird durch das in Fig. 2 gezeigte Ausführungsbeispiel beseitigt. Fig. 2 zeigt nur eine Leitung **26** und nur eine Ausgabeschnittstellenkomponente **16**, mit der jede Netzwerkausführungskomponente **14** verbunden ist. Abgesehen von dieser geringeren Redundanz arbeitet das in Fig. 2 gezeigte Ausführungsbeispiel gleich wie das in Fig. 1 gezeigte, wobei die Beschreibung für Fig. 1 auch auf die Beschreibung in Bezug auf Fig. 2 angewendet werden sollte.

**[0032]** Wie in Fig. 1 gezeigt, sind die Netzwerkausführungskomponenten **14**, die Ausgabeschnittstellenkomponenten **16** und die Schnittstellenausführungskomponente **20** in einem Ausführungsbeispiel als ein objektorientiertes Softwaresystem implementiert, das Schnittstellen zu Netzwerken mit Netzwerk-Clients **12** und Ausgabe-Bebilderungsvorrichtungen **18** bildet. Das Softwaresystem kann als Teil einer Ausgabe-Bebilderungsvorrichtung **18** implementiert werden, beispielsweise als ein digitaler Halbton-Laserabbildungsgerät, oder es kann

als Teil einer diskreten Schnittstellenvorrichtung implementiert werden, die die Kommunikation von Bildinformationen zwischen den Netzwerk-Clients **12** und der Ausgabe-Bebilderungsvorrichtung **18** steuert.

**[0033]** In einem Ausführungsbeispiel der Erfindung umfasst das Netzwerk eine Vielzahl verschiedener Clients, wie beispielsweise Magnetresonanz- (MR), Computertomographie- (CT), herkömmliche Radiographie- (Röntgen) oder Ultraschallvorrichtungen, die von einer Reihe verschiedener Hersteller hergestellt werden, beispielsweise von Siemens, Toshiba, GE oder Picker. Das Laserabbildungsgerät kann ein beliebiges Abbildungsgerät sein, wie beispielsweise einer der von Imation hergestellten, der die Protokolle 831, 952 oder SuperSet beherrscht. Das Laserabbildungsgerät kann direkt im Netzwerk angeordnet sein, wobei in diesem Fall das Softwaresystem üblicherweise auf einer Hardwarekarte angeordnet ist, die in das Laserabbildungsgerät gesteckt wird. Die Karte umfasst üblicherweise eine Eingabe-Ausgabe-Schaltung (IO) sowie einen Speicher, wie einen ROM oder Flash-ROM, bei dem es sich um einen umprogrammierbaren ROM handelt. Das Softwaresystem befindet sich in diesem Speicher.

**[0034]** In dem alternativen Ausführungsbeispiel befindet sich das Laserabbildungsgerät nicht direkt im Netzwerk, sondern ist statt dessen mit dem Netzwerk über einen Zwischencomputer verbunden, der sich selbst direkt im Netzwerk befindet. Der Zwischencomputer ist üblicherweise mit einem Schreib-/Lesespeicher (RAM), einem Lesespeicher (ROM), einer zentralen Verarbeitungseinheit (CPU) und einer Speichervorrichtung bestückt, beispielsweise einem Festplattenlaufwerk, einem programmierbaren ROM oder einem Plattenlaufwerk. In diesem Fall befindet sich das Softwaresystem auf der Speichervorrichtung des Zwischencomputers und wird in den RAM kopiert und von dort seitens der CPU ausgeführt. Wenn es sich bei der Speichervorrichtung um ein Plattenlaufwerk oder um eine andere auswechselbare Speichervorrichtung handelt, kann das Softwaresystem auf dem Speichermedium zur Einführung in die Vorrichtung gespeichert werden. Die vorliegende Erfindung ist allerdings nicht auf eine bestimmte Hardwareimplementierung beschränkt.

**[0035]** Die von den Eingabe-Bebilderungsvorrichtungen erzeugten und den Netzwerk-Clients **12** zugeordneten Bildinformationen umfassen sowohl Anforderungen nach Bebilderungsoperationen als auch Bilddaten, die digitale Bildwerte enthalten, die ein von der Ausgabe-Bebilderungsvorrichtung **18** zu handhabendes Bild darstellen. Die Leitung **26** wird hier so beschrieben, dass sie die Übermittlung von Bildinformationen in Form von Bebilderungsanforderungen handhabt, wobei Bildinformationen in Form digitaler Bildwerte das Bild darstellen, das von einem separaten Kommunikationsweg übermittelt wird. Innerhalb des Geltungsbereichs der vorliegenden Erfindung könnte die Leitung **26** jedoch auch so konfiguriert werden, dass sie die Kommunikation der Bildinformationen in Form von Anforderungen nach Bebilderungsoperationen und Bilddaten handhabt, die die digitalen Bildwerte enthalten.

**[0036]** In einem typischen medizinischen Bebilderungssystem umfassen Bebilderungsanforderungen Anforderungen zur Veranlassung eines Bilddruckauftrags seitens der Ausgabe-Bebilderungsvorrichtung **18**, Anforderungen zum Abbrechen eines zuvor veranlassten Bilddruckauftrags, Anforderungen zur Definition oder Modifikation eines Formats eines zu druckenden Bildes, Anforderungen zum Löschen eines Satzes von Bilddaten, die ein zuvor erfasstes Bild darstellen, sowie Anforderungen zum Speichern von Bilddaten an einer bestimmten Bildposition.

Komponenten der Erfindung: Schnittstellenausführungskomponente

**[0037]** Die Schnittstellenausführungskomponente **20** bildet eine oder mehrere (1 bis N) Kommunikationsleitungen. Jede Kommunikationsleitung **26** verbindet kommunikativ einen oder mehrere von M Netzwerk-Clients **12**, eine der Netzwerkausführungskomponenten **14**, eine der Ausgabeinterpreterkomponenten **22**, eine der Ausgabetreiberkomponenten **24** und eine Ausgabe-Bebilderungsvorrichtung **18** in bidirektionaler Weise. Die Ausgabe-Bebilderungsvorrichtung **18** kann mit jeder Leitung **26** auf gemeinsamer Basis kommunizieren. Alternativ hierzu könnte eine Vielzahl von Ausgabe-Bebilderungsvorrichtungen **18** bereitgestellt werden, von denen jede kommunikativ mit einer bestimmten Leitung **26** verbunden ist.

**[0038]** Die Schnittstellenausführungskomponente **20** stellt die höchste Intelligenzebene innerhalb des Systems **10** aus Fig. 1 dar. Sie lenkt und verwaltet die jeweiligen Netzwerkkomponenten **26** und Ausgabeschnittstellenkomponenten **16**, die für die Netzwerk-Clients **12** zur Kommunikation mit der Ausgabe-Bebilderungsvorrichtung **18** benötigt werden. Wie in Fig. 1 gezeigt, instanziiert die Schnittstellenausführungskomponente **20** auf Basis von N verschiedenen Protokollen, die die Clients **12** beherrschen, eine bestimmte Leitung **26**, die aus einer Netzwerkausführungskomponente **14** und der Ausgabeschnittstellenkomponente **16** besteht. Wenn P unterschiedliche Ausgabe-Bebilderungsvorrichtungen vorhanden sind (im Unterschied zu einer, wie in Fig. 1 gezeigt), instanziiert die Schnittstellenausführungskomponente N × P unterschiedliche Leitungen, und zwar

eine für jedes eindeutige Paar aus Bebilderungsvorrichtung und Protokoll. Dies kann in einem separaten Einrichtungsbetrieb oder "spontan" erfolgen, während Clients, die unterschiedliche Protokolle beherrschen, in das Netzwerk eintreten oder dieses verlassen.

**[0039]** Zwar verfügt die Schnittstellenausführungskomponente **20** über die größte Intelligenz aller Komponenten innerhalb der vorliegenden Erfindung, aber sie unterscheidet sich von der in US-A-5,630,101 offengelegten und beschriebenen Schnittstellenausführungskomponente darin, dass sie eine geringere Intelligenz aufweist als die in dieser Patentanmeldung beschriebene Schnittstellenausführungskomponente. Die in US-A-5,630,101 offengelegte und beschriebene Schnittstellenausführungskomponente instanziiert eine Eingabeschnittstellenkomponente speziell für jeden Client, der mit einer bestimmten Bebilderungsvorrichtung kommunizieren muss. Die Schnittstellenausführungskomponente konstruiert eine Leitung auf Client-Client-Basis. Im Unterschied dazu instanziiert die vorliegende Erfindung eine Netzwerkausführungskomponente **14** für ein bestimmtes Protokoll und delegiert damit die Zuständigkeit für die Bedienung der Netzwerk-Clients. Die erfindungsgemäße Schnittstellenausführungskomponente konstruiert somit eine Leitung auf Protokoll-Protokoll-Basis. Sie verfügt insofern über weniger Intelligenz, als dass sie die Kommunikation mit bestimmten Clients nicht verwalten muss, wie dies bei der Schnittstellenausführungskomponente nach US-A-5,630,101 der Fall ist. Die letztere Schnittstellenausführungskomponente „kennt" somit alle Details über die Eingabevorrichtung, während die erfindungsgemäße Schnittstellenausführungskomponente nur „weiß", dass sich im Netz Eingabevorrichtungen befinden, während sie die Zuständigkeit für die Handhabung der Implementierung der Schnittstelle bezüglich der Eingabevorrichtungen an die Netzwerkausführungskomponente **14** delegiert.

**[0040]** Die Schnittstellenausführungskomponente **20** definiert die Struktur der Leitung **26**. Die Leitung **26** ist derart konfiguriert, dass sie eine Reihe von Komponenten **30**, **32'** (die in Fig. 3 gezeigt werden und, wie dort gezeigt und später erläutert, von der Netzwerkausführungskomponente **14** instanziiert werden), **22** und **24** mit unterschiedlichen Protokollen wahlweise miteinander verbindet, was ein wesentliches Maß an Flexibilität bereitstellt. Diese Flexibilität ermöglicht ein medizinisches Bebilderungssystem **10**, das in der Lage ist, die Kommunikation zwischen einer Vielzahl verschiedener Netzwerk-Clients **12** und einer oder mehreren Ausgabe-Bebilderungsvorrichtungen **18** mit einer Vielzahl unterschiedlicher Funktionsmöglichkeiten herzustellen. Die Schnittstellenausführungskomponente **20** behandelt jede Funktionalität unabhängig von Komponente **14**, **22** und **24** als eine "Blackbox" mit einer eindeutig definierten Menge von Zuständigkeiten und einer definierten Schnittstelle. Die Schnittstellenausführungskomponente **20** wählt die entsprechende Reihe von Blackboxes je nach Umgebung aus und verbindet diese mit „Griffen" untereinander, um eine vollständige Leitung **26** zu bilden. Als ein weiterer Vorteil ist die Schnittstellenausführungskomponente **20** in einem Ausführungsbeispiel derart konfiguriert, dass sie die Komponenten dynamisch „spontan" verbinden kann, um eine Kommunikationsleitung **26** zu bilden, die für die aktuelle Bebilderungsumgebung geeignet ist. Die Schnittstellenausführungskomponente **20** ist zudem derart konfiguriert, dass sie eine skalierbare Softwarearchitektur mit einer Vielzahl von Kommunikationsleitungen **26** erzeugt, die nach unterschiedlichen Protokollen konfiguriert sind. Die skalierbare Architektur ermöglicht es der Ausgabe-Bebilderungsvorrichtung **18**, simultan mit mehreren Netzwerk-Clients **12** auf gemeinsamer Basis unter Verwendung der nötigen Netzwerkprotokolle, wie von jeder Leitung **26** bereitgestellt, zu kommunizieren. Alternativ hierzu könnte eine Vielzahl von Ausgabe-Bebilderungsvorrichtungen **18** bereitgestellt werden, von denen jede kommunikativ mit einer bestimmten Leitung **26** verbunden ist.

**[0041]** Die Schnittstellenausführungskomponente **20** skaliert die Softwarearchitektur somit derart, dass sie die Anforderungen an die Umgebung erfüllt, wobei so viele Netzwerkausführungskomponenten und Leitungen erstellt werden, wie es unterschiedliche Netzwerkprotokolle gibt. Die Schnittstellenausführungskomponente **20** verbindet wahlweise eine Reihe von Komponenten **14**, **22** und **24**, die bestimmte Protokolle aufweisen, die notwendig sind, um zu einem bestimmten Netzwerk-Client **12**, einer bestimmten Ausgabe-Bebilderungsvorrichtung **18** und den erforderlichen Hardwareschnittstellen zu passen.

Komponenten der Erfindung: Netzwerkausführungskomponenten

**[0042]** Die Netzwerkausführungskomponente **14** ist für die Handhabung aller Netzwerk-Clients **12** zuständig, die über ein bestimmtes Protokoll miteinander kommunizieren. Wie in Fig. 1 gezeigt, wird eine Netzwerkausführungskomponente **14** für jedes bestimmte von N Netzwerkprotokollen bereitgestellt. Die Netzwerkausführungskomponente **14** verwaltet somit mehrere Netzwerk-Clients **12** gleichzeitig. Die Schnittstellenausführungskomponente **20** delegiert die Zuständigkeit für die Verwaltung aller netzwerkspezifischen Dienste an die Netzwerkausführungskomponente **14**. Die Schnittstellenausführungskomponente **20** instanziiert eine bestimmte Netzwerkausführungskomponente **14** für jedes medizinische Bebilderungsnetzwerkprotokoll, das im Netzwerk von dem System **10** unterstützt wird. Wenn beispielsweise das Picker-Netzwerkprotokoll unterstützt wird, instanziiert die Schnittstellenausführungskomponente **20** eine Netzwerkausführungskomponente **14**, die

ein derartiges Protokoll bedienen kann. Beispielsweise instanziiert die Schnittstellenausführungskomponente **20** eine weitere Netzwerkausführungskomponente, die in der Lage ist, das DICOM-Protokoll zu bedienen, sofern dieses Protokoll unterstützt werden muss.

**[0043]** Die Netzwerkausführungskomponente **14** lenkt alle Objekte, die zur Verwaltung der Netzwerkkommunikation erforderlich sind. Die primäre Funktion der Netzwerkausführungskomponente **14** besteht darin, eine Netzwerkschnittstelle **28** zu überwachen oder auf Bebilderungsanforderungen von Netzwerk-Clients **12**, die ein bestimmtes Protokoll verwenden, „abzuhören". Wenn ein Netzwerk-Client **12** den Zugang zu einer Ausgabe-Bebilderungsvorrichtung **18** über ein bestimmtes Netzwerkprotokoll anfordert, erstellt die Netzwerkausführungskomponente **14** eine Netzwerktreiberkomponente **30** und eine Netzwerkinterpreterkomponente **32**, die für dieses Protokoll geeignet sind, wie in Fig. 3 gezeigt. Die Netzwerkausführungskomponente **14** bindet die Netzwerktreiberkomponente **30** an die Netzwerkinterpreterkomponente **32** und dann die Netzwerkinterpreterkomponente **32** an die Ausgabeinterpreterkomponente **22** unter Verwendung von Informationen, die zuvor von der Schnittstellenausführungskomponente **20** bereitgestellt wurden. Die Netzwerkausführungskomponente **14** hört dann die Netzwerkschnittstelle **28** auf neue Anforderungen ab, die gemäß dem bestimmten Netzwerkprotokoll gesendet werden. Die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** bilden zusammen eine Netzwerkschnittstellenkomponente **33**, wie ebenfalls in Fig. 3 gezeigt.

**[0044]** Das Vorhandensein der Netzwerkausführungskomponente in der vorliegenden Erfindung dient als Unterscheidungsmerkmal der Erfindung gegenüber US-A-5,630,101. In US-A-5,630,101 gibt es keine entsprechenden Netzwerkausführungskomponenten, sondern Eingabeschnittstellenkomponenten. die Eingabeschnittstellenkomponente ist allerdings keine intelligente Komponente wie die erfindungsgemäße Netzwerkausführungskomponente. Stattdessen wird die Eingabeschnittstellenkomponente von der Schnittstellenausführungskomponente für jede Verbindung zwischen einem bestimmten Client und der Bebilderungsvorrichtung instanziiert. Im Unterschied dazu delegiert die Schnittstellenausführungskomponente in der vorliegenden Erfindung die Zuständigkeit für die Client-Kommunikation an eine Netzwerkausführungskomponente, die ihrerseits weitere Komponenten instanziiert, wie für eine oder mehrere Clients erforderlich, die ein gemeinsames Protokoll beherrschen, um mit der Bebilderungsvorrichtung kommunizieren zu können.

**[0045]** Die Netzwerkausführungskomponenten verleihen der vorliegenden Erfindung somit den Vorteil der Netzwerkkommunikation unter minimaler Nutzung der Ressourcen. Beispielsweise bewirkt die Anwendung des in US-A-5,630,101 beschriebenen Systems auf ein Netzwerk von Clients die Erstellung von Leitungen für jeden dieser Clients. Durch Einbringung der Client-Kommunikation in eine intelligente Netzwerkausführungskomponente **14** entfällt für die vorliegende Erfindung die Notwendigkeit, Leitungen für jeden Client erstellen zu müssen, so dass nur die Erstellung einer Leitung für jedes der Protokolle angefordert zu werden braucht, über die die Clients kommunizieren können. Weil die Zahl der Kommunikationsprotokolle üblicherweise wesentlich kleiner als die Zahl der Clients ist, führt dies zu einer deutlichen Einsparung bei der Ressourcennutzung. Indem die Zuständigkeit für die Client-Kommunikation an die Netzwerkausführungskomponente **14** übergeben wird, wird die Schnittstellenausführungskomponente **20** von derartigen Verwaltungsaufgaben befreit, die ansonsten die Schnittstellenausführungskomponente übermäßig belasten könnten.

**[0046]** In einem Ausführungsbeispiel werden die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32**, wie in Fig. 3 gezeigt, "spontan" dann instanziiert, wenn die Netzwerkausführungskomponente **14** eines Netzwerk-Clients **12** ein bestimmtes Protokoll an der Netzwerkschnittstelle **84** erkennt, wodurch die zur Unterstützung dieser Komponenten erforderlichen Hardware- und Softwareressourcen reserviert werden, bis diese benötigt werden. Diese dynamische Instanziierung der Netzwerktreiberkomponente **30** und der Netzwerkinterpreterkomponente **32** ermöglicht eine Reduzierung des Systemoverheads, der ansonsten notwendig wäre. Wenn eine Reservierung der Ressourcen nicht kritisch ist, werden diese Komponenten alternativ dauerhaft bereitgestellt, um für jedes Protokoll eine feste, dedizierte Leitung **26** bereitzustellen.

**[0047]** Sobald die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** erstellt worden sind, delegiert die Netzwerkausführungskomponente **14** die gesamte Zuständigkeit für die Bedienung des jeweiligen Netzwerk-Clients **12** an das Paar aus Treiber und Interpreter. Die Netzwerkausführungskomponente **14** bindet den Netzwerk-Client **12**, die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** kommunikativ an eine der Ausgabeinterpreterkomponenten **22**, wobei Verbindungsinformationen genutzt werden, die zuvor von der Netzwerkausführungskomponente **14** über die Schnittstellenausführungskomponente **20** bereitgestellt wurden.

**[0048]** Jede Netzwerktreiberkomponente **30** ist derart konfiguriert, dass sie Bildinformationen von einem Netzwerk-Client **12** gemäß einer Vielzahl verschiedener Netzwerkschnittstellenprotokolle empfängt. Jedes

Netzwerkschnittstellenprotokoll ist speziell einem der Netzwerk-Clients **12** zugeordnet und gibt die modalitätsspezifischen Anforderungen zur Kommunikation mit dem jeweiligen Netzwerk-Client wieder. Jede der Netzwerkinterpreterkomponenten **30** ist derart konfiguriert, dass sie erste Bebilderungsanforderungen gemäß einem der Netzwerkschnittstellenprotokolle, basierend auf den empfangenen Bildinformationen, erzeugt. Die ersten Bebilderungsanforderungen werden von der Netzwerkinterpreterkomponente **32** erzeugt und entsprechen den von dem Netzwerk-Client **12** erzeugten Bebilderungsanforderungen. Die ersten Bebilderungsanforderungen werden an die Ausgabeschnittstellenkomponente **16** übermittelt.

**[0049]** Jedes der Netzwerkschnittstellenprotokolle umfasst sowohl ein Netzwerktreiberprotokoll, das auf Netzwerktreiberkomponenten **30** anwendbar ist, als auch ein Netzwerkinterpreterprotokoll, das auf Netzwerkinterpreterkomponenten **32** anwendbar ist. Die entsprechenden Netzwerktreiberprotokolle werden von den Kommunikationsanforderungen eines bestimmten Netzwerk-Clients **12** ermittelt, während die geeigneten Netzwerkinterpreterprotokolle von dem Bildinformationsformat einer bestimmten Eingabe-Bebilderungsvorrichtung ermittelt werden, die dem Netzwerk-Client zugeordnet ist. Das Bildinformationsformat bezieht sich auf die Art der Bebilderungsanforderungen, die gemäß dem Protokoll einer bestimmten Eingabe-Bebilderungsvorrichtung erzeugt werden. Das Netzwerktreiberprotokoll spezifiziert die Weise, in der eine Netzwerktreiberkomponente **30** die Übertragung von Bildinformationen von einer Eingabe-Bebilderungsvorrichtung durchführen sollte, die einem Netzwerk-Client **12** zugeordnet ist. Das Netzwerkinterpreterprotokoll spezifiziert die Weise, in der die Netzwerkinterpreterkomponente **32** die Bildinformationen interpretieren sollte, um die ersten Bebilderungsanforderungen zu erzeugen. Die Netzwerktreiber- und Netzwerkinterpreterprotokolle können sich je nach Art des Netzwerk-Clients **12** und des Herstellers der Ausgabe-Bebilderungsvorrichtung **18** erheblich voneinander unterscheiden.

