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

(10) **DE 602 20 662 T2** 2008.02.07

(12) # Übersetzung der europäischen Patentschrift

(97) **EP 1 381 945 B1**
(21) Deutsches Aktenzeichen: **602 20 662.6**
(86) PCT-Aktenzeichen: **PCT/US02/12617**
(96) Europäisches Aktenzeichen: **02 726 784.8**
(87) PCT-Veröffentlichungs-Nr.: **WO 2002/086706**
(86) PCT-Anmeldetag: **23.04.2002**
(87) Veröffentlichungstag der PCT-Anmeldung: **31.10.2002**
(97) Erstveröffentlichung durch das EPA: **21.01.2004**
(97) Veröffentlichungstag der Patenterteilung beim EPA: **13.06.2007**
(47) Veröffentlichungstag im Patentblatt: **07.02.2008**

(51) Int Cl.[8]: ***G06F 9/44*** (2006.01)

(30) Unionspriorität:
840727 **23.04.2001** **US**

(84) Benannte Vertragsstaaten:
DE, FR, GB, IT, SE

(73) Patentinhaber:
Electronic Data Systems Corp., Plano, Tex., US

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

(72) Erfinder:
BALLANTYNE, Alando M., Austin, TX 78704, US; SMITH, Michael K., Austin, TX 78737, US; HINES, Larry M., Austin, TX 78731, US

(54) Bezeichnung: **METHODE UND SYSTEM ZUM AUSGEBEN VON XML DATEN BASIEREND AUF VORBERECHNETEN KONTEXTEN UND EINEM DOKUMENT OBJEKT MODELL**

Beschreibung

Technisches Gebiet

[0001] Diese Erfindung betrifft allgemein das Gebiet von Computersystemen und insbesondere ein Verfahren und ein System zur Ausgabe eines Berichts in XML-Daten von einem Computersystem, wie z. B. einem Altcomputersystem anhand von vorberechnetem Kontext und einem Dokumentenobjektmodell.

Hintergrund der Erfindung

[0002] Das Internet und E-Commerce verändern schnell die Art, in der die Welt Geschäfte vornimmt. Zusätzlich zu den direkten Kaufen, die über das Internet vorgenommen werden, hängen die Verbraucher zunehmend von über das Internet verfügbaren Informationen ab, um Kaufentscheidungen zu treffen. Geschäfte haben dies beantwortet, indem sie größeren Zugang zu Informationen über das Internet ermöglicht haben, sowohl direkt an die Verbraucher als auch zu anderen Geschäften, wie z. B. Zulieferern. Ein Ergebnis des erweiterten Zugangs zu elektronischer Information über das Internet ist eine verminderte Abhängigkeit und Wunsch nach gedruckten Informationen als Druckexemplar.

[0003] Extensible Mark-up Language („XML") stellt ein exzellentes Werkzeug für elektronischen Business-to-Business Handel und für die Veröffentlichung von Daten über das Internet zur Verfügung. XML spezifiziert ein Format, das in einfacher Weise für die Datenübertragung über das Internet die direkte Übertragung als ein Objekt zwischen verschiedenen Anwendungen oder die direkte Anzeige und Manipulation von Daten über eine Browsertechnologie angepasst wird. Momentan werden komplexe Transformationen auf Daten ausgeführt, die in Formaten eines Altcomputersystems ausgegeben werden, um die Daten in ein XML-Format zu bringen.

[0004] Ein Beispiel der Transformation von geschriebenen Berichten, die üblicherweise von Altcomputersystemen ausgegeben werden, zu elektronischen Berichten ist die Telefonrechnung. Historisch haben sich Telefongesellschaften auf Mainframe-Computersysteme oder Altcomputersysteme verlassen, die einen COBOL-Code ausführen, um die Telefonanrufs-Abrechnungsinformation nachzuverfolgen und zu berichten. Üblicherweise werden diese Altcomputersystem-Berichte gedruckt, kopiert und an diejenigen, die die Information benötigen, verteilt. Jedoch ist es schwierig, herkömmliche Altcomputersystem-Berichtsformate elektronisch zu übertragen oder zu manipulieren. Jetzt erhöht die elektronische Verbreitung von Rechnungen, wie z. B. über E-Mail, eine Website des Rechnungsstellers oder an eine durch den Verbraucher gewählte Abrechnungsstelle die Flexibilität und die Kontrolle der Rechnungsbezahlung, insbesondere bei komplexen geschäftlichen Abrechnungen.