**[0050]** Die Netzwerkinterpreterkomponente **32** nutzt zudem einen gemeinsamen Satz von Aufgaben mit anderen Netzwerkinterpreterkomponenten, ungeachtet eines bestimmten Netzwerkinterpreterprotokolls. Nach Erhalt der Bildinformationen von einer Netzwerktreiberkomponente **30** analysiert eine Netzwerkinterpreterkomponente **32** Anforderungen, die in den Bildinformationen enthalten sind, und übersetzt diese, um erste Bebilderungsanforderungen zu erzeugen, die den von der Ausgabe-Bebilderungsvorrichtung **18** bereitgestellten Operationen entsprechen. Die ersten Bebilderungsanforderungen umfassen Anforderungen nach einer Reihe von gemeinsamen Bebilderungsdiensten, die von der Ausgabe-Bebilderungsvorrichtung **18** bereitgestellt werden.

**[0051]** Die Weise, in der die Netzwerkinterpreterkomponente **32** die Anforderungen interpretiert, die von dem Netzwerk-Client **12** erzeugt werden, kann sich je nach Netzwerkinterpreterprotokoll ändern. Die Netzwerkinterpreterkomponente **32** versteht das genaue Format, die Anweisungen und die Timing-Einschränkungen, die den Bildinformationen inhärent sind, die von einem bestimmten Netzwerk-Client **12** erzeugt wurden. Dennoch stellen alle Ausgabeinterpreterkomponenten **22** eine gemeinsame Grundfunktion zur Erzeugung erster Bebilderungsanforderungen zur Verfügung. Die Netzwerkinterpreterkomponente **32** sendet die ersten Bebilderungsanforderungen über die Leitung **26**. Sobald die ersten Bebilderungsanforderungen von nachgeordneten Komponenten in der bidirektionalen Leitung **26** verarbeitet worden sind und eine Antwort erhalten worden ist, erzeugt die Netzwerkinterpreterkomponente **32** eine entsprechende Antwort für den vernetztes System **13**. Die Netzwerkinterpreterkomponente **32** sendet die Antwort an den Netzwerk-Client **12** über die Leitung **26** und die Netzwerktreiberkomponente **30**, die die Kommunikationsanforderungen handhabt, die zur Übermittlung der Antwort an die Eingabe-Bebilderungsvorrichtung erforderlich sind.

**[0052]** Die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** wurden unter Berücksichtigung der Tatsache beschrieben, dass die Netzwerkschnittstellenkomponente **33** alternativ als ein einzelnes, integriertes Softwaremodul implementiert werden könnte. In dem beschriebenen Ausführungsbeispiel wird eine Netzwerkschnittstellenkomponente **33** von einer diskreten Netzwerktreiberkomponente **30** und einer diskreten Netzwerkinterpreterkomponente **32** realisiert. Eine diskrete Implementierung der Unterkomponenten teilt die Funktionalität jeder Netzwerkschnittstellenkomponente **33** zur besseren Modularität in kleinere Pakete auf. Beispielsweise bedürfen Änderungen der Hardwarespezifikationen für die Netzwerkschnittstelle **28**, die auf eine erweiterte Modularität zurückzuführen sind, nur einer Rekonfiguration der Netzwerktreiberkomponente **30**, statt der gesamten Netzwerkschnittstellenkomponente **33**.

**[0053]** Ungeachtet der protokollspezifischen Funktionen sind die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** des gleichen Typs (d.h. alle Netzwerktreiberkomponenten) so konfiguriert, dass sie mehrere gemeinsame Aufgaben wahrnehmen. Beispielsweise nutzen die Netzwerktreiberkomponenten **30** gemeinsame Aufgaben, die zur Kommunikation mit einem Netzwerk-Client **12** notwendig sind, der nach einem bestimmten Netzwerkprotokoll arbeitet. Eine Netzwerktreiberkomponente **30** ist derart konfiguriert, dass

sie alle hardwarespezifischen Faktoren handhabt, also beispielsweise Unterbrechungen, Puffer und Quittungsvorgänge, die notwendig sind, um Bebilderungsinformationen an einen bestimmten Netzwerk-Client **12** zu übermitteln oder von diesem zu empfangen. Eine Netzwerktreiberkomponente **30** ist zudem so konfiguriert, dass sie alle anderen spezifischen Notwendigkeiten eines Netzwerk-Clients **12** handhabt, wie beispielsweise die Paketisierung oder Initialisierung. Die Netzwerktreiberkomponente **30** führt alle notwendigen Kommunikationsaufgaben durch, und isoliert damit die verbleibende Leitung **26** von Kenntnissen über bestimmte Anforderungen zur Kommunikation mit dem Netzwerk-Client **12**. Die Netzwerktreiberkomponente **30** übernimmt somit eine zweifache Zuständigkeit. Erstens empfängt die Netzwerktreiberkomponente **30** Bildinformationen abseits des Netzwerks vom Netzwerk-Client **12** und bereitet diese Bildinformationen für die nächste Stufe der Leitung **26** auf, d.h. für die Netzwerkinterpreterkomponente **32**. Zweitens übermittelt die Netzwerktreiberkomponente **30** Antworten, die auf der bidirektionalen Leitung **26** auftreten, an das Netzwerk zur Kommunikation mit dem Netzwerk-Client **12**.

Komponenten der Erfindung: Ausgabeschnittstellenkomponenten

**[0054]** Wie in Fig. 1 gezeigt, ist jede Ausgabeschnittstellenkomponente **16** derart konfiguriert, dass sie zweite Bebilderungsanforderungen gemäß einer Vielzahl verschiedener Ausgabeprotokolle über eine Ausgabeinterpreterkomponente **22** erzeugt, und zwar je nach Inhalt der ersten Bebilderungsanforderung. Die zweiten Bebilderungsanforderungen stellen den Inhalt der ersten Bebilderungsanforderungen dar, wie von der Ausgabeinterpreterkomponente **22** zur Übermittlung an die Ausgabe-Bebilderungsvorrichtung **18** übersetzt. Jedes Ausgabeschnittstellenprotokoll ist speziell dem Typ der Ausgabe-Bebilderungsvorrichtung **18** zugeordnet und gibt ebenso wie das Netzwerkschnittstellenprotokoll die Anforderungen an die Kommunikation mit der jeweiligen Ausgabe-Bebilderungsvorrichtung wieder. Außerdem ist jede Ausgabeschnittstellenkomponente **16** so konfiguriert, dass sie die zweiten Bebilderungsanforderungen an die Ausgabe-Bebilderungsvorrichtung **18** über die Ausgabetreiberkomponente **24** gemäß einem der Ausgabeschnittstellenprotokolle übermittelt.

**[0055]** Jedes der Ausgabeschnittstellenprotokolle umfasst ein Ausgabeinterpreterprotokoll, das auf die Ausgabeinterpreterkomponenten **22** anwendbar ist, und ein Ausgabetreiberprotokoll, das auf die Ausgabetreiberkomponenten **24** anwendbar ist. Das Ausgabetreiberprotokoll wird durch die Kommunikationsanforderungen der Ausgabe-Bebilderungsvorrichtung **18** bestimmt, während das entsprechende Ausgabeinterpreterprotokoll durch das Bildinformationsformat der Ausgabe-Bebilderungsvorrichtung bestimmt wird. Das Ausgabeinterpreterprotokoll spezifiziert die Weise, in der die Ausgabeinterpreterkomponente **22** erste Bebilderungsanforderungen interpretieren sollte, um zweite Bebilderungsanforderungen in einer Form zu erzeugen, die von der Ausgabe-Bebilderungsvorrichtung **18** verstanden werden. Das Ausgabetreiberprotokoll spezifiziert die Weise, in der eine Ausgabetreiberkomponente **24** die Übermittlung der zweiten Bebilderungsanforderungen an die Ausgabe-Bebilderungsvorrichtung **18** durchführen sollte. Wie bei den Netzwerkschnittstellenprotokollen unterliegen die Ausgabeschnittstellenprotokolle Abweichungen. Beispielsweise kann sowohl das Ausgabetreiber- als auch das Ausgabeinterpreterprotokoll entsprechend dem Typ der Funktionsmöglichkeiten variieren, die von der Ausgabe-Bebilderungsvorrichtung **18** bereitgestellt werden, beispielsweise 831, 952 oder SuperSet im Falle des von Imation hergestellten Laserabbildungsgeräts.

**[0056]** Eine Ausgabeinterpreterkomponente **22** ist derart konfiguriert, dass sie über die Leitung **26** erste Bebilderungsanforderungen empfängt, die von einer Netzwerkinterpreterkomponente **32** erzeugt worden sind, und die ersten Bebilderungsanforderungen interpretiert, um zweite Bebilderungsanforderungen zu erzeugen, die dem von der Ausgabe-Bebilderungsvorrichtung **18** jeweils geforderten Protokoll entsprechen. Die zweiten Bebilderungsanforderungen entsprechen im Wesentlichen den ersten Bebilderungsanforderungen, sind aber entsprechend dem Ausgabeprotokoll konfiguriert, das von der Ausgabe-Bebilderungsvorrichtung **18** beherrscht wird. Somit dienen die zweiten Bebilderungsanforderungen als Anforderungen für dieselben Bebilderungsdienste, die von den ersten Bebilderungsanforderungen spezifiziert worden sind. Die Weise, in der die Ausgabeinterpreterkomponente **22** die Anweisungen interpretiert, kann je nach dem speziellen Ausgabeinterpreterprotokoll variieren, das von der Ausgabe-Bebilderungsvorrichtung **18** vorgegeben wird, aber alle Ausgabeinterpreterkomponenten **22** nutzen eine gemeinsame Aufgabe, um zweite Bebilderungsanforderungen in einem Protokoll zu erzeugen, das von der Ausgabe-Bebilderungsvorrichtung beherrscht wird. Die Ausgabeinterpreterkomponente **22** sendet die zweiten Bebilderungsanforderungen über die Leitung **26**. Wenn die Ausgabe-Bebilderungsvorrichtung **18** die zweiten Bebilderungsanforderungen verarbeitet und eine über die Leitung **26** empfangene Antwort formuliert, entfernt die Ausgabeinterpreterkomponente **22** ausgabeprotokollspezifische Informationen und erstellt eine entsprechende Antwort für die Netzwerkinterpreterkomponente **32**.

**[0057]** Mit Bezug auf die Ausgabetreiberkomponente **24**, führen alle Ausgabetreiberkomponenten **24**, ebenso wie die Netzwerktreiberkomponenten **30**, einen gemeinsamen Satz an Kommunikationsaufgaben durch. Eine

Ausgabetreiberkomponente **24** ist derart konfiguriert, dass sie alle hardwarespezifischen Faktoren handhabt, also beispielsweise Unterbrechungen, Puffer und Quittungsvorgänge, die notwendig sind, um Bebilderungsinformationen an eine bestimmte Ausgabe-Bebilderungsvorrichtung **18** zu übermitteln oder von diesem zu empfangen. Die Ausgabetreiberkomponente **24** isoliert die verbleibende Pipeline **26** von jeglicher Kenntnis, dass die Kommunikation mit der Ausgabe-Bebilderungsvorrichtung **18** über eine serielle Schnittstelle, eine parallele Schnittstelle oder ein Dual-Port-RAM usw. erfolgt. Die Ausgabetreiberkomponente **24** übermittelt zweite Bebilderungsanforderungen, die von der Ausgabeinterpreterkomponente **22** erzeugt wurden, an die Ausgabe-Bebilderungsvorrichtung **18**, wobei alle Kommunikationsanforderungen gewahrt bleiben. Die Ausgabetreiberkomponente **24** empfängt Antworten von der Ausgabe-Bebilderungsvorrichtung **18** und bereitet die Antwort zur Übertragung an die Ausgabeinterpreterkomponente **22** über die bidirektionale Leitung **26** vor.

**[0058]** Die Ausgabeinterpreterkomponente **22** und die Ausgabetreiberkomponente **24** wurden unter Berücksichtigung der Tatsache beschrieben, dass die Ausgabeschnittstellenkomponente **16** alternativ als ein einzelnes, integriertes Softwaremodul implementiert werden könnte. In dem beschriebenen Ausführungsbeispiel wird eine Ausgabeschnittstellenkomponente **16** allerdings von einer diskreten Ausgabeinterpreterkomponente **22** und einer diskreten Ausgabetreiberkomponente **24** realisiert. Eine diskrete Implementierung der Unterkomponenten teilt die Funktionalität jeder Ausgabeschnittstellenkomponente **16** zur besseren Modularität in kleinere Pakete auf. Beispielsweise bedürfen Änderungen der Hardwarespezifikationen für die Ausgabeschnittstellenkomponente **16**, die auf eine erweiterte Modularität zurückzuführen sind, nur einer Rekonfiguration der Ausgabetreiberkomponente **24** statt der gesamten Ausgabeschnittstellenkomponente **16**.

Objektorientierung der Komponenten

**[0059]** Um die Austauschbarkeit der Komponenten wie beschrieben zu ermöglichen, müssen die Softwareschnittstellen zwischen den Komponenten **30**, **32**, **22** und **24** vordefiniert werden, um jeden Komponententyp abzustimmen. Gleichzeitig muss eine individuelle Komponente **30**, **32**, **22** und **24** konfiguriert werden, um für ein bestimmtes Protokoll spezifische Funktionen zu implementieren. Die vorliegende Erfindung nutzt objektorientierte Techniken, insbesondere die der Weitervererbung, um ein generisches Basisklassenprotokoll für jeden Komponententyp zu entwickeln (z.B. Netzwerktreiberkomponente **30**).

**[0060]** Die Weitervererbung ist eine objektorientierte Technik, die als Mechanismus zur Erzeugung neuer Klassen aus vorhandenen Daten dient. Eine neue Klasse ist bis auf einen kleinen Unterschied ähnlich zu einer vorhandenen Klasse; die Weitervererbung dient dazu, die neue Klasse anhand der vorhandenen Klasse zu definieren. Die vorhandene Klasse, die als Quelle für die Weitervererbung dient, wird als Basisklasse bezeichnet, während die neue Klasse, die von der Basisklasse abgeleitet wird, als abgeleitete Klasse bezeichnet wird. Eine vorhandene Klasse kann als Basisklasse für mehrere abgeleitete Klassen dienen. Die Basisklasse ist eine Definition einer generischen Klasse von Softwareobjekten, während die Klassen, die von der Basisklasse abgeleitet sind, mehr spezifische oder spezialisierte Klassen der Objekte definieren. Das generische Basisklassenprotokoll spezifiziert die Funktionen, die von einer Komponente bereitgestellt werden, sowie die Prozeduren für den Zugang zu diesen Funktionen. Jede spezifische Protokollkomponente "erbt" von dem entsprechenden Basisklassenprotokoll und implementiert die Schnittstelle gemäß der Umgebung.

**[0061]** Klassenvererbung ermöglicht es, Mitglieder einer Klasse so zu benutzen, als ob sie Mitglieder einer zweiten Klasse seien. Es ist keine zusätzliche Programmierung erforderlich, um die Unterklasse zu implementieren, ausgenommen der Operationen, die entweder die von den anderen Klassen geerbten Mitglieder erweitern oder ersetzen. Während der Entwicklung dieses objektorientierten Systems werden Unterklassen aus bestehenden Klassen konstruiert, bis die entsprechende Funktionalität entwickelt ist. Die Konstruktion von Unterklassen fuhrt zur Bildung einer Klassenhierarchie. Die Klassenhierarchie ist in der Basisklasse begründet, die einen minimalen Verhaltenssatz umfasst, der allen Unterklassen gemeinsam ist.

**[0062]** Erfindungsgemäß ist jede Komponente **30**, **32**, **22** und **24** gemäß einem bestimmten Protokoll konfiguriert, dient aber auch als Unterklasse des Basisklassenprotokolls. Weil jede Komponente **30**, **32**, **22** und **24** von dem Basisklassenprotokoll erbt und einen minimalen Funktionssatz implementiert, so dass die Basisklassenanforderungen erfüllt werden, kann sie direkt gegen eine andere Komponente des gleichen Typs ausgetauscht werden, die von demselben Basisklassenprotokoll erbt. Die Austauschbarkeit, die sich aus den objektorientierten Techniken ergibt, erzeugt eine „direktverbundene" Softwarearchitektur, in der jede Komponente effektiv in die Leitung **26** eingefügt werden kann, ohne dass eine zusätzliche Schnittstellenentwicklung notwendig wäre.

**[0063]** Fig. 5 und Fig. 6 zeigen ein Beispiel einer objektorientierten Protokollhierarchie, die die Austauschbar-

keit der Komponenten **30**, **32**, **22** und **24** erleichtert. Die Protokollhierarchie veranschaulicht die Implementierung der Komponenten **30**, **32**, **22** und **24** für bestimmte Protokolle, die als abgeleitete Klasse jeweils ein generisches Basisklassenprotokoll „beerben". Wie in Fig. 4 gezeigt, kann ein Netzwerkausführungs-Basisklassenprotokoll **34** eine Vielzahl von „vererbenden" Netzwerkausführungsprotokollen **40**, **42**, **44** für verschiedene Netzwerk-Clients **12** umfassen, wie beispielsweise DICOM, Picker und LP, die es einer entsprechend instanziierten Netzwerkausführungskomponente **14** ermöglichen, das Vorhandensein eines bestimmten Netzwerk-Clients zu erkennen. Auf ähnliche Weise kann ein Netzwerktreiber-Basisklassenprotokoll **36** eine Vielzahl von „vererbenden" Netzwerktreiberprotokollen **46**, **48**, **50** für verschiedene Netzwerkschnittstellenanforderungen umfassen, die einem Netzwerk-Client **12** zugeordnet sind, beispielsweise DICOM, Picker oder LP. Ein Basisklassen-Netzwerkinterpreterprotokoll **38** kann eine Vielzahl von vererbenden Netzwerkinterpreterprotokollen **52**, **54**, **56** für verschiedene Arten von Eingabe-Bebilderungsvorrichtungen oder Herstellern umfassen, die einem Netzwerk-Client **12** zugeordnet sind, beispielsweise DICOM, Picker und LP.

**[0064]** Wie in Fig. 5 gezeigt, kann ein Basisklassen-Ausgabeinterpreterprotokoll **35** eine Vielzahl von vererbenden Ausgabeinterpreterprotokollen für verschiedene Arten von Ausgabe-Bebilderungsvorrichtungen **18** umfassen, wie beispielsweise ein SuperSet Ausgabeinterpreterprotokoll **41** von Imation, ein 831 Ausgabeinterpreterprotokoll **43** von Imation oder ein 952 Ausgabeinterpreterprotokoll **45** von Imation. Ein Basisklassen-Ausgabetreiberprotokoll **37** kann eine Vielzahl von vererbenden Ausgabetreiberprotokollen für verschiedene Hardwareschnittstellenanforderungen umfassen, die der Ausgabe-Bebilderungsvorrichtung **18** zugeordnet sind, wie ein Dual-Port-RAM-Ausgabetreiberprotokoll **47**, ein serielles Ausgabetreiberprotokoll **49** oder ein paralleles Ausgabetreiberprotokoll **51**. Jedes der vorstehend beschriebenen vererbenden Protokolle umfasst protokollspezifische Funktionen, die von einer Komponente **30**, **32**, **22** und **24** bereitgestellt werden, implementiert aber derartige Funktionen über eine generische Schnittstellen, die das entsprechende Basisklassenprotokoll **34**, **35**, **36**, **37**, **38** beerbt. Für jedes zuvor beschriebene Basisklassenprotokoll **34**, **35**, **36**, **37**, **38** kann eine Reihe zusätzlicher vererbender Protokolle implementiert werden, und zwar gemäß den Anforderungen der medizinischen Bebilderungssystemumgebung.

**[0065]** Die Art der Komponenten **30**, **32**, **22** und **24** ermöglicht eine wahlweise und modulare „Einsetzung" und „Entnahme" in bzw. aus einer Leitung **26** durch die Schnittstellenausführungskomponente **20**. Jede der Komponenten **39**, **32**, **22**, **24** ist mit einer anderen Komponente des gleichen Typs, aber eines anderen Protokolls, mittels einer Reihe von Softwareschnittstellen austauschbar. Diese Basisklassenschnittstelle ist ein Ausführungsbeispiel, das in jede Komponente eingebaut ist, so dass jede Komponente **30**, **32**, **22** und **24** in einer Pipeline **26** ersetzt werden kann, ohne die Konfiguration der anderen Komponenten in der Pipeline zu beeinträchtigen. Jede einzelne Komponente **30**, **32**, **22** und **24** ist also wiederverwendbar, wodurch sich die bisher notwendigen Kosten für ein Redesign erheblich reduzieren.

**[0066]** Wenn die Leitung **26** beispielsweise für die Kommunikation zwischen den Siemens Netzwerk-Clients **12** und einer Ausgabe-Bebilderungsvorrichtung **18**, die die Funktionalität des Imation SuperSet implementiert, konfiguriert werden soll, würde die Schnittstellenausführungskomponente **20** zunächst eine Netzwerkausführungskomponente **14** instanziieren, die zur Überwachung des Vorhandenseins der Siemens Netzwerk-Clients konfiguriert ist. Bei Erkennung eines Siemens Netzwerk-Clients **12** würde die Netzwerkausführungskomponente **14** eine Netzwerktreiberkomponente **30** und eine Netzwerkinterpreterkomponente **32** erstellen, die für den Betrieb gemäß dem Siemens Netzwerkprotokoll konfiguriert sind. Die Netzwerktreiberkomponente **30** würde für den Betrieb gemäß einem Netzwerktreiberprotokoll konfiguriert sein, das für den Empfang von Bebilderungsinformationen seitens des Siemens Netzwerk-Clients **12** geeignet ist. Die Netzwerkinterpreterkomponente **32** würde gemäß einem Netzwerk-Interpreterprotokoll arbeiten, das zur Erstellung erster Bebilderungsforderungen geeignet ist, und zwar gestützt auf das Format der Bildinformationen, die von dem Siemens Netzwerk-Client eingehen. Die Netzwerkausführungskomponente **14** würde dann die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** kommunikativ an eine Ausgabeinterpreterkomponente **22** binden, die ein Ausgabeinterpreterprotokoll aufweist, das zur Erzeugung zweiter Bebilderungsanforderungen geeignet ist, die von der Ausgabe-Bebilderungsvorrichtung des Typs Imation SuperSet verstanden werden, wobei die Netzwerkinterpreterkomponente **32** bereits an eine Ausgabetreiberkomponente **24** gebunden ist, die ein Ausgabetreiberprotokoll aufweist, das für die Übermittlung der zweiten Bebilderungsanforderungen über eine serielle Hardwareschnittstelle geeignet ist, die der Ausgabe-Bebilderungsvorrichtung des Typs Imation SuperSet zugeordnet ist.

**[0067]** Alternativ hierzu und sofern die Leitung **26** für die Kommunikation zwischen einem Toshiba Netzwerk-Client **12** und einer Ausgabe-Bebilderungsvorrichtung **18** des Typs Imation SuperSet konfiguriert ist, wäre es nur erforderlich, die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** mit Komponenten auszuwechseln, die gemäß den Netzwerktreiber- bzw. Netzwerkinterpreterprotokollen konfigu-

riert ist, die für die Toshiba-Modalität geeignet sind. Eine Netzwerkausführungskomponente **14**, die zur Überwachung auf Toshiba-Netzwerk-Clients **12** instanziiert wurde, würde eine Netzwerktreiberkomponente **30** und eine Netzwerkinterpreterkomponente **32** erstellen, die für den Betrieb gemäß dem Toshiba-Protokoll konfiguriert sind. Die für die Siemens Netzwerk-Clients **12** verwendete Ausgabeschnittstellenkomponente **16** könnte repliziert und in einer separaten Kommunikationsleitung **26** für Toshiba-Netzwerk-Clients verwendet werden. Die Ausgabeschnittstellenkomponente **16** würde eine für den Imation SuperSet konfigurierte Ausgabeinterpreterkomponente **22** und eine seriell für den Imation SuperSet konfigurierte Ausgabetreiberkomponente **24** umfassen und somit bereits gemäß den Anforderungen der Ausgabe-Bebilderungsvorrichtung **18** konfiguriert sein, und zwar unabhängig von dem Netzwerk-Client **12**. Die Netzwerkausführungskomponente **14** würde in einer separaten Leitung **26** die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** kommunikativ an die standardmäßige Ausgabeinterpreterkomponente **22** und Ausgabetreiberkomponente **24** binden, die für die Ausgabe-Bebilderungsvorrichtung des Typs Imation SuperSet konfiguriert sind und in einer beliebigen Leitung mit einer SuperSet-Ausgabevorrichtung verwendbar sind, welche bereits aneinander gebunden sind.

**[0068]** Als weitere Alternative und sofern die zuvor beschriebene Leitung **26** zur Kommunikation zwischen einem Toshiba-Netzwerk-Client **12** und einer Ausgabe-Bebilderungsvorrichtung **18** des Typs Imation 952 modifiziert werden müsste, wäre nur die Modifikation der Ausgabeschnittstellenkomponente **16** erforderlich. Die Netzwerkausführungskomponente **14** würde dann die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** kommunikativ an eine Ausgabeinterpreterkomponente **22** binden, die ein Ausgabeinterpreterprotokoll aufweist, das zur Erzeugung zweiter Bebilderungsanforderungen geeignet ist, die von der Ausgabe-Bebilderungsvorrichtung des Typs Imation 952 verstanden werden, die bereits an eine Ausgabetreiberkomponente **24** gebunden ist, die ein Ausgabetreiberprotokoll aufweist, das für die Übermittlung der zweiten Bebilderungsanforderungen über eine serielle Hardwareschnittstelle geeignet ist, die der Ausgabe-Bebilderungsvorrichtung des Typs Imation 952 zugeordnet ist. Somit wäre die von der Netzwerkausführungskomponente **14** erstellte Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** von einer Änderung in der Ausgabebebilderungsvorrichtung nicht betroffen, die der Kommunikationsleitung **26** zugeordnet ist.

**[0069]** Abschließend und sofern die zuvor beschriebene Leitung **26** zur Kommunikation zwischen einem Toshiba-Netzwerk-Client **12** und einer Ausgabe-Bebilderungsvorrichtung **18** des Typs Imation 952 mit Dual-Port-RAM-Schnittstelle modifiziert werden müsste, wäre nur die Modifikation der Ausgabeschnittstellenkomponente **16** erforderlich. Die Netzwerkausführungskomponente **14** würde dann die Netzwerktreiberkomponente **30** und die Netzwerkinterpreterkomponente **32** kommunikativ an eine Ausgabeinterpreterkomponente **22** binden, die ein Ausgabeinterpreterprotokoll aufweist, das zur Erzeugung zweiter Bebilderungsanforderungen geeignet ist, die von der Ausgabe-Bebilderungsvorrichtung des Typs Imation 952 verstanden werden, die bereits an eine Ausgabetreiberkomponente **24** gebunden ist, die ein Ausgabetreiberprotokoll aufweist, das für die Übermittlung der zweiten Bebilderungsanforderungen über eine Dual-Port-RAM-Hardwareschnittstelle geeignet ist, die der Ausgabe-Bebilderungsvorrichtung des Typs Imation 952 zugeordnet ist. Somit bliebe die Netzwerkausführungskomponente **14**, einschließlich der für Toshiba konfigurierten Netzwerktreiberkomponente **30** und Netzwerkinterpreterkomponente **32**, von der Modifikation nicht betroffen.

**[0070]** Die Verwendung von Vererbungskonzepten der objektorientierten Programmierung seitens der vorliegenden Erfindung hat den Vorteil der Wiederverwendbarkeit von Netzwerktreiber- und Netzwerkinterpreterkomponenten sowie die Vereinfachung in der Erstellung neuer Netzwerktreiber- und Netzwerkinterpreterkomponenten. Die Vererbung ermöglicht es, neue Komponenten durch Vergleich mit bereits entwickelten Komponenten zu definieren, was als differenzielle Programmierung („Differential Programming") bekannt ist. Innerhalb dieser Komponenten wird eine gemeinsame Funktionalität wiederverwendet, so dass diese nicht erneut entwickelt zu werden braucht. Alle an der Basisklasse vorgenommenen Fehlerbehebungen und Verbesserungen werden außerdem automatisch an die abgeleiteten Klassen weitergegeben. Auf diese Weise ermöglicht die vorliegende Erfindung die Einbeziehung neuer Protokolle in das Softwaresystem innerhalb eines üblicherweise kürzeren Zeitraums sowie die Nutzung einer kleineren Zahl von Ressourcen, als dies nach dem Stand der Technik üblich ist.

Client-Server-Hierarchie der Komponenten

**[0071]** Wie in Fig. 6 gezeigt, bildet die Schnittstellenausführungskomponente **20** in einem Ausführungsbeispiel die Leitung **26** gemäß einer Client-Server-Architektur. In Fig. 6 weist ein von Komponente A auf Komponente B gerichteter Pfeil darauf hin, dass Komponente A eine Client-Komponente der Server-Komponente B ist. Die bidirektionalen Pfeile zwischen der Netzwerktreiberkomponente **30** und dem Netzwerk-Client **12** sowie zwischen der Ausgabetreiberkomponente **24** und der Ausgabe-Bebilderungsvorrichtung **18** stellen keine Cli-

ent-Server-Beziehung dar, sondern die Hardware-/Software-Schnittstellen des medizinischen Bebilderungssystems **10**. Wie anhand der Pfeile in Fig. 6 dargestellt, definiert die Schnittstellenausführungskomponente **20** in einem Ausführungsbeispiel die Client-Server-Beziehung des Softwaresystems derart, dass: (1) die Schnittstellenausführungskomponente **20** eine Client-Komponente der Netzwerkausführungskomponente **14**, der Ausgabeinterpreterkomponente **22** und der Ausgabetreiberkomponente **24** ist; dass (2) die Netzwerkausführungskomponente **14** eine Client-Komponente der Netzwerktreiberkomponente **30** und der Netzwerkinterpreterkomponente **32** ist; dass (3) die Netzwerktreiberkomponente **30** eine Client-Komponente der Netzwerkinterpreterkomponente **32** ist; dass (4) die Netzwerkinterpreterkomponente **32** eine Client-Komponente der Ausgabeinterpreterkomponente **22** ist, und dass (5) die Ausgabeinterpreterkomponente **22** eine Client-Komponente der Ausgabetreiberkomponente **24** ist.

**[0072]** Das Client-Server-Paradigma ermöglicht eine nahtlose Integration unter den erfindungsgemäßen Komponenten. Die Client-Komponente fordert einen durchzuführenden Dienst an; der Server ist die Ressource, die die Client-Anfrage abwickelt. Der Client sendet eine Nachricht an einen Server, um den Server zur Durchführung einer Aufgabe aufzufordern, worauf der Server auf die Anfrage des Clients antwortet. Durch die Verwendung von Client-Server-Beziehungen in der vorliegenden Erfindung ergeben sich Vorteile in Bezug auf die Wartungsfreundlichkeit im Vergleich mit objektorientierten Programmierungsgrundsätzen. Hinter dem Client-Server-Konzept steht die Idee, dass separate Komponenten, die von einer objektorientierten Architektur bereitgestellt werden, nicht alle aus demselben Speicherraum ausgeführt zu werden brauchen. Client-Server-Computing fördert somit die Skalierbarkeit: jede Komponente der vorliegenden Erfindung kann ersetzt werden, wenn es der wachsende oder sinkende Verarbeitungsbedarf für diese Komponente diktiert, ohne dass die übrigen Komponenten davon wesentlich beeinträchtigt werden. Wie zuvor beschrieben, befinden sich die Komponenten der vorliegenden Erfindung innerhalb desselben Speichers, sei es auf einer Karte in der Bebilderungsvorrichtung oder in dem RAM eines Computers, an den die Vorrichtung gekoppelt ist. Sollte die Zahl der Bebilderungsvorrichtungen, mit denen die Clients kommunizieren können, relativ groß werden, könnten sich die Ausgabeschnittstellenkomponenten für jede Vorrichtung auf einer Karte in der Vorrichtung befinden, während sich die übrigen Komponenten auf einem an das Netz angeschlossenen Computer befinden können. Als Ergebnis der Übernahme eines Client-Server-Modells ermöglicht die vorliegende Erfindung die Neuanordnung einzelner Komponenten, ohne dass davon die Logik der übrigen Komponenten besonders betroffen wäre.

**[0073]** In der beschriebenen Client-Server-Beziehung der vorliegenden Erfindung ist die Ausgabetreiberkomponente **24** eine reine Server-Komponente für die Ausgabeinterpreterkomponente **22**. Die Ausgabetreiberkomponente **24** ist für die Hardwareanforderungen auf unterer Ebene zuständig und unterliegt der Steuerung durch die auf höherer Ebene angeordnete Ausgabeinterpreterkomponente **22**. Die Netzwerkinterpreterkomponente **32** ist eine Client-Komponente der Ausgabeinterpreterkomponente **22**, die einen Funktionssatz bereitstellt, mit dem die Netzwerkinterpreterkomponente die Ausgabe-Bebilderungsvorrichtung **18** steuert. Die Ausgabeinterpreterkomponente **22** initiiert niemals die Kommunikation mit der Netzwerkinterpreterkomponente **32**, sondern stellt auf Anfrage der Netzwerkinterpreterkomponente Services bereit. Die Netzwerktreiberkomponente **30** ist eine Client-Komponente der Netzwerkinterpreterkomponente **32**, die mit der Netzwerktreiberkomponente **30** kommuniziert, um die Bildinformationen von einem Client zu empfangen und zu interpretieren und die ersten Bebilderungsanforderungen zu erzeugen. Die Netzwerktreiberkomponente **30** kommuniziert direkt mit den Clients gemäß einem bestimmten Protokoll. Jede Komponente **30**, **32**, **22** und **24** ist eine Server-Komponente für die Schnittstellenausführungskomponente **20**. Die Schnittstellenausführungskomponente **20** steuert somit das gesamte Softwaresystem.

Kommunikation unter den Komponenten

**[0074]** Die Kommunikation unter den erfindungsgemäßen Komponenten erfolgt über die Ausgabe von RPCs (Remote Procedure Calls/Verfahrensfernabrufe). Ein RPC ist ein gemeinsamer Kommunikationsmechanismus, der oft in komplexen, verteilten Softwaresystemen verwendet wird. Eine Client-Komponente führt eine bestimmte Funktion aus, indem sie einen RPC an eine entsprechende Server-Komponente absetzt. Der RPC wickelt alle Mechanismen ab, die für die Kommunikation zwischen den Komponenten erforderlich sind. Jede Komponente ist derart konfiguriert, dass sie Services für eine Client-Komponente bereitstellt, wobei sie allerdings nicht weiß, von wie vielen Komponenten sie als Server-Komponente benutzt wird. Die Server-Komponenten führen einfach Anfragen der Client-Komponenten aus, ohne protokollspezifische Abhängigkeiten aufzuweisen.

**[0075]** Die Verwendung von RPCs ermöglicht der vorliegenden Erfindung die Nutzung von Vorteilen, die sich aus einem als „Kapselung" bezeichneten Konzept ergeben. Die Kapselung einer Komponente bedeutet, dass

die übrigen Komponenten nur die Services oder Aufgaben sehen, die diese Komponente anbietet, ohne zu sehen, wie diese Services und Aufgaben implementiert sind. Wie eine Komponente ihre Aktionen implementiert und wie ihre internen Daten angeordnet sind, ist also innerhalb eines prozeduralen Mantels „gekapselt", der den gesamten Zugang zu dem Objekt über RPCs vermittelt. Die Prozeduren und deren Daten sind nur für die Komponente selbst sichtbar. Die erfindungsgemäßen Komponenten sind somit gekapselte Funktionseinheiten. Anders ausgedrückt ermöglicht die Kapselung das Verstecken von Informationen und eine Datenabstraktion. Welches Verfahren von einer bestimmten Komponente verfolgt wird, ist ein Implementierungsdetail, das davon abhängt, wie die Daten verwendet werden. Die Operationen, die auf die gekapselten Daten ausgeführt werden können, werden als Teil der Schnittstelle zu der Komponente angegeben, also als RPCs. Die Implementierungsdetails der Operationen, die die gespeicherten Daten verarbeiten, können also geändert werden, ohne dass die RPCs betroffen sind. Zusammen mit der Vererbung hat das Kapselungskonzept den Vorteil, dass die Komponenten innerhalb der vorliegenden Erfindung austauschbar sind.

**[0076]** In einem Ausführungsbeispiel der vorliegenden Erfindung wird ein RPC verwendet, um eine Funktion auf folgende Weise auszuführen. Wenn ein Softwareprozess, der von einem Client durchgeführt wird, eine bestimmte Funktion ausführen muss, ruft der Prozess einfach die Funktion anhand ihres Bezeichners. Eine Softwareschicht, die innerhalb der Client-Komponente angeordnet ist, die als „Client-Stub" bezeichnet wird, fängt den Funktionsaufruf ab. Wenn der Client-Stub feststellt, dass der zur Durchführung der aufgerufenen Funktion notwendige Softwarecode bereits in einer anderen Server-Komponente vorhanden ist, erzeugt er eine Meldung, wobei er dem Funktionsaufruf alle Daten sowie die notwendige Paketierung und Adressierung mitgibt. Der Client-Stub sendet in einem Ausführungsbeispiel die Meldung über das Echtzeitbetriebssystem, das in dem Softwaresystem vorhanden ist, an die Server-Komponente. Das Servermodul enthält eine Schicht des Software-Codes, die als „Server-Stub" bezeichnet wird, die die Meldung entgegennimmt. Der Server-Stub entnimmt die Meldung und ruft die richtige lokale Funktion ggf. in Verbindung mit Daten auf, die der Meldung entnommen worden sind. Die lokale Funktion wird ausgeführt, als wäre sie ursprünglich lokal aufgerufen worden, und gibt alle angeforderten Informationen zurück. Der Server-Stub erzeugt eine Antwort anhand der zurückgegebenen Informationen und sendet die Antwort über das Betriebssystem an die Client-Komponente. Bei Erhalt der Antwort entnimmt der Client-Stub die zurückgegebenen Informationen und übergibt die Informationen an den lokalen Softwareprozess, der die Funktion ursprünglich aufgerufen hat. Der lokale Softwareprozess fährt dann fort, ohne zu wissen, dass eine intermodulare Kommunikation stattgefunden hat.

Komponentendefinitionen eines Ausführungsbeispiels der vorliegenden Erfindung

**[0077]** Die folgenden Unterabschnitte stellen Details bezüglich der Art und Weise vor, in denen jedes Basisklassenprotokoll in einem Ausführungsbeispiel des erfindungsgemäßen medizinischen Bebilderungssystems aus **Fig. 1** implementierbar ist. Die Unterabschnitte stellen Definitionen und Anforderungen von Services bereit, die von jeder Komponente **30**, **32**, **22** sowie **24**, **14** bereitgestellt werden, wobei die Darstellung zur Veranschaulichung in der objektorientierten Programmiersprache C++ erfolgt, die nach Bedarf kommentiert wird. Wenn nachstehend Programmcode in C++ zur Veranschaulichung der Funktionalität einer bestimmten Komponente verwendet wird, wird ggf. das Label „Host" benutzt, um einen Netzwerk-Client **12** zu bezeichnen, und das Label „Laserabbildungsgerät" oder "LI" wird ggf. benutzt, um die Ausgabe-Bebilderungsvorrichtung **18** zu bezeichnen.

Das Netzwerkausführungs-Basisklassenprotokoll

**[0078]** Das Netzwerkausführungs-Basisklassenprotokoll umfasst in dem vorliegenden Ausführungsbeispiel einen RPC, den die Netzwerkausführungskomponente **14** benötigt, um die Client-Komponente, also die Schnittstellenausführungskomponente **20**, bereitzustellen. Der RPC wird nachstehend in Bezug auf die Art der verarbeiteten Parameter und der durchgeführten Funktionen beschrieben.

| | | | |
|---|---|---|---|
| 1. | set_debug_level | Parameters: | Type: |
| | | debug level | DEBUG_LEVEL |
| | | Returns: | Type: |
| | | void | n/a |

**[0079]** Das tatsächliche Basisklassenprotokoll für die Netzwerkausführungskomponente **14** kann in C++ folgendermaßen definiert werden:

```
class NETWORK_EXECUTIVE{
protected:
   LI_INTERFACE *li_handle;     // Zeiger auf Laserbelichter-Schnittstelle
   INT32 return_code;           // RC für OS-Operationen
   DEBUG_LEVELS debug_level;    //Debug-Level für Modul
   IMAGER_CONFIG *im_cfg;       // Belichter-/Dicom-Konfigurationsobjekt
public:
   NETWORK_EXECUTIVE(LI_INTERFACE *,
   IMAGER_CONFIG*);
   virtual~NETWORK_EXECUTIVE(void);
   virtual void_set_debug_level(DEBUG_LEVELS level); //auf neuen Debug-Level setzen
};
```

**[0080]** Das Basisklassenprotokoll für eine nach dem DICOM-Protokoll konfigurierte Netzwerkausführungskomponente kann in C++ folgendermaßen definiert werden:

```
class DICOM_EXEC : public TaskVStack, public NETWORK_EXECUTIVE{
   friend class EVENT_MGR;
private:
   void execute(void);
   Bool init_network(); //initialisiere DIMSE-Schnittstelle auf net
   int connect(); //Netzwerk überwachen und auf SCU warten
   Bool checkHeapSpace(); //Prüfen, ob genügend Speicher für neue SCU
   Bool checkHeapSpaceVer();   //Prüfen, ob genügend Speicher für neue Verify-only-SCU
public:
   int numConnections;
   int numConnectionsVer;
   Bool verification_only;
   int port;
   int network_socket; // Socket für Ausführungsmodul, um das Netz abzuhören
   int assoc_sockfd; // Socket für neue Association
   DIMSE_nethandle netHandle;
      DICOM_EXEC(LI_INTERFACE *, IMAGER_CONFIG *);
   ~DICOM_EXEC(void);
   void async_handler(char, ID);
   //Asynkrone Befehle vom Dicom Driver
   void set_debug_level(DEBUG_LEVELS level); //setze auf neuen Debug-Level
};
```

**[0081]** In diesem Beispiel enthält die DICOM-Ausführungsbasisklasse zwei RPCs: set_debug_level() und async_handler(). Der async_handler() RPC ermöglicht einem DICOM_Driver, um das DICOM_executive darüber zu informieren, dass es eine Aufgabe abgeschlossen hat und beendet werden sollte.