[0005] Um allgemeinen herkömmliche Altcomputerberichte in verschiedenen Formaten verfügbar zu machen, wird eine komplexe Transformation der Daten anhand eines Berichtdruckdatenstromes durchgeführt. Eine Transformationstechnik besteht darin, einen „Wrapper" um das Altcomputersystem herum zu schreiben. Der Wrapper umfasst Parser und Generatoren, die Altcomputersystemberichte in XML-formatierte Ausgaben transformieren. Parser wenden eine Vielfalt von Regeln an, um die Datenausgabe in einem Altcomputerbericht zu identifizieren und zu kennzeichnen. Zum Beispiel könnte ein Parser anhand des Vorhandenseins eines Dollarzeichens oder der Position eines Dezimalpunktes in dem Datenfeld feststellen, dass ein Datenfeld einer Telefonrechnung eine Dollar-Größe darstellt, oder aufgrund des Nichtvorhandenseins von Ziffern feststellen, dass ein Datenfeld einen Kundennamen darstellt. Wenn der Parser den Altcomputerbericht dechiffriert, transformiert ein Generator die Daten des Altcomputersystems in ein in geeigneter Weise gekennzeichnetes XML-Format.

[0006] Obwohl das Endergebnis des Parsens und des Transformierungsprozesses Daten in einem XML-Format sind, ist es schwierig und aufwändig, den Prozess selbst zu implementieren, und mühsam, diesen aufrecht zu erhalten. Ohne vorsichtiges Studium der zugrunde liegenden Programmierlogik ist es im Allgemeinen nicht möglich, alle möglichen Ausgaben aus dem Altcomputersystem zuverlässig zu bestimmen. Insbesondere ist es sogar bei einer durchschnittlich großen Ausgabemenge fast sicher, dass sie unvollständig ist, weil ein Teil der Programmierlogik nur selten ausgeführt wird. Eine weitere Schwierigkeit beim Parsen und Transformationsprozess besteht darin, dass, da Änderungen an den zugrunde liegenden Programmanwendungen des Altcomputersystems vorgenommen werden, die Parser- und Transformationssysteme im Allgemeinen Aktualisierungen erforderlich machen, die die zugrunde liegenden Änderungen spiegeln. Diese nachgeordneten Änderungen erhöhen die Zeit und den Aufwand, die dem Aufrechterhalten des Altcomputersystems gewidmet sind, und erhöhen auch die Wahrscheinlichkeit von Fehlern, die in die XML-formatierte Ausgabe eingebaut werden.

[0007] Eine weitere Schwierigkeit bei der Verwendung von XML besteht darin, dass, obwohl XML die Verwendbarkeit der Ausgabedaten erheblich verbessert, die Erzeugung der XML-Ausgabe von den zugrunde liegenden Programmen abhängt, die sich an eine exakte Datenstruktur halten. Zum Beispiel erfordert die Erzeugung von syntaktisch korrektem XML das Festhalten an einer fest bezeichneten Baumstruktur, so dass Ausgabedaten durch „Tags" und „End-Tags" identifiziert werden, die der XML-Datenstruktur zugeordnet ist, wie es durch das XML-Schema definiert ist. Beim Schreiben eines tief in einem XML-Baum eingebetteten Elements, wie z. B. ein Unterschema innerhalb eines definierten XML-Schemas, müssen auch Tags, die allen Elementevorgängern des Elementes entsprechen, ebenfalls geschrieben werden. Beim Schreiben eines weiteren Elements, das nicht Teil eines aktuellen XML-Subschemas ist, muss das aktuelle Unterschema bis zu einer geeigneten Ebene durch ausgleichende abschließende End-Tags für die Vorgängerelemente abgeschlossen werden. Das XML-Schema spezifiziert auch Typ und Kardinalitätbeschränkungen ihrer Elemente. Somit ist eine substantielle und exakte Buchführung von Programmen, die XML ausgeben, mit Bezug auf das XML-Schema notwendig, um Fehler seitens von Programmierern zu minimieren.

[0008] Eine bestimmte Anwendung von XML, die akzeptiert worden ist, ist das Dokumentenobjektmodell („DOM"), das durch das World Wide Web-Konsortium („W3C") erzeugt wurde. DOM ist eine plattformneutrale und sprachneutrale Schnittstelle, die es Programmen ermöglicht, dynamisch auf Inhalt, Struktur und Stile von Dokumenten zuzugreifen und diese zu aktualisieren. Kommerzielle Pakete sind verfügbar, die DOM-Anwendungsprogrammierschnittstellen („APIs") zur Verfügung stellen, und die Werkzeuge zur Extensible Stylesheet Language („XSL") und zur XSL-Transformation („XSLT") zur Verfügung stellen, um ein XML-DOM gemäß XSL und XSLT Vorlagen zu modifizieren.