Das Netzwerktreiber-Basisklassenprotokoll

**[0082]** Das Netzwerktreiber-Basisklassenprotokoll kann in dem vorliegenden Ausführungsbeispiel zwei RPCs umfassen: set_debug_level() und ni_event_handler(). Die RPCs werden nachstehend in Bezug auf die Art der verarbeiteten Parameter und der durchgeführten Funktionen beschrieben.

| | | |
|---|---|---|
| 1. set_debug_level | Parameter: | Typ: |
| | debug level | DEBUG_LEVEL |
| | Returns: | Typ: |
| | void | nicht vorhanden |
| 2. ni_event_handler | Parameter: | Typ: |
| | Network Interpreter event | NI_EVENT |
| | Returns: | Typ: |
| | void | nicht vorhanden |

**[0083]** Der RPC ni_event-handler empfängt asynchrone Ereignisse von der Ausgabe-Bebilderungsvorrichtung **18**, die über die Netzwerkinterpreterkomponente **32**, die Ausgabeinterpreterkomponente **22** und die Ausgabetreiberkomponente **24** weitergegeben werden.

**[0084]** Wie zuvor erwähnt, stellt die Netzwerktreiberkomponente **30** einen Mechanismus zur Handhabung asynchroner Ereignisse bereit, die von der Ausgabe-Bebilderungsvorrichtung **18** empfangen wurden. Die Ereignisse dienen dazu, die Netzwerktreiberkomponente **30** über eine Statusänderung an der Ausgabe-Bebilderungsvorrichtung **18** zu informieren. Verschiedene Ereignisse, die den Status der Ausgabe-Bebilderungsvorrichtung **18** bezeichnen, sind u.a. (1) NI_printer_update, was anzeigt, dass die Ausgabe-Bebilderungsvorrichtung ihren Status geändert hat, und (2) NI_print_job_update, was anzeigt, dass ein Bebilderungsauftrag seinen Status geändert hat. Die Funktion der vorstehenden Statusereignisse besteht darin, zu vermeiden, dass der Netzwerk-Client **12** die Ausgabe-Bebilderungsvorrichtung **18** fortlaufend abfragen muss.

**[0085]** Das tatsächliche Basisklassenprotokoll für die Netzwerktreiberkomponente **30** kann in C++ folgendermaßen definiert werden:

```
class NETWORK_DRIVER{
protected:
   INT32 return_code;          // RC für OS-Operationen
   ID assoc_socket; // Socket-Deskriptor für association
   IMAGER_CONFIG *imager_config; // Konfigurationsinformationen
   NETWORK_DRIVER(ID port, DEBUG_LEVELS,IMAGER_CONFIG *);
   virtual.about.NETWORK_DRIVER(void);
public:
   DEBUG_LEVELS debug_level;    //Debug-Level für Modul
   virtual void set_debug_level(DEBUG_LEVELS level); //setze auf neuen Debug-Level
   virtual void ni_event_handler(NI_EVENT,NI_async_data)=0;
};
```

**[0086]** Das Basisklassenprotokoll für eine nach dem DICOM-Protokoll konfigurierte Netzwerktreiberkomponente kann ein Objekt DD_NET_MONITOR verwenden, das in C++ folgendermaßen definiert werden kann:

```
class DD_NET_MONITOR : public TaskVStack{
private:
   DICOM_DRIVER *master; //Zeiger auf kontrollierendes Objekt
   void execute(void); //Hauptausführungs-Thread
public:
   DD_NET_MONITOR(DICOM_DRIVER *);
   ~DD_NET_MONITOR(void);
};
typedef enum {
   DD_PrinterStatusChange,
   DD_JobStatusChange
} DD_event
```

**[0087]** DD_NET_MONITOR ist ein Objekt, das sich in einem Objekt DICOM_DRIVER befindet, das die DICOM-Treiberkomponente implementiert. Das Objekt DD_NET_MONITOR überwacht kontinuierlich das Netzwerk auf eingehende Nachrichten und informiert bei Eintreffen einer Nachricht das Objekt DICOM_DRIVER. Das Objekt DICOM_DRIVER liest und verarbeitet die Meldungen, wobei Informationen an das Objekt DICOM_INTERPRETER (Netzwerkinterpreterkomponente **32**) über RPC-gestützte Funktionen weitergegeben werden, die von der Netzwerkinterpreterkomponente definiert sind.

**[0088]** Das Basisklassenprotokoll für eine nach dem DICOM-Protokoll konfigurierte Netzwerktreiberkomponente kann in C++ folgendermaßen definiert werden:

```
class DICOM_DRIVER : public TaskVStack,
public NETWORK_DRIVER {
    friend class DD_NET_MONITOR;
private:
    DD_NET_MONITOR network_monitor;
    //hole Meldungen vom Netzwerk
    //Association-Handlingverfahren und -parameter
    void associationServer(); //Association-Handling
    void monitorNetwork(); //DICOM-Meldungen kontinuierlich entgegennehmen und
prüfen
    int processMessage(); //empfangene DICOM-Meldung handeln
    //FilmSession-Verfahren und -parameter
    void handleFilmSession(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
    void fsNCreate(DIMSE_CmdSet cmdSet,  DIMSE_MsgHandle msg);
    void fsNSet(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
    void fsNAction(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
    void fsNDelete(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
    //FilmBox-Verfahren und -parameter
    void handleFilmBox(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
    void fbNCreate(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
    void fbNSet(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
    void fbNAction(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
    void fbNDelete(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
    //ImageBox-Verfahren und -Parameter
    void handleImageBox(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
```

```
void ibNSet(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
//Annotation Box-Verfahren und -Parameter
void handleAnnoBox(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
void abNSet(DIMSE_CmdSet cmdSet, DIMSE_MsgHandle msg);
//Printer-Verfahren und -parameter
void handlePrinter(DIMSE_CmdSet cmdSet);
void prNGet(DIMSE_CmdSet cmdSet);
//PrintJob-Verfahren und -Parameter
void handlePrintJob(DIMSE_CmdSet cmdSet);
void pjNGet(DIMSE_CmdSet cmdSet);
//verschiedene Funktionen
void async_event_server(); // aufgerufene Serverfunktionen zur Verarbeitung asynchroner Ereignisse
public:
DICOM_DRIVER(DICOM_EXEC *,ID,
DICOM_INTERPRETER *,
DIMSE_AssocHandle,DIMSE_SOPInfo *,
    DEBUG_LEVELS,IMAGER_CONFIG *);
~DICOM_DRIVER(void);
void ni_event_handler(NI_EVENT,NI_async_data);
};
```

**[0089]** In diesem Beispiel umfasst der DICOM_DRIVER eine große Zahl von Funktionen, die auf die eingehenden DICOM-Meldungen wirken. Die meisten Funktionen können DICOM-spezifisch sein und sind für einschlägige Fachleute unter Bezug auf den DICOM-Standard verständlich. Jede dieser Funktionen ist intern und eng an die betreffenden DICOM DIMSE Befehle gebunden. Außerdem enthält der DICOM_DRIVER den RPC, der in der Basisklasse network_driver angegeben worden ist: ni_event_handler(). Die DICOM-Funktionen rufen netzwerkinterpreterspezifische Funktionen auf, die den RPC-Mechanismus verwenden.

Das Netzwerkinterpreter-Basisklassenprotokoll

**[0090]** Das Netzwerkinterpreter-Basisklassenprotokoll umfasst in dem vorliegenden Ausführungsbeispiel RPCs, die die Netzwerkinterpreterkomponente **32** anfordern, um die Client-Komponente, also die Netzwerkausführungskomponente **14**, bereitzustellen.

**[0091]** Das eigentliche Basisklassenprotokoll für die Netzwerkinterpreterkomponente **32** kann in C++ folgendermaßen definiert werden, wobei der Netzwerkinterpreter als "NETWORK INTERFACE" bezeichnet wird:

```
//Asynchrone Ereignisse für Netzwerkschnittstelle definieren
typedef enum {
   NI_printer_update, // Bekanntgeben, dass sich LI-Status geändert hat
   NI_print_job_update // Bekanntgeben, dass sich Auftragsstatus geändert
} NIEVENT;
typedef union {
   int id; // ID der Komponente, die ihren Status geändert hat
} NI_async_data;
typedef NETWORK_DRIVER *ND_PTR; // Zeiger auf ND-Client
typedef void (NETWORK_DRIVER::*ND_METHOD_PTR)
      // Zeiger auf Client-Verfahren
   (NI_EVENT, NI_async_data);
class NETWORK_INTERFACE{
protected:
   INT32 return_code; // RC for OS operations
   ID port_id; // Netzwerkanschluss initialisieren
   Semaphore rpc_reply; // RPC-Antwort abgeschlossen
   Semaphore rpc_free;  // RPC-Mailbox frei
   Semaphore event_reply; // asynchrones Ereignis empfangen
   Semaphore event_free; // Mailbox für asynchrone Ereignisse frei
   Mailbox event_mbox; // Ereignis-Mailbox
   Mailbox rpc_mbox;  // RPC-Mailbox
   ND_PTR driver; // Treibermodul ruft uns
```

```
   ND_METHOD_PTR driver_async_handler; // Zeiger auf async handler
   IMAGER_CONFIG *iconfig; // Zeiger auf Belichterkonfigurationsobjekt
   NETWORK_INTERFACE(ID, LI_INTERFACE *,
   DEBUG_LEVELS,
      IMAGER_CONFIG *);
   virtual ~NETWORK_INTERFACE(void);
   LI_INTERFACE *li_handle // Handle für LI-Schnittstelle
   DEBUG_LEVELS debug_level;    //Debug-Level für Modul
   Param_Blk parameters; // Zeiger auf Parameter
   LI_async_data li_async_data; // Daten von LI-Ereignissen
public:
   virtual Bool li_event_handler(LI_INTERFACE_EVENT, LI_async_data);
   virtual void set_async_func(ND_PTR, ND_METHOD_PTR);
   virtual void set_debug_level(DEBUG_LEVELS level); //setze neuen Debug-Level
};
```

**[0092]** Ein Basisklassenprotokoll für eine nach dem DICOM-Protokoll konfigurierte Netzwerkinterpreterkomponente kann in C++ folgendermaßen definiert werden:

```
// DICOM-Schnittstellenmeldungstyp für RPCs
   typedef struct {
   DICOM_Command command;
   // auszuführender DICOM-Befehl
   DICOM_Response *response; // Zeiger auf Antwortobjekt
   DICOM_Data data;  // Meldungsdaten
} DICOM_Message;
class DICOM_INTERFACE : public TaskVStack, public
NETWORK_INTERFACE,
        public PrintServerInf, public SessionHandler {
private:
      DICOM_Message *message; // Zeiger auf RPC-Client-Meldung
      DICOM_Response *response; // Zeiger auf Antwortobjekt
      DICOM_Data *data; // Meldungsdaten
```

```
    INT32 return_code; // rpc-Return-Code
    PrinterStatus currentPrStatus;
    void execute(void);
    void send_rpc(DICOM_Message *);
    // Meldung an Server-Thread senden
    void ack_rpc(void); // RPC-Beendigung quittieren
public:
    DICOM_INTERFACE(ID LI_INTERFACE *, DEBUG_LEVELS,
     IMAGER_CONFIG *);
    ~DICOM_INTERFACE(void);
    Bool li_event_handler(LI_INTERFACE_EVENT,LI_async_data);
    // Handler für Asynchrones Ereignis
};
class PrintServerInf : public BasePrintServer {
public:
    // Constructor - Datenelemente initialisieren und
    // Verbindung mit PrintServer-Prozess herstellen.
    PrintServerInf(DICOM_INTERFACE*,IMAGER_CONFIG*);
    // Destructor - Sitzung und dynamisch zugewiesenen Speicher aufräumen
    ~PrintServerInf();
    // Dieses Verfahren öffnet eine Filmsitzung am PrintServer und
    // gibt einen Zeiger auf ein Objekt FilmSessionInf zurück.
    FilmSessionInf* openSession() { return openSession(String()); }
    FilmSessionInf* openSession(const String& origId);
    // Dieses Verfahren schließt eine Sitzung am PrintServer, worauf
    // alle zuvor in der Sitzung gespeicherten Bilder und alle
    // offenen Filmboxes gelöscht werden.
    void closeSession();
    // Dieses Verfahren holt die Statusinformationen eines Auftrags anhand dessen ID.
    bool getJobStatus(JobStatus& status, ID jobId);
    // Folgende Verfahren ermöglichen einem Client die Bearbeitung von
    // Aufträgen in der Print-Server-Warteschlange.
    // Dieses Verfahren entfernt einen Auftrag aus der Server-Warteschlange.
```

```
    bool cancel(ID jobId);
    // Dieses Verfahren ändert die Priorität eines Auftrags in der Server-Warteschlange.
    bool alterPriority(ID jobId, Priority p);
    // Dieses Verfahren gibt ein Objekt zurück, das Druckerstatusinformationen enthält
    bool getPrinterStatus(PrinterStatus& status);
    // Dieses Verfahren weist den Server an, herunterzufahren.
    void shutdown();
    void setHostName(const String& name);
    void setHostName(char* name);
protected:
    void connect();
    bool getPrintQueue(JobIdArray& jobs, ID sessionId, const String& origId);
    DICOM_INTERFACE* master;
    IMAGER_CONFIG *m_config;
private:
    FilmSessionInf* m_fs;
};
class SessionHandler {
    friend class DICOM_INTERFACE;
public:
    DICOM_INTERFACE* master;
    ID m_sessionId;
    ID m_fbId;
    ID m_imgId;
    LIST m_printJobs;
    Bool m_contrastTest;
    int m_images_acquired;
    LSVR_Status m_acquire_status;
    States m_state;
    String hostName;
    Bool m_contrastTestMode;
    PARAMETERS m_liparams;
    IMAGE *image_list[MAX_IMAGES_PER_PAGE];
```

```
IMAGE image_list_store[MAX_IMAGES_PER_PAGE];
IMAGE *raw_image_list[MAX_IMAGES_PER_PAGE];
IMAGE raw_image_list_store[MAX_IMAGES_PER_PAGE];
SessionHandler(DICOM_INTERFACE* m);
virtual ~SessionHandler();
ID getSessionId();
ID getNextFilmBoxId() { return ++m_fbId; }
ID getNextImageId() { return ++m_imgId; }
Bool queryPrintJobs(ID jobId){ return m_printJobs.query(jobId); }
void mapJobStatus(JobStatus* js);
Bool handleContrastTestEvent(char* nid);
void printContrastTest(ID image);
LSVR_Status acquireAllImageMemory(int rows, int cols);
void sessionClientHandler();
void openSessionHandler();
void closeSessionHandler();
void newFilmBoxHandler();
void printHandler();
void imageAllocateHandler();
void imageDataHandler();
void deleteImageHandler();
void filmAttrHandler();
void imageAttrHandler();
void associateImageHandler();
void eraseImageHandler();
void eraseAllImages();
void deleteFilmBoxHandler();
void getPrinterStatusHandler();
void getPrintQueue();
void getJobStatusHandler();
void cancelJobHandler();
void setPriorityHandler();
void noImageHandler();
```

```
    void errorHandler();
    void cleanUpImages();
    void updateParameters(BaseFilmSession *fs, BaseFilmBox *fb);
};
```

Das Ausgabeinterpreter-Basisklassenprotokoll

**[0093]** Die Netzwerkinterpreterkomponente **32** bildet über einen Satz von Bebilderungsobjekten eine Schnittstelle zur Ausgabeinterpreterkomponente **22**. Die Bebilderungsobjekte dienen als Parameter für die RPCs und enthalten alle verfügbaren Informationen bezüglich der Eigenschaften der Ausgabe-Bebilderungsvorrichtung **18** und des Bebilderungsprozesses. Die Netzwerkinterpreterkomponente **32** kann beliebige Teile der Informationen verwenden und den übrigen Teil ignorieren. Es gibt sechs Definitionen für Bebilderungsobjekte, nämlich (1) ein Boxobjekt, (2) ein Formatobjekt, (3) ein Bildobjekt, (4) ein Testbildobjekt, 5) ein Stringobjekt und 6) eine Vielzahl allgemeiner Bebilderungsparameterobjekte.

**[0094]** Ein Formatobjekt wird verwendet, um ein gesamtes Blatt an Bebilderungsmedien zu beschreiben, auf denen die Ausgabe-Bebilderungsvorrichtung **18** ein Bild erzeugt. Das Formatobjekt enthält Informationen bezüglich Filmtyp, Filmformat, Randfarbe, Randdichte usw. Die Eigenschaften des Formatobjekts können in C++ folgendermaßen definiert werden:

```
class FORMAT {
public:
   FORMAT(FORMAT_ID);              // Constructor
   FORMAT(FORMAT_ID);              // Constructor
   void init(void);              // Parameter auf Standardwerte initialisieren
   FORMAT_ID id;                 // Format, das dieser Box zugeordnet ist
   TABLE bkgnd_color_table;      // Hintergrund/Randfarben-Medientabelle
   TABLE bkgnd_color_mixing_table; // Hintergrund/Randfarben-Mischtabelle
   LEVEL bw_border_level;        // Schwarzweißrandstufe
   COLOR color_brd_level;        // Randfarbstufen
   LEVEL bw_density_max;         // Schwarzweiß-Maximaldichte
   FILM_TYPE film_type;          // Art des verwendeten Films
   FILM_TYPE film_type;          // Format des verwendeten Films
};
```

**[0095]** Eine Box ist ein rechtwinkliger Bereich des Filmbogens, der zur Aufnahme eines Bildes vorgesehen ist. Die Box hat zahlreiche Eigenschaften, wie beispielsweise Lage, Größe, Kontrast, Farbe usw. Die Boxdefinitionen sind einem bestimmten Format zugeordnet. Mehrere Boxen werden also in Verbindung mit einem bestimmten Format verwendet. Das folgende Beispiel in C++ beschreibt das Boxobjekt und dessen Eigenschaften:

```
class BOX {
public:
   BOX(BOX_ID id,FORMAT_ID id);          // Constructor
   BOX(void);              // Constructor
   void init(void);          // Parameter auf Standardwerte initialisieren
   BOX_ID id;              // Box-ID-Nr
   FORMAT_ID format_id       // Box formatieren
   TABLE beta_x1;          // horizontale Achse Beta-Durchgang 1
   TABLE beta_y1;          // vertikale Achse Beta-Durchgang 1
   TABLE beta_x2;          // horizontale Achse Beta-Durchgang 2
   TABLE beta_y2;          // vertikale Achse Beta-Durchgang 2
   TABLE color_media_table;    // Farbmedientabelle
   TABLE contrast_table;      // Schwarzweiß-Kontrasttabelle
   TABLE color_contrast_table; // Farbkontrasttabelle
   TABLE color_mixing_table;   // Farbmischtabelle
   FRAME frame;            // Rahmen für Rand
   LOCATION x_location;       // horizontale Pixellage
   LOCATION y_location;       // vertikale Pixellage
   Switch mirroring;         // Spiegelung ein- und ausschalten
   Switch rotation;          // Drehung ein- und ausschalten
   OUTPUT_SIZE output_size_x1; // X-Ausgabegröße, Durchgang 1
   OUTPUT_SIZE output_size_y1; // Y-Ausgabegröße, Durchgang 1
   OUTPUT_SIZE output_size_x2; // X-Ausgabegröße, Durchgang 2
   OUTPUT_SIZE output_size_y2; // Y-Ausgabegröße, Durchgang 2
OFFSET window_x_offset;     // Window X-Versatz zur Ecke
OFFSET window_y_offset;     // Window Y-Versatz zur Ecke
LENGTH window_x_length;     // horizontale Länge des Fensters
LENGTH window_y_Length;     // vertikale Länge des Fensters
};
```

**[0096]** Ein Bild wird anhand von Bilddaten dargestellt, die digitale Bildwerte enthalten. Die Bilddaten werden in einem Bildspeicher gespeichert, der der Ausgabe-Bebilderungsvorrichtung **18** zugeordnet ist. Das Bildobjekt wird verwendet, um dem Bild bestimmte Eigenschaften zuzuordnen. Wie zuvor erwähnt, können die Eigenschaften Pixellänge, Pixelbreite, Pixeltiefe, Farbformat usw. umfassen. Beim Drucken wird ein Bild verwendet, um die für das zu verwendende Format definierten Boxen auszufüllen. Das folgende Beispiel in C++ beschreibt das Bildobjekt und dessen Eigenschaften:

```
class IMAGE {
public:
   IMAGE(void);          // Constructor
   IMAGE(IMAGE_ID id);   // Constructor
   void init(void);              // Parameter auf Standardwerte initialisieren
   IMAGE_ID id;          // ID-Nummer
   COLOR_FORMAT mode;    // Farbbildformat
   LENGTH x_length;      // horizontale Bildlänge in Pixel
   LENGTH y_length;      // vertikale Bildlänge in Pixel
   DEPTH image_depth;    // Tiefe des Bildes 8-12 Bits
   DURATION timeout;     // Timeout für dieses Bild erfassen
   Switch permanent;     // Bild wird für eine Zeit gehalten
};
```

**[0097]** Um Bilder zu symbolisieren, die für Testzwecke verwendet werden, wird ein Testbildobjekt benutzt. Die Bilder werden per Software erzeugt und haben andere Attribute als ein Bild. Das folgende Beispiel in C++ beschreibt das Testbildobjekt und dessen Eigenschaften:

```
class TEST_IMAGE {
public:
   TEST_IMAGE(void);     // Constructor
   TEST_IMAGE(IMAGE_ID id);          // Constructor
   void init(void);              // Parameter auf Standardwerte initialisieren
   IMAGE_ID id           // ID-Nummer
   COLOR_FORMAT mode;    // Farbbildformat
   LENGTH x_length;      // horizontale Bildlänge in Pixel
   LENGTH y_length;      // vertikale Bildlänge in Pixel
   DEPTH image_depth;    // Tiefe des Bildes 8-12 Bits
   DURATION timeout;     // Timeout für dieses Bild erfassen
   TEST_IMAGE_TYPE       // Art des Testmusters image_type;
   LEVEL red_density;    // Konstante Dichte - Rotdichte;
   LEVEL green_density;  // Konstante Dichte - Gründichte;
   LEVEL blue_density;   // Konstante Dichte - Blaudichte;
   };
```

**[0098]** Ein Stringobjekt wird benutzt, um ASCII-Text im Bildspeicher zu halten. Das Stringobjekt ermöglicht zudem die Verwendung derartiger Parameter, wie Länge, Intensität, Typ usw. Das folgende Beispiel in C++ beschreibt das Stringobjekt und dessen Eigenschaften:

```
class STRING {
public:
   STRING(void);              // Constructor
   STRING(IMAGE_ID id);       // Constructor
   void init(void);           // Parameter auf Standardwerte initialisieren
   STRING_ID id;              // String-ID
   TEXT_TYPE type;            // Texttyp
   char *text;                // String
   LEVEL bw_foregnd_intensity;  // Schwarzweiß-Vordergrundstärke
   LEVEL bw_backgnd_intensity;  // Schwarzweiß-Hintergrundstärke
   COLOR color_foregnd_intensity; // Farbvordergrundstärke
   COLOR color_backgnd_intensity; // Farbhintergrundstärke
LENGTH width;              // Breite des Strings
LENGTH lead;               // Zahl schwarzer Linien zwischen ASCII-Zeilen
};
```

**[0099]** Das Objekt „allgemeine Parameter" wird benutzt, um alle Prozesskonfigurationsparameter zu speichern. Dieses Objekt ist verwendbar, um die Parameter in dem Laserabbildungsgerät einzustellen oder um die aktuellen Einstellungen der Parameter auszulesen. Beispiele einiger Parameter sind Standard-Betatabelle, Standard-Farbkonstrast, Standardziel, Standard-Filmformat sowie -typ usw. Einige Parameter sind nur lesbar und können somit nicht eingestellt werden, wie beispielsweise die Größe des verfügbaren Speichers, die aktuelle Softwarerevision, die Gesamtzahl der in die Warteschlange eingestellten Prints usw. Das folgende Beispiel in C++ beschreibt das Objekt „allgemeine Parameter" und dessen Eigenschaften:

```
class PARAMETERS {
public:
  PARAMETERS(void);          // Constructor
  void set_defaults(void);   // Parameter auf Standardwerte initialisieren
  DURATION acq_timeout;      // Timeout 1..65535 Sekunden erfassen
  TABLE def_beta_x1;         // horizontale Achse Beta-Durchgang 1
  TABLE def_beta_y1;         // vertikale Achse Beta-Durchgang 1
  TABLE def_beta_x2;         // horizontale Achse Beta-Durchgang 2
  TABLE def_beta_y2;         // vertikale Achse Beta-Durchgang 2
  LEVEL def_bw_border;       // Schwarzweißrandstufe
  COLOR def_color_border;    // Farbrandstufe
  COLOR_FORMAT def_cformat;  // Bildformat für Standarderfassung
  TABLE def_bw contrast;     // Standardkontrasttabelle in Schwarzweiß
  TABLE def_color_contrast;  // Standardkontrasttabelle in Farbe
  TABLE def_color_mix;       // Standardmischtabelle in Farbe
  LEVEL def_max_density;     // Standard-Maximaldichtewert
  DEPTH def_depth;           // Standard-Bits je Pixel
  DESTINATION def_destination; // Standardziel für Druckbilder
  LEVEL def_bw_dmax;         // Standardmaximaldichtewert für Schwarzweiß.
  IMAGE_TYPE def_image_type; // Standard für akzeptablen Bildtyp
```

```
FILM_TYPE def_film_type;     // Standardmedien
FILM_SIZE def_film_size;     // Standardmediengröße
LENGTH def_image_xsize;      // Standardbreite des Bildes in Pixel
LENGTH def_image_size;       // Standardbreite des Bildes in Zeilen
Switch fixed_formatting;     // Schalter für feste Formatierung
FIXED_FORMAT fixed_format;   // Feste Formatnummer
/**Nur lesbare Parameter**/
long int fixed_image_pattern; // Bilderfassungsmuster
MEMORY memory;               // Speicherstatusstruktur
OP_MODE op_mode;             // Betriebsmodus
RELEASE revision;            // Aktuelle Revision
SYSTEM system;               // Bebilderungssystem des Laserbelichters
int total_queued;            // Gesamte Drucke im System
int total_completed;         // Gesamte Drucke in aktuellen Aufträgen abgeschlossen
int total_failed;            // Gesamte Drucke in aktuellen Aufträgen fehlgeschlagen
};
```

**[0100]** Eine der Hauptaufgaben der Ausgabeinterpreterkomponente **22** besteht darin, den Status der Ausgabe-Bebilderungsvorrichtung **18** mit der Client-Komponente, also der Netzwerkinterpreterkomponente **32**, in Beziehung zu setzen. Dieser Prozess erfolgt in zwei Stufen. Wenn die Ausgabeinterpreterkomponente **22** eine Statusänderung in der Ausgabe-Bebilderungsvorrichtung **18** erkennt, wird der Event-Handler in der Client-Komponente direkt von der Ausgabeinterpreterkomponente gerufen. Ein Status-Ereignis wird an den Event-Handler übergeben. Mögliche Ereignisstati sind (1) FP_status_change, (2) PR_status_change, (3) IMS_status_change, (4) JOB_status_change und (5) XFR_status_change. Die Ausgabetreiberkomponente **24** benachrichtigt den Client, also die Ausgabeinterpreterkomponente **22**, über die vorstehenden Statusänderungen, so dass die Netzwerkinterpreterkomponente das Laserabbildungsgerät nicht ständig abzurufen braucht.

**[0101]** Der Client, also die Netzwerkinterpreterkomponente **32**, ignoriert entweder die Statusänderung oder fragt weitere Informationen an. Alle Statusinformationen sind in fünf Statusobjekten enthalten. Es gibt ein Statusobjekt für den Filmprozessor, den Drucker, das Bildverwaltungssystem, Aufträge und Hintergrundaufträge (Transfers). Jedes Statusobjekt weist ein Statusfeld auf, das einfach daraufhin geprüft werden kann, ob Warnungen oder Fehler vorhanden sind. Wenn Warnungen oder Fehler vorhanden sind, kann eine weitere Untersuchung der Warnstruktur oder der Fehlerstruktur erfolgen. Der Client kann nach Wahl nur die Informationen verwenden, die er benötigt. Das folgende Beispiel in C++ zeigt die Definition für jedes Statusobjekt und die darin enthaltenen Strukturen:

```
/**Filmprozessorstatus, Definition von Objekttypen und Klassen **/
class Film_Processor {
public:
   Film_Processor(void);     // Constructor
   void clear(void);         // Statusobjekt löschen
   int id;                   // ID
   int Warming time;         // Aufwärmzeit
   FP_Type type;             // Filmprozessortyp
   FP_Status status;         // Filmprozessorstatus
   FP_Warnings warnings;     // Aktuelle Warnungen im Filmprozessor
   FP_Errors errors;         // Aktuelle Fehler im Filmprozessor
};
typedef enum {
   Antares_FP,               // Antares-Filmprozessor
   LT_SE154_FP               // LT-Filmprozessor
   No_FP,                    // Kein Filmprozessor angeschlossen
   Spectrum_FP               // Spectrum-Filmprozessor
} FP_Type;
typedef struct {
   unsigned Busy  : 1;       // Prozessor im Clean-Up-Zustand oder beschäftigt mit Medien
   unsigned NoFP  : 1;       // Kein Filmprozessor angeschlossen
   unsigned OpenLoop : 1;    // Keine Kalibrierung
   unsigned Ready  : 1;      // Fertig zur Filmverarbeitung
   unsigned Warming : 1;     // Aufwärmen
   unsigned Warnings: 1;     // Warnungen vorhanden
   unsigned Errors  : 1;     // Fehler vorhanden
} FP_Status;
```

```
typedef struct {
   unsigned CheckChem : 1;   // Chemikalien verschlechtern sich
   unsigned Generic  : 1;    // Verschiedenes
   unsigned HiOvf  : 1;      // Ein oder mehrere Überlauftanks werden voll
   unsigned LoChem  : 1;     // Ein oder mehrere Überlauftanks werden leer
} FP_Warnings;
typedef struct {
   unsigned FPDown  : 1;     // Prozessor nicht betriebsbereit
   unsigned FullOvf : 1;     // Ein oder mehrere Überlauftanks sind voll
   unsigned Generic  : 1;    // Verschiedenes
   unsigned MediaJam : 1;    // Medienstau im Filmprozessor
   unsigned OutChem : 1;     // Ein oder mehrere Überlauftanks sind leer
} FP_Errors;
/**Bildmanagementsystemstatus, Definition von Objekttypen und Klassen **/
class Image_Mgmnt_System {
public:
   Image_Mgmnt_System(void); // Constructor
   void clear(void);         // Statusobjekt löschen
   IMS_status status;        // Status des Bildmanagementsystems
   IMS_errors errors;        // aktuelle Fehler im Bildmanagementsystem
};
typedef struct {
   unsigned PowerUp : 1;     // Erster Status seit hochfahren
   unsigned Errors : 1;      // Fehler im System vorhanden
} IMS_status;
typedef struct {
   unsigned Badconfig  : 1;  // IMS falsch konfiguriert
   unsigned BadTblEprom : 1; // Tabellen-EPROMS haben falsche Prüfsumme
   unsigned IMNVRamErr : 1;  // Kein Fehler im flüchtigen RAM im Eingabemodul
   unsigned IMSDown  : 1;    // IMS nicht betriebsbereit
   unsigned OMNVRamErr1 : 1; // Kein Fehler im flüchtigen RAM im Ausgabemodul 1
   unsigned OMNVRamErr 2: 1; // Kein Fehler im flüchtigen RAM im Ausgabemodul 2
   unsigned MemBlkErr : 1;   // 10% oder mehr des Bildspeichers ist schlecht
```

```
} IMS_errors;
/**Druckerstatus, Definition von Objekttypen und Klassen **/
class Printer {
public:
   Printer(void);          // Constructor
   void clear(void);       // Statusobjekt löschen
   int id;                 // Drucker-ID
   int SheetsRemaining;    // # Anzahl verbliebener Bogen
   FILM_TYPE MediaType;    // Art des eingelegten Films
   FILM_SIZE MediaSize;    // Format des eingelegten Films
   int ImgPixels;          // Anzahl bebilderbarer Pixel
   int ImgLines;           // Anzahl bebilderbarer Zeilen in Medien
   Quality quality;        // Aktueller Qualitätszustand
   PR_type type;           // Druckertyp
   PR_status status;       // Druckerstatusmarken
   PR_warnings warnings;   // Aktuelle Warnungen im Drucker
   PR_errors errors;       // Aktuelle Fehler im Drucker
};
typedef struct {
   unsigned Warnings: 1;   // Warnungen im System vorhanden
   unsigned Errors : 1;    // Fehler im System vorhanden
} PR_status;
typedef enum {
   Draft,
   Photo
} Quality;
typedef enum {
   Spectrum_PR,            // Spectrum-Drucker
   Antares_PR,             // Antares-Drucker
   LT_SE154_PR,            // LT-Drucker
   No_PR,                  // Kein Drucker angeschlossen
   XL_PR                   // XL (Roadrunner) Drucker
} PR_type;
```

```
typedef struct {
   unsigned MediaLow  : 1;   // Medien gehen zur Neige (weniger als 20 Blatt).
   unsigned Busy  : 1;       // Drucker weist vorübergehend ein Problem auf
   unsigned PrCalib  : 1;    // Drucker erstellt einen Kalibrierungsbogen
} PR_warnings;
typedef struct {
   unsigned BadCass  : 1;    // Medienkassette nicht betriebsbereit
   unsigned CassErr  : 1;    // Kassettenfehler liegt vor
   unsigned CassJam  : 1;    // Medienstau an der Kassette
   unsigned CoverOpen : 1;   // Eine der Klappen ist geöffnet
   unsigned ExpJam  : 1;     // Medienstau am Belichtungspunkt
   unsigned MediaOut  : 1;   // Keine Medien im Drucker
   unsigned NoCass  : 1;     // Keine Medienkassette im Drucker
   unsigned PanelErr  : 1;   // Drucker-LCD-Bedienfeld nicht betriebsbereit
   unsigned PrDown  : 1;     // Drucker nicht betriebsbereit
   unsigned RecMagFull : 1;  // Ausgabemagazin voll und muss geleert werden
   unsigned RecMagMiss : 1;  // Ausgabemagazin nicht im Drucker
   unsigned ToExpJam  : 1;   // Medienstau im Transport zum Belichtungspunkt
   unsigned ToProcJam : 1;   // Medienstau im Transport zum Filmprozessor
} PR_errors;
/**Druckeraufträge, Definition von Objekttypen und Klassen **/
class Job {
public:
   Job(void);              // Constructor
   void clear(void);       // Statusobjekt löschen
   int id;                 // Auftrags-ID
   int PrintsComplete;     // # Druckaufträge einwandfrei ausgeführt
   int PrintsFailed;       // # Druckaufträge nicht einwandfrei ausgeführt
   int PrintsQueued;       // Anzahl zu druckender Druckaufträge
   int FilmsComplete;      // Anzahl einwandfrei gedruckter Bogen
   int FilmsFailed;        // Anzahl nicht einwandfrei gedruckter Bogen
   int FilmsQueued;        // Anzahl von Druckaufträgen in Warteschlange
   JOB_status status;      // Druckauftragsstatus
```

```
   JOB_errors errors;        // Aktuelle Fehler im Druckauftrag
};
typedef struct {
   unsigned Done  : 1;       // Auftrag abgeschlossen
   unsigned Killed : 1;      // Auftrag abgebrochen
   unsigned Stopped : 1;     // Auftrag angehalten
   unsigned Wait  :          // Drucke in Warteschlange
   unsigned Errors : 1;      // Auftrag weist Fehler auf
} JOB_status
typedef struct {
   unsigned Aborted  : 1;    // Abbruchbefehl ausgegeben
   unsigned BadBand  : 1;    // Bilder nicht in einem Band
   unsigned BadMedia  : 1;   // Keine Medien vorhanden
   unsigned BadTable  : 1;   // Ungültige Tabelle angegeben
   unsigned CrossPrtErr : 1; // Ungültige Konfiguration'
   unsigned FPErr  : 1;      // Filmprozessor fehlerhaft
   unsigned ImgAbut  : 1;    // Bilder stoßen unzulässig aneinander
   unsigned IMSErr  : 1;     // Bilder stoßen unzulässig aneinander
   unsigned LinePixelErr : 1; // Zu viele Pixel
   unsigned MaxBadCnt  : 1;      // Zwei identische Fehler
   unsigned MaxBandImg : 1;  // maximale Bilder je Band
   unsigned MaxHorImg  : 1;  // maximale horizontale Bilder
   unsigned MinBand  : 1;    // Mindestzeilenzahl je Band unterschritten
   unsigned Parity  : 1;     // Paritätsfehler in einem Bild
   unsigned PrErr  : 1;      // Drucker fehlerhaft
   unsigned RecMagErr  : 1;  // Eingangsmagazin fehlt oder voll
   unsigned WrongQual  : 1;  // Qualität nicht verfügbar
} JOB_errors;
/** Übergabeauftragsstatus, Definition von Objekttypen und Klassen **/
class Xfr {
public:
   Xfr(void);              // Constructor
   void clear(void);       // Statusobjekt löschen
```

```
   int id;              // Auftrags-ID
   Length X_size;       // Horizontale Bildgröße (falls Auftrag abgeschlossen)
   Length Y_size;       // Vertikale Bildgröße (falls Auftrag abgeschlossen)
   XFR_status status;   // Auftragsstatus
   XFR_errors errors;   // Aktuelle Fehler im Auftrag
};
typedef int Length;
typedef struct {
   unsigned Wait : 1;   // Auftrag in Warteschlange
   unsigned Done : 1;   // Auftrag abgeschlossen
   unsigned Killed : 1;  // Auftrag abgebrochen
   unsigned Errors : 1;  // Auftrag weist Fehler auf
} XFR_status
typedef struct {
   unsigned Aborted  : 1;   // Abbruchbefehl ausgegeben
   unsigned AcqErr  : 1;    // Erfassungsfehler
   unsigned BadDepth  : 1;  // Angegebene Tiefe nicht einstellbar
   unsigned BadMode  : 1;   // Aktueller Modus nicht korrekt
   unsigned ConnectErr : 1; // Verbindungsfehler
   unsigned EibParamErr: 1; // Falscher Parameter in NVRAM
   unsigned EibSrcErr : 1;  // Falscher Quellenwert in NVRAM
   unsigned EibTranErr : 1; // Fehler bei Übersetzung der EIB-Parameter
   unsigned FifoErr  : 1;   // FIFO-Überlauf
   unsigned MemBoundErr : 1; // Außerhalb der Grenzen des verfügbaren Speichers
   unsigned MemErr  : 1;    // Speicherfehler während Speicherung
   unsigned MemFull  : 1;   // Bildspeicher ist voll
   unsigned NVRamErr  : 1;  // Verschiedene Fehler mit NVRAM
   unsigned ParityErr : 1;  // Paritätsfehler
   unsigned ResErr  : 1;    // Speicherung im reservierten Speicher fehlgeschlagen
   unsigned SetUpErr  : 1;  // Konfigurationsfehler
   unsigned SizeErr  : 1;   // Bildgrößenfehler
   unsigned TimeOut  : 1;   // System-Timeout während Bildspeicherung
} XFR_errors
```

**[0102]** Die Ausgabetreiberkomponente **24** stellt in diesem Ausführungsbeispiel fünfzehn Arten von RPCs bereit. Bei Verwendung der zuvor beschriebenen Bebilderungsobjekte und RPCs kann der Client die Ausgabe-Bebilderungsvorrichtung **18** vollständig betreiben. Es sei darauf hingewiesen, dass sämtliche Parameter,

die in den vorstehend beschriebenen Bebilderungsobjekten enthalten sind, auf einen „nichtzugewiesenen Wert/unassigned value" initialisiert werden. Wenn die Parameter von dem Client nicht geändert werden, ignoriert die Ausgabetreiberkomponente **24** diese. Dieses Merkmal ermöglicht dem Client, nur die Parameter zu verwenden, die er benötigt. Jeder von der Ausgabetreiberkomponente **24** bereitgestellte RPC wird nachstehend beschrieben. Im Unterschied dazu ist der zurückgegebene Wert für jeden der folgenden RPCs ein Laser Imager Response Object des Typs LI_response, wie nachstehend ausführlicher beschrieben wird.