[0009] Das DOM umfasst eine Standardmenge von Verfahren zum Manipulieren von DOM-Elementen. Die Erzeugung einer DOM-Instanz, die einem XML-Schema genügt, erfordert im Allgemeinen einen Schritt-für-Schritt-Aufbau jedes Knotens in dem DOM-Baum, so dass alle Elternelemente gemeinsam mit eingebetteten Elementen eines XML-Baumes erzeugt werden. Wenn ein Element hinzugefügt wird, das nicht Teil des aktuellen Unterschemas ist, muss der DOM-Baum im Allgemeinen zu einem geeigneten Vorgängerknoten gebracht werden (traversed) mit neuen Nachfahren des Knotens, der zum Aufbauen eines korrekten Kontextes erzeugt wurde. Somit ist eine substantielle und exakte Buchführung für die DOM-Konstruktion notwendig, um Fehler seitens von Programmierern zu minimieren.

[0010] Die Druckschrift „Automated Support for Legacy Code Understanding", Jim Q. Ning et. al., veröffentlicht in Communications of the ACM, New York, US, Band 37, Nr 5, 1. Mai 1994, Seiten 50–54, ISSN: 0001-0782 offenbart ein System zum Analysieren einer Anwendung eines Altcomputersystems.

Zusammenfassung der Erfindung

[0011] Daher besteht zunehmend ein Bedarf nach einem Verfahren und einem System, das schnell und automatisch Altcomputersysteme modifiziert, um eine Ausgabe in einem XML-Format zu erzeugen.

[0012] Ein weiterer Bedarf besteht nach einem Verfahren und einem System, das Altcomputersysteme modifiziert, um eine Ausgabe in einem XML-Format zu erzeugen, ohne die zugrunde liegende Altcomputersystemprogrammlogik oder Geschäftsregeln zu verändern.

[0013] Ein weiterer Bedarf besteht nach einem Verfahren und einem System, das Schreiboperationen eines Altcomputersystems feststellt, um Modifikationen dieser Knoten zu ermöglichen, so dass das Altcomputersystem Daten in einem XML-Format ausgibt.

[0014] Ein weiterer Bedarf besteht nach einem Verfahren und einem System, das syntaktisch eine korrekte XML-Ausgabe mit automatischer Buchhaltung erzeugt, um Programmierfehler zu minimieren.

[0015] Ein weiterer Bedarf besteht nach einem Verfahren und einem System, das syntaktisch korrekte XML-Ausgabe generiert, indem ein DOM erstellt wird, um eine XML-Datenstruktur zu erzeugen, wie z. B. bei der Modifikation von Altcomputersystemcode.

[0016] Gemäß der vorliegenden Erfindung, die in den Ansprüchen 1 und 10 definiert ist, wird ein Verfahren und ein System zur Verfügung gestellt, das im Wesentlichen die Nachteile und Probleme bei zuvor entwickelten Verfahren und Systemen, die die Ausgabe eines Altcomputersystems in ein XML-Format transformiert, eliminiert oder reduziert. Die vorliegende Erfindung stellt eine XML-Ausgabe zur Verfügung, indem die zugrunde liegenden Altcomputersystemprogrammanwendungen modifiziert werden, um Daten in einem XML-Format

auszugeben, anstatt die Ausgabe von dem Altcomputersystem zu transformieren, nachdem die Daten in dem Format des Altcomputersystems ausgegeben worden sind.

[0017] Insbesondere modifiziert eine Codeerzeugungseinheit Altcomputerprogrammanwendungen, um modifizierte Altprogrammanwendungen zu erzeugen. Die modifizierten Altprogrammanwendungen werden auf dem Altcomputersystem so ausgeführt, dass die Datenausgabe von dem Altcomputersystem in dem XML-Format erfolgt. Die modifizierten Altprogrammanwendungen sind in der Computersprache des Altcomputersystems geschrieben, so dass das Altcomputersystem direkt eine XML-Version ihrer Ausgabe erzeugt, ohne die Notwendigkeit, die Logik oder Geschäftsregeln, die in den nicht modifizierten Programmanwendungen des Altcomputersystems vorhanden sind, zu verändern.