1. RPCs für das Bedrucken von Medien

| a. print | Parameter: | Typ: |
|---|---|---|
| | copies (opt) | int |

**[0103]** Der vorstehende RPC initiiert einen allgemeinen Druckvorgang eines Laserabbildungsgeräts, der als Ausgabe-Bebilderungsvorrichtung **18** dient. Der vorstehende RPC ist zur Verwendung mit Festformaten ausgelegt. Das Format ist ein momentan gewähltes Festformat. "Copies" ist ein optionaler Parameter, der die Anzahl der zu erstellenden Kopien oder Exemplare angibt. Die seit dem letzten Druckvorgang erfassten Bilder werden für den Druck verwendet.

| b. print | Parameter: | Typ: |
|---|---|---|
| | format | int |
| | image list | LIST |
| | copies (opt) | int |
| | density (opt) | int |
| | destination (opt) | DESTINATION |

**[0104]** Der vorstehende RPC initiiert einen Druck von dem Laserabbildungsgerät. Das Format ist die zu verwendende Format-ID. Die Bildliste (image list) zeigt an, welche Bilder verwendet werden, um das Format zu füllen. "Copies" ist ein optionaler Parameter, der die Anzahl der zu erstellenden Kopien oder Exemplare angibt. Die Dichte ist eine optionale Ganzzahl, die verwendet wird, wenn ein Dichtetestfeld erwünscht ist. Der Wert der Ganzzahl entspricht einer Bild-ID. Das Ziel (destination) ist ein optionaler Parameter, der für die Ausgabe ein anderes Ziel anstelle des Standardziels angibt.

| c. print_test | Parameter: | Typ: |
|---|---|---|
| | format | int |
| | image list | LIST |
| | dens_id | IMAGE_ID |
| | copies (opt) | int |
| | destination (opt) | DESTINATION |

**[0105]** Der vorstehende RPC initiiert einen Druck von dem Laserabbildungsgerät. Das Format ist die zu verwendende Format-ID. Die Bildliste (image list) zeigt an, welche Bilder verwendet werden, um das Format zu füllen. "Dens_id" ist eine Ganzzahl, die die Bild-ID eines Dichtetestfeldes darstellt. "Copies" ist ein optionaler Parameter, der die Anzahl der zu erstellenden Kopien oder Exemplare angibt. Das Ziel (destination) ist ein optionaler Parameter, der für die Ausgabe ein anderes Ziel anstelle des Standardziels angibt.

| d. abort | Parameter: | Type |
|---|---|---|
| | job ID | JOB_ID |

**[0106]** Der vorstehende RPC bricht einen Auftrag mit der entsprechenden ID ab.

e. abort Parameter: Typ:

none nicht vorhanden

**[0107]** Der vorstehende RPC bricht alle gestarteten Aufträge ab.

2. RPC für das Formatieren

a. define Parameter: Typ:

format object FORMAT

**[0108]** Der vorstehende RPC definiert ein Format mit den in dem FORMAT-Objekt aufgefundenen Parametern. Alle Parameter, die gleich NOT_ASSIGNED sind, sind in der Definition nicht enthalten.

b. define Parameter: Typ:

box object BOX

**[0109]** Der vorstehende RPC definiert eine Box mit den in dem BOX-Objekt aufgefundenen Parametern. Alle Parameter, die gleich NOT_ASSIGNED sind, sind in der Definition nicht enthalten.

c. modify Parameter: Typ:

box object BOX

**[0110]** Der vorstehende RPC modifiziert die Box, die der in dem BOX-Objekt angegebenen ID entspricht. Alle Parameter, die in dem Boxobjekt gleich NOT_ASSIGNED sind, werden nicht modifiziert.

d. modify Parameter: Type:

box object BOX

x_shift LENGTH

y_shift LENGTH

**[0111]** Der vorstehende RPC modifiziert die Box, die der in dem BOX-Objekt angegebenen ID entspricht. Die Lage der Box wird anhand der in x_shift und y_shift genannten Angaben verschoben. Alle Parameter, die in dem Boxobjekt gleich NOT_ASSIGNED sind, werden nicht modifiziert.

e. modify Parameter: Typ:

format object FORMAT

**[0112]** Der vorstehende RPC modifiziert das Format, das der in dem BOX-Objekt angegebenen ID entspricht. Alle Parameter, die in dem Formatobjekt gleich NOT_ASSIGNED sind, werden nicht modifiziert.

f. remove Parameter: Typ:

none nicht vorhanden

**[0113]** Der vorstehende RPC löscht das zuletzt erfasste Bild.

g. remove Parameter: Type:

box object BOX

def (opt) Bool

all (opt) Bool

**[0114]** Der vorstehende RPC löscht die Box, die der ID des empfangenen BOX-Objekts entspricht. DEF ist ein optionaler Parameter, der, sofern auf TRUE gesetzt, bewirkt, dass der Auftrag zurückgestellt und im Hin-

tergrund verarbeitet wird. Wenn DEF nicht empfangen wird, wird DEF auf FALSE gesetzt. ALL ist ein optionaler Parameter, der, sofern auf TRUE gesetzt, bewirkt, dass alle definierten Boxen gelöscht werden. Wenn ALL nicht empfangen wird, wird ALL auf FALSE gesetzt.

| h. remove | Parameter: | Type: |
|---|---|---|
| | format object | FORMAT |
| | def (opt) | Bool |
| | all (opt) | Bool |

**[0115]** Der vorstehende RPC löscht das Format, das der ID des empfangenen FORMAT-Objekts entspricht. DEF ist ein optionaler Parameter, der, sofern auf TRUE gesetzt, bewirkt, dass der Auftrag zurückgestellt und im Hintergrund verarbeitet wird. Wenn DEF nicht empfangen wird, wird DEF auf FALSE gesetzt. ALL ist ein optionaler Parameter, der, sofern auf TRUE gesetzt, bewirkt, dass alle definierten Formate gelöscht werden. Wenn ALL nicht empfangen wird, wird ALL auf FALSE gesetzt.

| i. remove | Parameter: | Typ: |
|---|---|---|
| | image object | IMAGE |
| | def (opt) | Bool |
| | all (opt) | Bool |

**[0116]** Der vorstehende RPC löscht das Bild, das der ID des empfangenen IMAGE-Objekts entspricht. DEF ist ein optionaler Parameter, der, sofern auf TRUE gesetzt, bewirkt, dass der Auftrag zurückgestellt und im Hintergrund verarbeitet wird. Wenn DEF nicht empfangen wird, wird DEF auf FALSE gesetzt. ALL ist ein optionaler Parameter, der, sofern auf TRUE gesetzt, bewirkt, dass alle definierten Bilder gelöscht werden. Wenn ALL nicht empfangen wird, wird ALL auf FALSE gesetzt.

| j. remove_all | Parameter: | Typ: |
|---|---|---|
| | def (opt) | Bool |

**[0117]** Der vorstehende RPC löscht alle Bilder, Boxen, Formate und Tabellen, die in dem Laserabbildungsgerät definiert sind. DEF ist ein optionaler Parameter, der, sofern auf TRUE gesetzt, bewirkt, dass der Auftrag zurückgestellt und im Hintergrund verarbeitet wird. Wenn DEF nicht empfangen wird, wird DEF auf FALSE gesetzt.

| h. remove_fixed_images | Parameter: | Typ: |
|---|---|---|
| | none | nicht vorhanden |

**[0118]** Der vorstehende RPC löscht alle Bilder, die über RPCs in Festformaten gespeichert wurden.

3. RPC zur Bildbearbeitung

| a. store | Parameter: | Typ: |
|---|---|---|
| | none | nicht vorhanden |

**[0119]** Dieser RPC wird ausschließlich mit Festformatierung verwendet. Dieser RPC legt das nächste Bild in der nächst verfügbaren, festen Bildstelle ab. Die Stellen erstrecken sich von 1 bis N, wobei N formatspezifisch ist.

| b. store | Parameter: | Type: |
|---|---|---|
| | id | FIXED_ID |

**[0120]** Dieser RPC wird ausschließlich mit Festformatierung verwendet. Dieser RPC erfasst das nächste Bild in der durch die ID angegebenen Stelle. Die Stellen erstrecken sich von 1 bis N, wobei N formatspezifisch ist.

c. store Parameter: Typ:

image IMAGE

**[0121]** Der vorstehende RPC erfasst das nächste Bild. Der zurückgegebene Wert über die Bildgröße wird in LI_response abgelegt.

d. store Parameter: Typ:

image TEST_IMAGE

**[0122]** Der vorstehende RPC erfasst das nächste Bild als Testmuster. Der zurückgegebene Wert über die Bildgröße wird in LI_response abgelegt.

e. store Parameter: Typ:

string STRING

**[0123]** Der vorstehende RPC speichert den Text und die ID im STRING-Objekt. Dadurch kann die Client-Komponente den Text jederzeit über die id abrufen. Der zurückgegebene Wert über die Stringgröße wird in LI_response abgelegt.

f. transfer Parameter: Typ:

image IMAGE

**[0124]** Der vorstehende RPC überträgt das nächste Bild als Hintergrundauftrag. Der zurückgegebene Wert bezüglich der Bildgröße ist verfügbar, wenn die Bildübertragung abgeschlossen ist.

g. reserve Parameter: Typ:

image IMAGE

**[0125]** Der vorstehende RPC weist genügend Bildspeicher zu, um das von IMAGE-Objekt beschriebene Bild aufzubewahren.

4. RPC über Prozesskonfiguration/Status

a. set Parameter: Typ:

parameter object PARAMETER

**[0126]** Der vorstehende RPC stellt die Bebilderungsparameter für das Laserabbildungsgerät ein. Alle auf NOT_ASSIGNED gesetzten Parameter bleiben unverändert.

5. Status-RPCs

a. show Parameter: Typ:

parameter object *PARAMETER

**[0127]** Der vorstehende RPC ruft die Bebilderungsparameter für das Laserabbildungsgerät ab.

b. show_fixed Parameter: Typ:

parameter object *PARAMETER

**[0128]** Der vorstehende RPC ruft die Festformatierungs-Bebilderungsparameter für das Laserabbildungsgerät ab. Alle übrigen Elemente in dem Parameterobjekt bleiben unverändert.

| c. show_mem | Parameter: | Typ: |
|---|---|---|
| | parameter object | *PARAMETER |

**[0129]** Der vorstehende RPC ruft die Speicherbedingungen des Laserabbildungsgeräts ab.

| d. show | Parameter: | Typ: |
|---|---|---|
| | image object | *IMAGE |

**[0130]** Der vorstehende RPC ruft die Länge und Breite des Bildes ab, dessen ID der in dem Bildobjekt angegebenen ID entspricht. Alle Bildinformationen werden in dem Bildobjekt abgelegt.

| e. show | Parameter: | Typ: |
|---|---|---|
| | printer object | *PRINTER |

**[0131]** Der vorstehende RPC ruft den Status des Druckers ab, dessen ID der in dem Druckerobjekt angegebenen ID entspricht. Alle Druckerinformationen werden in dem Druckerobjekt abgelegt.

| f. show | Parameter: | Typ: |
|---|---|---|
| | job object | *JOB |

**[0132]** Der vorstehende RPC ruft den Status des Auftrags ab, dessen ID der in dem Auftragsobjekt angegebenen ID entspricht. Alle Auftragsinformationen werden in dem Auftragsobjekt abgelegt.

| g. show | Parameter: | Typ: |
|---|---|---|
| | printer object | *XFR |

**[0133]** Der vorstehende RPC ruft den Status des Übertragungsauftrags ab, dessen ID der in dem Übertragungsauftragsobjekt angegebenen ID entspricht. Alle Übertragungsauftragsinformationen werden in dem Auftragsobjekt abgelegt.

| h. show_formats | Parameter: | Typ: |
|---|---|---|
| | string | *char |

**[0134]** Der vorstehende RPC ruft einen String der IDs der definierten Formate ab.

| i. show_images | Parameter: | Typ: |
|---|---|---|
| | string | *char |

**[0135]** Der vorstehende RPC ruft einen String der IDs der erfassten Bilder ab.

| j. show_con_tables | Parameter: | Typ: |
|---|---|---|
| | string | *char |

**[0136]** Der vorstehende RPC ruft einen String der IDs der definierten Kontrasttabellen ab.

| k. show_con_tables | Parameter: | Typ: |
|---|---|---|
| | string | *char |

**[0137]** Der vorstehende RPC ruft einen String der IDs der definierten Farbkontrasttabellen ab.

| | | | |
|---|---|---|---|
| 1. | set_debug_level | Parameter: | Typ: |
| | | debug level | DEBUG_LEVEL |
| | | Returns: | Typ: |
| | | Driver return code | DRIVER_RC |

**[0138]** Der vorstehende RPC ermöglicht es der Client-Komponente, den Debug-Level der Netzwerkinterpreterkomponente **32** einzustellen. Die Debug-Level sind NO_DEBUG, LOW_DEBUG, MEDIUM_DEBUG und HIGH_DEBUG. Dieser Parameter betrifft die während des Debuggings angezeigten Informationen.

**[0139]** Ein Vorteil der Schnittstelle zu der Ausgabeinterpreterkomponente **22** besteht darin, dass jeder RPC ein ähnliches Objekt zurückgibt. Dieses Objekt wird als Laserabbildungsgeräte-Antwortobjekt (Laser Imager Response Object) bezeichnet, wie zuvor erwähnt. Innerhalb des Laserabbildungsgeräte-Antwortobjekts befindet sich eine Fülle von Informationen bezüglich des Ergebnisses des RPC. Allerdings verwendet der Client ggf. nur die Informationen, die er benötigt. Das Laserabbildungsgeräte-Antwortobjekt setzt sich aus drei Hauptfeldern zusammen. Ein erstes Feld ist ein einfacher boolescher Wert mit dem Titel „success" (Erfolg). Der boolesche Wert besagt, ob die dem RPC zugehörige Anfrage erfolgreich war oder fehlgeschlagen ist. Diese Informationen erfüllen die Anforderungen der meisten Client-Komponenten. Das zweite Feld, „success_data" (Erfolgsdaten), gibt Werte zurück, die die Client-Komponente erwartet, wenn der Befehl erfolgreich war. Normalerweise gibt es keine Informationen für einen erfolgreichen Befehl. Ein Beispiel für Informationen, die bei einem erfolgreichen Befehl zurückgegeben werden, wäre die Bildgröße, die nach erfolgreicher Ausführung des Bildspeicherbefehls zurückgegeben wird. Das dritte Feld, „errors" (Fehler), dient dazu, zu erläutern, warum der RPC fehlgeschlagen ist. Dieses Feld ist ein Gesamt-Bit-Feld von Fehlern, die am Laser-Abbildungsgerät aufgetreten sind. Auch dieses Feld ist nur gültig, wenn „success" gleich „false" ist.

**[0140]** Der nachfolgend aufgeführte Programmcode in C++ beschreibt das Laserabbildungsgeräte-Antwortobjekt. Die Klasse definiert die von dem Laser-Abbildungsgerät empfangene Antwort, nachdem ein Befehl ausgegeben worden ist. Wenn der Befehl erfolgreich ausgeführt worden ist, wird die Markierung SUCCESS auf TRUE gesetzt. Alle Daten, die bei einem erfolgreichen Abschluss empfangen worden sind, werden in Success_Data gespeichert. Wenn der Befehl nicht erfolgreich ausgeführt war, wird die Markierung SUCCESS auf FALSE gesetzt. Die Fehlerursache wird in der Struktur "failures" (Fehler) gespeichert.

```
class LI_response {
   friend SS_EXECUTIVE;
   Command cmd;            // SS-Befehl
public:
   LI_response(void);      // Constructor
   Bool success;           // Befehl erfolgreich ausgeführt
   Success_Data success_data;    // Nur gültig bei erfolgreicher Ausführung
   Failures errors;        // Falls Befehl fehlgeschlagen, Fehlerursache angeben
};
typedef struct {
   unsigned AcqErr  : 1;     // Erfassungsfehler
   unsigned AcqLockout      : 1;  // Erfassung nicht versucht, nicht verfügbar
   unsigned BadBoxId        : 1;  // Box-ID zur Modifikation nicht vorhanden
   unsigned BadDepth  : 1;  // Pixeltiefenfehler
   unsigned BadFmtId        : 1;  // Format-ID nicht vorhanden
   unsigned BadPar  : 1;          // Falscher Parameter
   unsigned BadCConTbl      : 1;  // Falsche Farbkontrasttabelle
   unsigned BadCMediaTbl    : 1;  // Falsche Farbmedientabelle
   unsigned BadConTbl       : 1;  // Falsche Kontrasttabelle
   unsigned BadCMixTbl      : 1;  // Falsche Farbmischtabelle
```

```
  unsigned BadDensTest    : 1;  // Bild ist kein gültiger Dichtetest
patch
  unsigned BadDest  : 1;    // Ungültiges Ziel
  unsigned BadImgId       : 1;  // Bild nicht gefunden
  unsigned BadJobId       : 1;  // Auftrag nicht gefunden
  unsigned BadMedia  : 1;  // Medientyp korrekt
  unsigned BadMode  : 1;  // Falscher Eingabemodus (Farbe/Schwarzweiß)
  unsigned BoxInUse  : 1   // Box wird derzeit benutzt
  unsigned Busy  : 1;      // Modul führt bereits Bildübertragung aus
  unsigned CConInUse  : 1;   // Farbkontrasttabelle wird derzeit benutzt
  unsigned ConInUse  : 1;   // Kontrasttabelle wird derzeit benutzt
  unsigned ConnectErr : 1; // Hardware-Verbindungsfehler
  unsigned EibParamErr: 1; // EIB-Parameterfehler
  unsigned EibSrcErr : 1; // Ungültige EIB-Quelle
  unsigned EibTranErr : 1; // EIB-Übertragungsparameter ungültig
  unsigned Empty  : 1;     // Mbox ist derzeit leer
  unsigned FifoErr  : 1;     // FIFO-Überlauf
  unsigned FmtFull  : 1;   // würde 255 Boxen in einem Format überschreiten
  unsigned FmtInUse  : 1;  // Format wird derzeit benutzt
  unsigned FmtOvrLap : 1;  // Die Boxen in diesem Format überlagern sich
  unsigned FmtOffSheet : 1; // Box in diesem Format passt nicht auf die Medien
  unsigned FmtTMCon       : 1;  // Zu viele Kontrasttabellen in diesem Format
  unsigned FmtTMCCon      : 1;  // Zu viele Farbverlaufstabellen in diesem Format
  unsigned FmtTMCMix : 1;       // Zu viele Farbmischtabellen in diesem Format
  unsigned FmtTMCMedia : 1;     // Zu viele Farbmedientabellen in diesem Format
  unsigned FmtTMImgs      : 1;  // Zu viele Bilder in der Bildliste
  unsigned Full  : 1;      // MBOX ist voll
  unsigned InModInUse : 1; // Eingabemodul wird derzeit benutzt
  unsigned ImgInuse  : 1;  // Bild wird derzeit benutzt
  unsigned ImgInvalid : 1; // Bild noch nicht vollständig gespeichert
  unsigned JobDone  : 1;   // Auftrag bereits abgeschlossen
  unsigned MagErr  : 1;    // Vergrößerungsfehler
  unsigned MaxFmts  : 1;   // Mehr als 255 Formate
```

```
    unsigned MaxJobs : 1;  // Maximale Zahl gleichzeitiger Aufträge würde überschritten
    unsigned MemBoundErr : 1;    // Ungültige Bildspeicheradresse
    unsigned MemErr : 1;  // Speicherfehler während des Speicherns
    unsigned MemFull : 1;  // Bildspeicher ist voll
    unsigned MissPar : 1;       // Fehlender Parameter
    unsigned MovErr : 1;  // Durch Verschiebung würde Boxlage negativ werden
    unsigned NoMem : 1;   // Nicht genügend Speicher zur Ausführung des Befehls
    unsigned NVRamErr : 1;      // Problem mit nicht flüchtigem Speicher
unsigned ParityErr 1:     // Hardware-Paritätsfehler
    unsigned PassErr : 1;  // Doppelter Durchgang erforderlich, Modul mit einem Durch-
gang
    unsigned QueueFull : 1; // Druckwarteschlange voll. Keine weiteren Aufträge möglich.
    unsigned ResErr : 1;  // Bildgröße entspricht nicht dem reservierten Speicher
    unsigned SetUpErr : 1; // Anfrage entspricht nicht der Systemkonfiguration
    unsigned SizeErr : 1;  // Größe im Img Header entspricht nicht der Bildgröße
    unsigned StoErr : 1;  // Video- oder Digitalsignalfehler während Erfassung
    unsigned TimeOut : 1;  // Bilderfassung konnte nicht abgeschlossen werden
    unsigned TooLong : 1;  // Meldung für die mbox zu lang
    unsigned Unkillable : 1; // Aufträge lassen sich nicht abbrechen
    unsigned UnknownCmd : 1;     // Unbekannter Befehl ausgegeben
    unsigned WinErr : 1;  // Angegebenes Fenster hat falsche Größe
  } Failures;
```

**[0141]** Die folgende Struktur enthält Daten, die die Ausgabe-Bebilderungsvorrichtung **18** (das Laser-Abbildungsgerät) zurückgibt, wenn der Befehl einwandfrei ausgeführt wird. Diese Daten sind somit nur gültig, wenn während der Ausführung keine Fehler auftreten.

```
typedef struct {
  ID id;              // Platzhalter für eine Return-ID
  LENGTH x_size;      // Platzhalter für eine Bildgröße
  LENGTH y_size;      // Platzhalter für eine Bildgröße
  LIST list;      // Platzhalter für eine ID-Liste
} Success_Data;
```