[0018] Die Codeerzeugungseinheit erzeugt die modifizierten Programmanwendungen gemäß einer Modifikationsspezifikation, die durch eine Mapping-Einheit erzeugt wird. Die Mapping-Einheit erzeugt die Modifikationsspezifikation und eine Kontexttabelle, indem ein Modell von Schreiboperationen des Altcomputersystems auf ein XML-Schema gemappt wird. Die Mapping-Einheit stellt die Modifikationsspezifikation der Codeerzeugungseinheit zur Verfügung. Die Codeerzeugungseinheit erzeugt modifizierte Altcomputersytemprogrammanwendungen zur Verwendung in dem Altcomputersystem. Eine Schreibereinheit ist ein Anwendungsprogramm, das auf das Altcomputersystem geladen wird und in der Sprache des Altcomputersystems geschrieben ist. Die Schreibereinheit wird durch die modifizierten Programmanwendungen aufgerufen, um eine XML-Ausgabe in dem Format des XML-Schemas, das durch die Kontexttabelle codiert wird, zu schreiben.

[0019] Das durch die Mapping-Einheit verwendete Modell wird durch eine Modellierungseinheit erzeugt, die das Altcomputersystem analysiert, um die Schreiboperationen zu identifizieren und zu modellieren, wie z. B. mit einem Berichtsdatenmodell. Die Modellierungseinheit bestimmt eine Liste von Altcomputersystemprogrammanwendungen, die Daten ausgeben. Die Programmanwendungen, die Daten ausgeben, werden weiterhin analysiert, um die Ereignisse in jeder Programmanwendung zu bestimmen, an denen eine Schreiboperation vorkommt. Ein Ausgabedatenmodell wird dann mit einem Wert und/oder Typ für die Datenfelder jedes Ereignisses kompiliert. Das Ausgangsdatenmodell ist um eine formale Grammatik erweitert, die den Prozess des Beziehen von Schreiboperationen zu Ausführungspfaden von Altcomputersystemprogrammanwendungen vereinfacht.

[0020] Ist die modifizierte Programmanwendung auf das Altcomputersystem geladen, fährt das Altcomputersystem damit fort, seinen funktionalen Betrieb ohne Änderung der zugrunde liegenden Geschäfts- oder Programmlogik zu ändern. Wenn eine Programmanwendung des Altcomputersystems die Ausgabe von Daten anweist, rufen modifizierte Befehle, die in der modifizierten Programmanwendung vorgesehen sind, die Schreibereinheit auf, um syntaktisch korrekte XML-Daten auszugeben. Die Schreibereinheit bestimmt den aktuellen Kontext der XML-Ausgabe und öffnet geeignete Schema-Element-Datenstrukturen in Verbindung mit der Kontexttabelle. Die Schreibereinheit analysiert dann die aktuelle Schema-Element-Datenstruktur und das aufgerufene Schema-Element, um die Beziehung des aufgerufenen Schema-Elements zu dem aktuellen Schema-Element zu bestimmen. Wenn das aufgerufene Schema-Element ein Nachfolger des aktuellen Schema-Elements ist, öffnet die Schreibereinheit die ID-Tags des Schema-Elements durch das aufgerufene Schema-Element und gibt die Daten von dem Schema-Element in einem syntaktisch korrekten XML-Format aus. Wenn das Schema-Element kein Nachfolger des aktuellen Schema-Elements ist, findet die Schreibereinheit einen gemeinsamen Vorgänger mit einer konsistenten Kardinalität, schließt die ID-Tags des Schema-Elements bis zu dem Vorgänger-Schema-Element und fährt damit fort, die ID-Tags des Schema-Elements durch das aufgerufene Schema-Element zu öffnen, um Daten in einem syntaktisch korrekten XML-Format auszugeben. Zusätzlich unterstützt die Schreibereinheit ein verzögertes Drucken von Tags und Attributen bis zu einer Zeit, zu der eine vollständige syntaktische Einheit verfügbar ist.

[0021] Bei einer Ausführungsform wird ein Ziel-DOM erstellt und XML ausgegeben, wenn der Aufbau des gesamten Ziel-DOMs vollständig ist. Ein API schreibt XML durch Erzeugen einer Zwischeninstanz des DOM und gibt dann direkt von dem DOM mit der möglichen Anwendung einer Stylesheet-Transformation aus. Der API puffert XML-Daten in willkürlicher Anzahl von Kontexten, die gleichzeitig aktiv sind, so dass jeder Aufruf der API auf irgendeinem Knoten der DOM-Struktur betrieben werden kann. Durch Aufbau der gesamten DOM-Instanz vor der Ausgabe von irgendeiner XML kann die API einen Knoten der DOM-Instanz manipulieren, der willkürlich weit zurück in einer Abfolge von API-Aufrufen erzeugt wird. Zusätzlich kann eine DOM-Instanz durch die Anwendung eines XSLT-Stylesheets restrukturiert werden, um eine bestimmte XML-Schema-Datenstruktur auszugeben.