**[0142]** Die tatsächliche Basisklasse für die Ausgabetreiberkomponente **24** kann in C++ folgendermaßen definiert werden:

```
class LI_INTERFACE {
public:
    LI_INTERFACE(PORT_ID new_id, OUTPUT_INTERFACE *p); // Constructor
    ~LI_INTERFACE(void);
    INT32 return_code;          // Return-Code für OS-Operationen
    DRIVER_RC out_driver_rc;    // Return-Code von Ausgabetreiber
    DEBUG_LEVELS debug_level;   // Debug-Level für Modul
    Semaphore rpc_reply;        // RPC-Antwort abgeschlossen
    Semaphore rpc_free;         // RPC-Mailbox frei
    Semaphore event_reply;      // Asynchrones Ereignis empfangen
    Semaphore event_free        // Mailbox für asynchrone Ereignisse frei
    PORT_ID exec_id;
    Mailbox rpc_mbox;           // RPC-Mailbox
    Mailbox event_mbox;         // Ereignis-Mailbox
    OUTPUT_INTERFACE *output_handle;
    FE_PTR client;              // Verwendendes Client-Modul
    FE_METHOD_PTR client_async_handler;    // Zeiger auf asynchronen Handler
    virtual Bool output_ev_handler(enum IO_EVENT event) =0  // asynchr. Ereignis-Hand-
ler
    virtual void set_async_func(FE_PTR,FE_METHOD_PTR)=0; // ptr auf FE-Handler
setzen
    /*** Laserbelichter-Client-Schnittstelle ***/
    // Transparent-Grundbefehl
    virtual LI_response send(char *);  // Generischen Text senden
    virtual LI_response receive(char *); // Generischen Text empfangen
    // Druckbefehle
    virtual LI_response print(int copies=1)=0; // Festformatdruck
    virtual LI_response print(FORMAT_ID id,LIST *images,
        int copies=1,DESTINATION d=Film_Processor_1)=0;
        virtual LI_response print_test(FORMAT_ID id,LIST *images,
```

```
    IMAGE_ID dens_id int copies=1,
    DESTINATION d=Film_Processor_1)=0;
virtual LI_response abort(JOB_ID id)=0;
virtual LI_response abort(void)=0; //Abort all jobs
// Formatierungsbefehle
virtual LI_response define(BOX box)=0;    // Box definieren
virtual LI_response define(FORMAT format)=0; // Format definieren
virtual LI_response modify(BOX box)=0;    // Box modifizieren
virtual LI_response modify(LENGTH X_SHIFT, LENGTH Y_SHIFT, BOX
    box)=0;
virtual LI_response modify(FORMAT format)=0; // Format modifizieren
virtual LI_response remove(FIXED_ID); // Bilder von einer Position verschieben
virtual LI_response remove(BOX box,Bool def=FALSE,Bool all=FALSE);
                                  // Box löschen
virtual LI_response remove(FORMAT format,Bool def=FALSE,Bool
    all=FALSE);
virtual LI_response remove(BOX box,Bool def=FALSE,Bool all=FALSE);
virtual LI_response remove_fixed_images(void); // Alle festen Bilder entfernen
virtual LI_response remove_all(Bool def=FALSE); // Alles löschen
// Manipulationsbefehle
virtual LI_response reserve(IMAGE image)=0; // Speicher reservieren
virtual LI_response store(void)=0;        // Nächstes Bild speichern
virtual LI_response store(FIXED_ID)=0;    // Bild für eine Position speichern
virtual LI_response store(IMAGE image)=0; // Bild speichern
virtual LI_response store(TEST_IMAGE image)=0;  // Testbild speichern
virtual LI_response store(STRING string)=0; // Testbild speichern
virtual LI_response transfer(IMAGE image)=0; // Bild übertragen
// Mailbox-Befehle
virtual LI_response clear(MAILBOX mbox)=0; // Mailbox leeren
virtual LI_response receive(MAILBOX mbox,char *msg)=0;
                              // Meldung in eine Mailbox holen
virtual LI_response send(MAILBOX mbox,char *msg)=0;
                              // Meldung an eine Mailbox senden
```

```
// Prozesskonfiguration / Statusbefehle
virtual LI_response set(PARAMETERS ptr)=0; // Bebilderungsparameter einstellen
virtual LI_response show_fixed(PARAMETERS *);
virtual LI_response show_mem(PARAMETERS *ptr);  // Bildspeicher zeigen
virtual LI_response show_PARAMETERS *ptr)=0;    // Bebilderungsparam. zeigen
virtual LI_response show(IMAGE *ptr)=0;         // Bildinfo zeigen
virtual LI_response show(Film_Processor *ptr)=0; // Filmprozessorstatus zeigen
virtual LI_response show(Image_Mgmnt_System *ptr)=0; // IMS-Status zeigen
virtual LI_response show(Printer *ptr)=0; // Druckerstatus zeigen
virtual LI_response show(Job *ptr)=0;   // Auftragsstatus zeigen
virtual LI_response show(Xfr *ptr)=0; // Xfr-Auftragsstatus zeigen
virtual LI_response show_formats(char *ptr)=0; // String definierter Formate zeigen
virtual LI_response show_images(char *ptr)=0; // String definierter Bilder zeigen
virtual LI_response show_con_tables(char *ptr)=0; // String von Verlaufstabelle zeigen
virtual LI_response show_ccon_tables(char *ptr)=0; // String von Farbverlaufstabellen
zeigen
};
```

Ausgabetreiber-Basisklassenprotokoll

**[0143]** Die Ausgabetreiberkomponente **24** stellt fünf RPCs für die Ausgabeinterpreterkomponente **22** bereit. Mit den fünf RPCs kann die Ausgabeinterpreterkomponente **22** eine direkte Schnittstelle zu einer Ausgabe-Bebilderungsvorrichtung **18** bilden, beispielsweise einem Laser-Abbildungsgerät. Jede der fünf RPCs wird nachfolgend beschrieben:

| | | |
|---|---|---|
| 1. xmit_message | Parameter: | Typ: |
| | message | char * |
| | Returns: | Typ: |
| | Driver return code | DRIVER_RC |

**[0144]** Der vorstehende RPC übergibt der Ausgabetreiberkomponente **24** die Meldung, die über die Leitung **30** an die Netzwerk-Client **12** übertragen werden soll. Die Ausgabetreiberkomponente wickelt alle Anforderungen für die Kommunikation mit der Ausgabe-Bebilderungsvorrichtung **18** ab.

| | | |
|---|---|---|
| 2. receive_message | Parameter: | Typ: |
| | message | char * |
| | Returns: | Typ: |
| | Driver return code | DRIVER_RC |

**[0145]** Der vorstehende RPC ruft eine Meldung von der Ausgabetreiberkomponente ab, die von der Ausgabe-Bebilderungsvorrichtung **18** gesendet worden ist. Die Ausgabetreiberkomponente wickelt auch hier alle Anforderungen für die Kommunikation mit der Ausgabe-Bebilderungsvorrichtung ab.

| | | | |
|---|---|---|---|
| 3. | set_xmit_timeout | Parameter: | Typ: |
| | | timeout | int |
| | | Returns: | Typ: |
| | | Driver return code | DRIVER_RC |

**[0146]** Der vorstehende RPC setzt den Timeout-Wert, den die Ausgabetreiberkomponente verwenden sollte, wenn sie Daten an die Ausgabe-Bebilderungsvorrichtung **18** sendet.

| | | | |
|---|---|---|---|
| 4. | set_async_func | Parameter: | Typ: |
| | | client ptr | FE_CLIENT_PTR |
| | | method ptr | FE_METHOD_PTR |
| | | Returns: | Typ: |
| | | Driver return code | DRIVER_PC |

**[0147]** Der vorstehende RPC übergibt der Ausgabetreiberkomponente ein Handle an den asynchronen Handler der Client-Komponente, also der Ausgabetreiberkomponente **24**. Der vorstehende RPC wird verwendet, um die Client-Komponente über asynchrone Ereignisse zu informieren, die aufgetreten sind. Das einzige Ereignis ist MSG_PENDING, welches darauf hinweist, dass eine Meldung vollständig von der Ausgabe-Bebilderungsvorrichtung **18** empfangen wurde und für die Ausgabeinterpreterkomponente bereit steht.

| | | | |
|---|---|---|---|
| 5. | set_debug_level | Parameter: | Typ: |
| | | debug level | DEBUG_LEVEL |
| | | Returns: | Typ: |
| | | Driver return code | DRIVER_RC |

**[0148]** Der vorstehende RPC ermöglicht es der Client-Komponente, den Debug-Level für die Ausgabetreiberkomponente einzustellen. Die Debug-Level sind NO_DEBUG, LOW_DEBUG, MEDIUM_DEBUG und HIGH_DEBUG. Dieser Parameter betrifft die während des Debuggings angezeigten Informationen.

**[0149]** Wie zuvor erwähnt, gibt jeder RPC einen von drei Treiber-Rückgabecodes zurück: (1) RPC_OK, (2) PORT_BUSY und (3) NO_MESSAGE. Die Treiber-Rückgabecodes (Return-Codes) können in C++ folgendermaßen definiert werden:

```
// Return-Typen für I/O-Treiberschnittstelle
typedef enum {
RPC_OK,          // RPC wurde ausgegeben und quittiert
PORT_BUSY,       // RPC-Übertragung fehlgeschlagen, Anschluss sendet bereits
NO_MESSAGE       // RPC-Empfang fehlgeschlagen, keine Meldung anliegend
} DRIVER_RC;
```

**[0150]** Das tatsächliche Basisklassenprotokoll für die Ausgabetreiberkomponente kann in C++ folgendermaßen definiert werden:

```
class OUTPUT_INTERFACE
{
public:
   OUTPUT_INTERFACE(PORT_ID newport);
   .about.OUTPUT_INTERFACE(void);
   virtual DRIVER_RC xmit_message(char *message) = 0;
   virtual DRIVER_RC receive_message(char *message) =0;
   virtual DRIVER_RC set_xmit_timeout(int timeout) =0;
   virtual DRIVER_RC set_async_func(CLIENT_PTR,
      CLIENT_METHOD_PTR)=0; //
PORT_ID port;      // Die ID dieses Anschlusses
```

**[0151]** Obwohl die Erfindung mit besonderem Bezug auf bevorzugte Ausführungsbeispiele bereits beschrieben wurde, ist die Erfindung nicht darauf beschränkt, sondern kann innerhalb des Geltungsbereichs Änderungen und Abwandlungen unterzogen werden. Die Beschreibung und die verwendeten Beispiele sind daher nur exemplarisch zu verstehen, während Geltungsbereich und Umfang der Erfindung in den anhängenden Ansprüchen dargelegt sind.

## Patentansprüche

1. System zum Übermitteln medizinischer Bildinformationen zwischen verschiedenen medizinischen Abbildungsmodalitäten (**12**) und mindestens einem aus einer Vielzahl von unterschiedlichen Abbildungsgeräten (**18**) über eine Netzwerk-Schnittstelle (**28**), mit:
einer Netzwerk-Ausführungskomponente, die eine oder mehrere Netzwerk-Schnittstellenkomponenten (**33**) instanziiert gemäß einem ausgewählten Netzwerk-Schnittstellenprotokoll aus einer Vielzahl von Netzwerk-Schnittstellenprotokollen, wobei jede Netzwerk-Schnittstellenkomponente derart ausgebildet ist, dass sie medizinische Bildinformationen von einer der medizinischen Abbildungsmodalitäten über die Netzwerk-Schnittstelle empfängt, wobei die medizinischen Bildinformationen gemäß dem ausgewählten Netzwerk-Schnittstellenprotokoll empfangen werden, wobei jedes Netzwerk-Schnittstellenprotokoll ausgewählten medizinischen Abbildungsmodalitäten speziell zugeordnet ist, und wobei zum Erzeugen erster Abbildungsanforderungen auf der Grundlage der empfangenen medizinischen Bildinformationen die ersten Abbildungsanforderungen gemäß dem ausgewählten Netzwerk-Schnittstellenprotokoll erzeugt werden;
einer oder mehreren Ausgabeschnittstellenkomponenten (**16**), von denen jede derart ausgebildet ist, dass sie zweite Abbildungsanforderungen auf der Grundlage der ersten, von einer der Netzwerk-Schnittstellenkomponenten erzeugten Abbildungsanforderungen erzeugt, wobei die zweiten Abbildungsanforderungen gemäß einem aus einer Vielzahl unterschiedlicher Ausgangsschnittstellenprotokolle erzeugt werden, wobei jedes der Ausgangsschnittstellenprotokolle einem der Abbildungsgeräte speziell zugeordnet ist, und wobei zum Übermitteln der zweiten, von einer der Ausgangsschnittstellenkomponenten erzeugten Abbildungsanforderungen zu einem der Abbildungsgeräte die zweiten Abbildungsanforderungen gemäß dem einen der Ausgangsschnittstellenprotokolle übermittelt werden; und
einer Schnittstellen-Ausführungskomponente (**20**) zum Bilden einer oder mehrerer Übermittlungsleitungen (**26**), von denen jede Leitung eine oder mehrere medizinische Abbildungsmodalitäten (**12**) mit einer der Netzwerk-Schnittstellenkomponenten (**33**) kommunikativ verbindet unter Verwendung des gleichen Netzwerk-Schnittstellenprotokolls, einer der Ausgangsschnittstellenkomponenten (**16**) und eines der Abbildungsgeräte (**18**), wodurch mehrere medizinische Abbildungsmodalitäten unter Verwendung des gleichen Netzwerk-Schnittstellenprotokolls mit einem der Abbildungsgeräte über eine einzelne Übermittlungsleitung (**26**) kommunizieren können.

2. System nach Anspruch 1, worin jede der Netzwerk-Schnittstellenkomponenten eine erste Schnittstelle umfasst zum Übermitteln der ersten Abbildungsanforderungen zu einer der Ausgangsschnittstellenkomponenten gemäß einem Basisklassenprotokoll, das generisch ist für jede Netzwerk-Schnittstellenkomponente und von jeder Ausgangsschnittstellenkomponente verstanden wird.

3. System nach Anspruch 2, worin das Basisklassenprotokoll gemäß einer objektorientierten Hierarchie definiert ist.

4. System nach Anspruch 2, worin
jede Ausgangsschnittstellenkomponente derart ausgebildet ist, dass sie von einem der Abbildungsgeräte erste Antworten auf die zweiten Abbildungsanforderungen erhält,
wobei die ersten Antworten empfangen werden gemäß einem der Ausgangsschnittstellenprotokolle, und wobei zum Erzeugen zweiter Antworten auf der Grundlage der ersten Antworten die zweiten Antworten gemäß einem der Ausgangsschnittstellenprotokolle erzeugt werden; und
jede Netzwerk-Schnittstellenkomponente derart ausgebildet ist, dass sie auf der Grundlage der zweiten, von einer der Ausgangsschnittstellenkomponenten erzeugten Antworten dritte Antworten erzeugt, die gemäß einem der Netzwerk-Schnittstellenprotokolle erzeugt sind, und dass sie die dritten Antworten zu einer der medizinischen Abbildungsmodalitäten überträgt, wobei die dritten Antworten gemäß einem der Netzwerk-Schnittstellenprotokolle übertragen werden; und
jede der von der Schnittstellenausführungskomponente gebildeten Leitungen eine bidirektionale Leitung ist, die eine oder mehrere medizinische Abbildungsmodalitäten, eine der Netzwerkschnittstellenkomponenten, eine der Ausgangsschnittstellenkomponenten und eines der Abbildungsgeräte zur bidirektionalen Kommunikation zwischen den medizinischen Abbildungsmodalitäten und einem der Abbildungsgeräte kommunikativ verbindet.

5. System nach Anspruch 4, worin jede der Ausgangsschnittstellenkomponenten eine zweite Schnittstelle umfasst zum Übermitteln der zweiten Antworten zu einer der Netzwerk-Schnittstellenkomponenten gemäß einem zweiten Basisklassenprotokoll, das generisch ist für jede Ausgangsschnittstellenkomponente und von jeder der Netzwerkschnittstellenkomponenten verstanden wird.

6. System nach Anspruch 4, worin die Schnittstellenausführungskomponente jede der Leitungen gemäß einer Client/Server-Beziehung derart definiert, dass jede der Netzwerk-Schnittstellenkomponenten ein Client einer der Ausgangsschnittstellenkomponenten ist und dass die Schnittstellenausführungskomponente ein Client einer jeden Netzwerk-Schnittstellenkomponente ist.

7. System nach Anspruch 6, worin die Kommunikation zwischen den Netzwerk-Schnittstellenkomponenten und den Ausgangsschnittstellenkomponenten von Verfahrensfernabrufen ausgeführt wird, die erzeugt werden von den Netzwerk-Schnittstellenkomponenten und ausgeführt werden von den Ausgangsschnittstellenkomponenten, und worin die Kommunikation zwischen den Schnittstellenausführungskomponenten, den Netzwerk-Schnittstellenkomponenten und den Ausgangsschnittstellenkomponenten von Verfahrensfernabrufen durchgeführt werden, die erzeugt werden von der Schnittstellenausführungskomponente und ausgeführt von den Netzwerk-Schnittstellenkomponenten.

8. Vorrichtung zum Verteilen medizinischer Informationen, die von Abbildungsmodalitäten (**12**) auf einem Netzwerk übermittelbar sind, mit:
einem Netzwerkausführungsmittel (**14**) zum Erzeugen einer entsprechenden ersten Abbildungsanforderung in Abhängigkeit vom Empfang einer Abbildungsanforderung von einer der Abbildungsmodalitäten;
einem Ausgabeschnittstellenmittel (**16**) zum Erzeugen einer entsprechenden zweiten Abbildungsanforderung und zu deren Übermittlung zu einem Abbildungsgerät (**18**) in Abhängigkeit vom Empfang der entsprechenden ersten Abbildungsanforderung von den Netzwerkausführungsmitteln; und
einem Schnittstellenausführungsmittel (**20**) zum Instanziieren des Netzwerkausführungsmittels gemäß einem von der Abbildungsmodalität vorgegebenen Eingangsprotokoll und Instanziieren des Ausgangsschnittstellenmittels gemäß einem vom Abbildungsgerät vorgegebenen Ausgangsprotokoll,
worin das Netzwerkausführungsmittel ein Netzwerkschnittstellenmittel instanziiert, welches umfasst:
einen Netzwerktreiber (**30**) gemäß einem Netzwerktreiberprotokoll des Eingangsprotokolls zum Empfangen der Abbildungsanforderung von der Abbildungsmodalität; und
einen Netzwerkinterpreter (**32**) gemäß einem Netzwerkinterpreterprotokoll des Eingangsprotokolls zum Erzeugen der entsprechenden ersten Abbildungsanforderung; und
worin das Schnittstellenausführungsmittel (**20**) eine oder mehrere Kommunikationsleitungen (**26**) bildet, von denen jede eine oder mehrere der Abbildungsmodalitäten kommunikativ verbindet und wobei eines der Netzwerkschnittstellenmittel (**33**) das gleiche Netzwerkschnittstellenprotokoll, eines der Ausgangsschnittstellenmittel (**16**) und eines der Abbildungsgeräte (**18**) verwendet, wodurch multiple, das gleiche Netzwerkschnittstellenprotokoll verwendende Abbildungsmodalitäten mit einem der Abbildungsgeräte über eine einzelne Kommunikationsleitung (**26**) kommunizieren können.

9. Vorrichtung nach Anspruch 8, worin das Ausgangsschnittstellenmittel umfasst:
einen Ausgangsinterpreter (**22**), der spezifisch ist für ein Ausgangsinterpreterprotokoll des Ausgangsprotokolls zum Empfangen der ersten Abbildungsanforderung von den Netzwerkausführungsmitteln und zum Erzeugen der entsprechenden zweiten Abbildungsanforderung; und
einen Ausgangstreiber (**24**), der spezifisch ist für ein Ausgangstreiberprotokoll des Ausgangsprotokolls zum Übermitteln der entsprechenden zweiten Abbildungsanforderung zum Abbildungsgerät.

Es folgen 5 Blatt Zeichnungen

**Anhängende Zeichnungen**

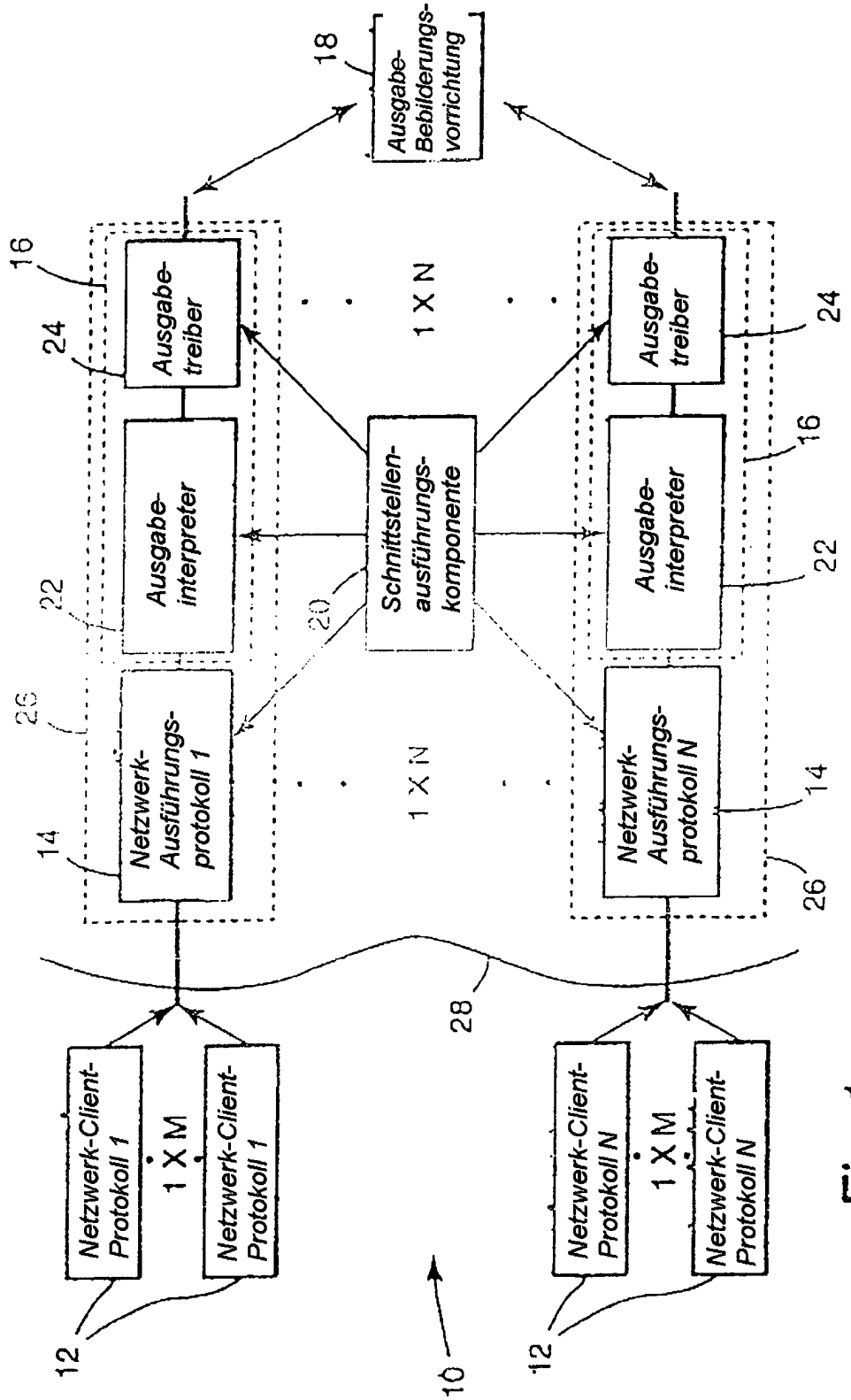


Fig. 1

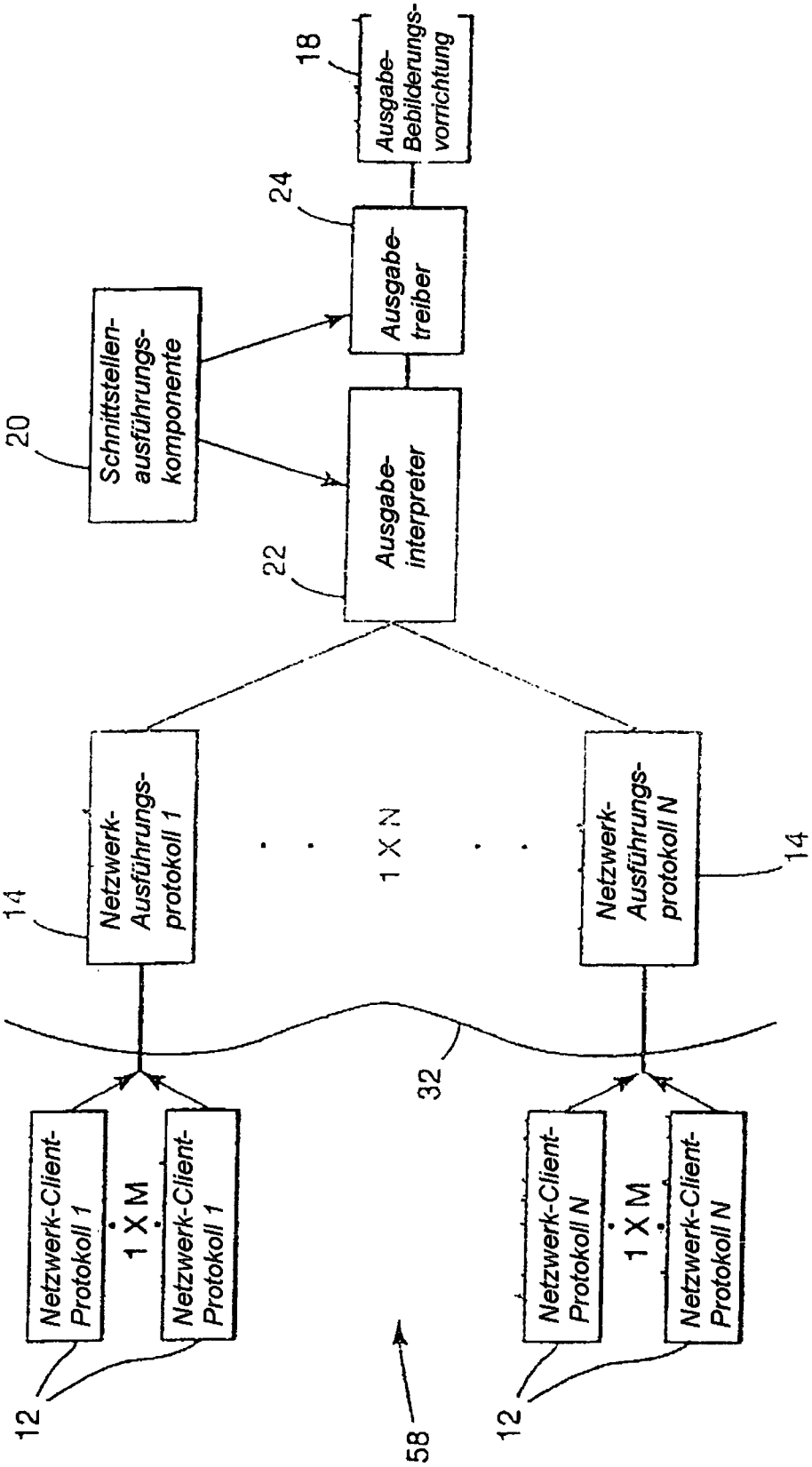
Ausgabe-Bebilderungs-vorrichtung
18
Ausgabe-treiber
24
Schnittstellen-ausführungs-komponente
20
Ausgabe-interpreter
22
Netzwerk-Ausführungs-protokoll 1
14
1 X N
Netzwerk-Ausführungs-protokoll N
14
32
Netzwerk-Client-Protokoll 1
1 X M
Netzwerk-Client-Protokoll 1
12
58
Netzwerk-Client-Protokoll N
1 X M
Netzwerk-Client-Protokoll N
12

Fig. 2

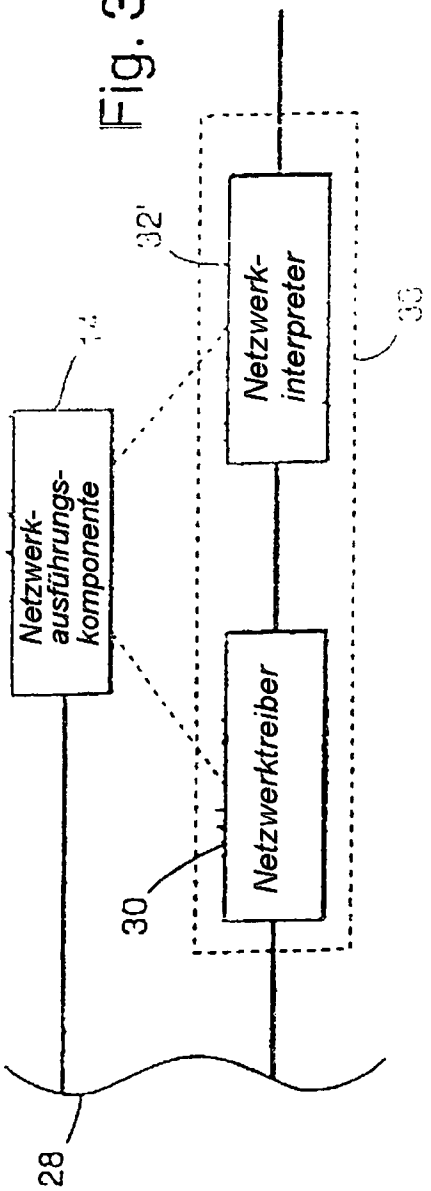
Netzwerk-
ausführungs-
komponente
Netzwerk-
interpreter
Netzwerktreiber
14
32'
30
33
28

Fig. 3

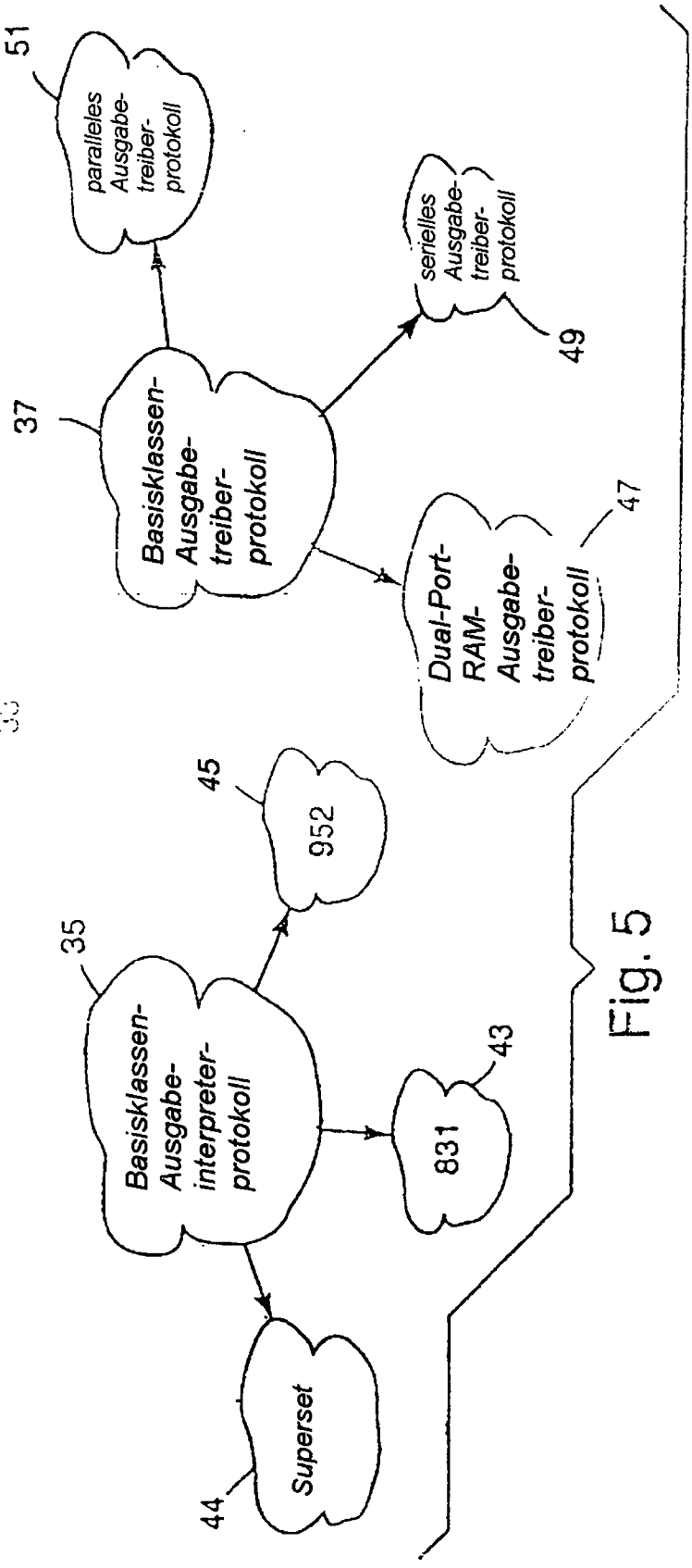
51
paralleles
Ausgabe-
treiber-
protokoll
37
Basisklassen-
Ausgabe-
treiber-
protokoll
serielles
Ausgabe-
treiber-
protokoll
49
Dual-Port-
RAM-
Ausgabe-
treiber-
protokoll
47
45
952
35
Basisklassen-
Ausgabe-
interpreter-
protokoll
831
43
Superset
44

Fig. 5

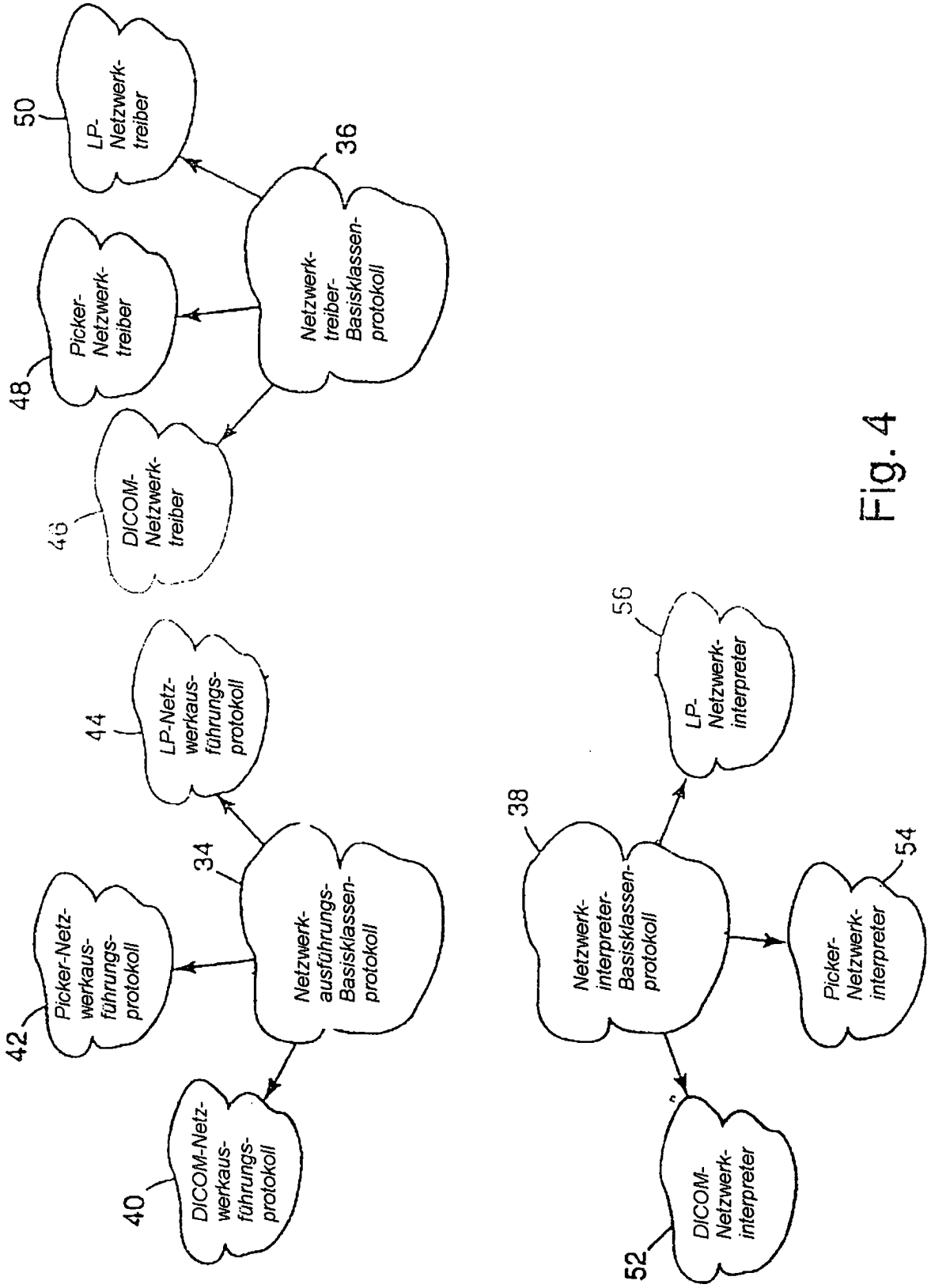
50
LP-Netzwerk-treiber
36
Netzwerk-treiber-Basisklassen-protokoll
48
Picker-Netzwerk-treiber
46
DICOM-Netzwerk-treiber
44
LP-Netz-werkaus-führungs-protokoll
34
Netzwerk-ausführungs-Basisklassen-protokoll
42
Picker-Netz-werkaus-führungs-protokoll
40
DICOM-Netz-werkaus-führungs-protokoll
56
LP-Netzwerk-interpreter
38
Netzwerk-interpreter-Basisklassen-protokoll
54
Picker-Netzwerk-interpreter
52
DICOM-Netzwerk-interpreter

Fig. 4

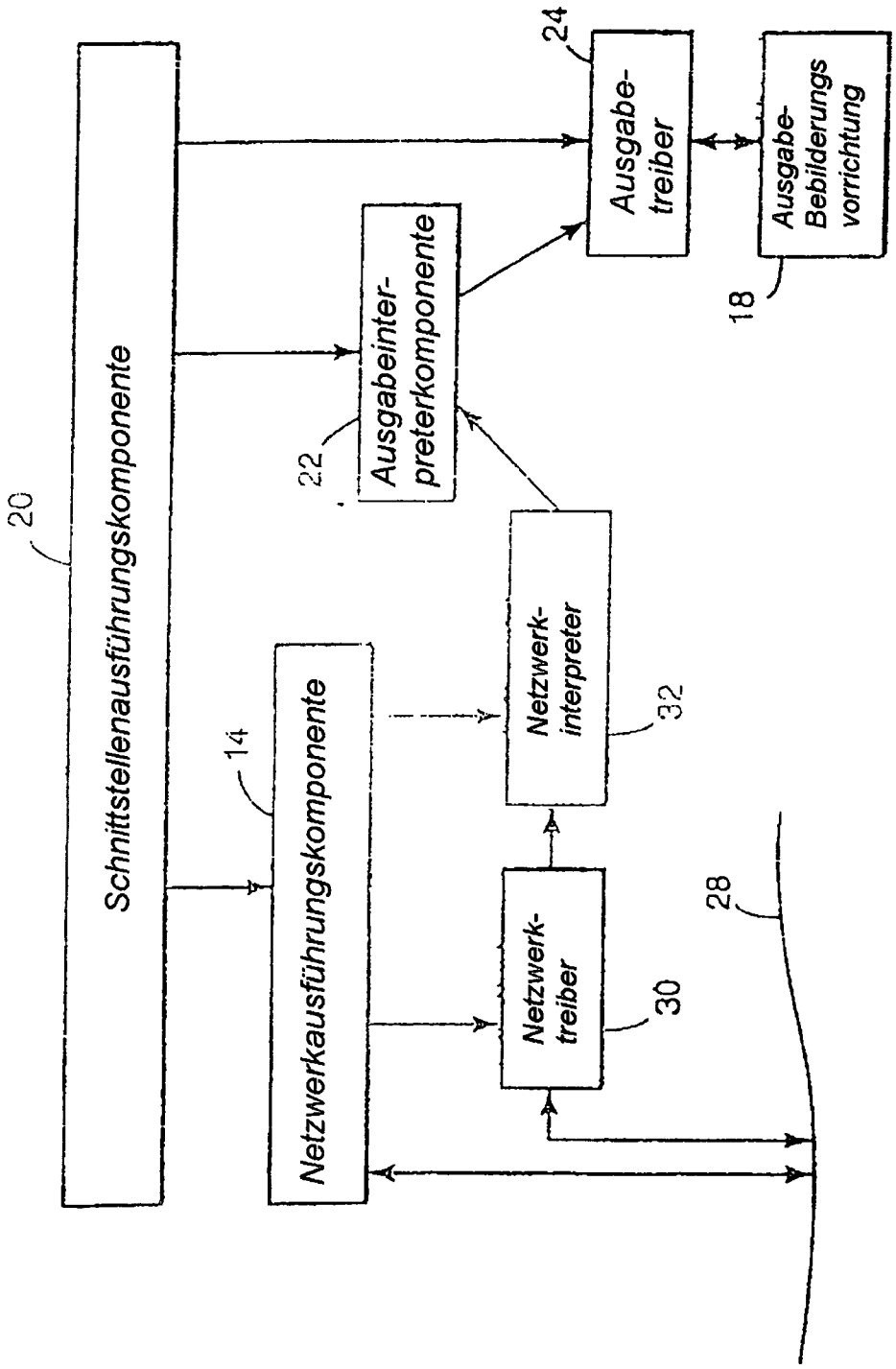
Schnittstellenausführungskomponente
20
Netzwerkausführungskomponente
14
Ausgabeinter-
preterkomponente
22
Ausgabe-
treiber
24
Ausgabe-
Bebilderungs
vorrichtung
18
Netzwerk-
interpreter
32
Netzwerk-
treiber
30
28

Fig. 6