[0022] Wenn insbesondere eine Programmanwendung eines Altcomputersystems das Ausgeben von Daten

anweist, rufen modifizierte Befehle, die in der modifizierten Programmanwendung vorgesehen sind, die Schreibereinheit auf, um ein DOM-Objekt mit strukturell korrekten XML-Daten zu versehen. Die Schreibereinheit verwendet entweder den aktuellen Kontext des XML-DOM oder einen weiteren Kontext, der als ein Argument des API-Aufrufs bereitgestellt wird, und öffnet geeignete Schema-Element-Datenstrukturen in Verbindung mit der Kontexttabelle. Die Schreibereinheit analysiert die Datenstruktur des aktuellen Schema-Elements und das aufgerufene Schema-Element, um die Beziehung des aufgerufenen Schema-Elements zu dem aktuellen Schema-Element zu bestimmen. Wenn das aufgerufene Schema-Element ein Nachfolger des aktuellen Schema-Elements ist, fügt die Schreibereinheit die Schema-Elementknoten durch das aufgerufene Schema-Element ein und konstruiert den Elementknoten mit den Daten von dem Schema-Element. Wenn das Schema-Element kein Nachfolger des aktuellen Schema-Elements ist, findet die Schreibereinheit den kleinsten gemeinsamen Vorgänger mit einer konsistenten Kardinalität, verfahrt die Schema-Element-Knoten nach oben zu dem Vorgänger-Schema-Element und fährt fort, die Schema-Element-Knoten bis hinab durch das aufgerufene Schema-Element einzufügen, um den Element-Knoten aufzubauen. Zusätzlich unterstützt die Schreibereinheit die Erfassung von Attributen und ihren Werten.

[0023] Die vorliegende Erfindung stellt eine Anzahl von wichtigen technischen Vorteilen zur Verfügung. Ein wichtiger technischer Vorteil besteht in der Fähigkeit, schnell und automatisch Programmanwendungen eines Altcomputersystems zu modifizieren, um es ihnen zu ermöglichen, direkt eine XML-Version ihrer Datenausgabe zu erzeugen. Indem die zugrunde liegenden Programmanwendungen des Altcomputersystems modifiziert werden, wird eine XML-Ausgabe direkt aus dem Altcomputersystem ohne eine Transformation der Daten selbst von einem Altcomputersystemformat verfügbar gemacht. Weiterhin bleibt die zugrunde liegende Programmlogik und Geschäftsregeln unverändert, so dass die wesentlichen Funktionen des Altcomputersystems nicht verändert werden müssen. Somit wird einem Geschäftsunternehmen, das ein Altcomputersystem verwendet, eine größere Zugreifbarkeit auf Daten zur Verfügung gestellt, die durch Ausgabe in einem XML-Format bereitgestellt werden, ohne dass berechnete Werte beeinträchtigt werden.

[0024] Ein weiterer wichtiger technischer Vorteil der vorliegenden Erfindung besteht darin, dass eine Modifikation der zugrunde liegenden Altcomputerprogrammanwendungen tatsächlich weniger teuer, komplex und zeitaufwendig ist, als die Transformation einer Ausgabe eines Altcomputersystems in einem XML-Format. Zum Beispiel ist eine XML-formatierte Ausgabe ohne weiteres Zutun zu den Daten verfügbar, wenn die modifizierten Programmanwendungen auf dem Altcomputersystem ausgeführt werden. Zum Vergleich erfordert eine Transformation einer Ausgabe in einem XML-Format nachdem die Daten durch das Altcomputersystem ausgegeben wurden, eine Maßnahme mit jeder Datenausgabe. Wenn somit Änderungen an der zugrunde liegenden Altprogrammanwendungen vorgenommen werden, müssen die Änderungen auch im Allgemeinen auf die Transformationsanwendungen vorgenommen werden, die die zugrunde liegenden Änderungen widerspiegeln. Dies verkompliziert weiterhin die Instandhaltung des Altcomputersystems.

[0025] Ein weiterer wichtiger technischer Vorteil der vorliegenden Erfindung besteht darin, dass ungeachtet, ob die Schreibereinheit und die Kontexttabelle mit einem Altcomputersystem verwendet werden, diese bei der Erzeugung einer syntaktisch korrekten XML-Ausgabe helfen. Zum Beispiel stellt die Schreibereinheit sicher, dass ein Befehl zum Schreiben eines eingebetteten XML-Elements, Tags umfasst, die allen eingebetteten Vorgängerelementen des Elements entsprechen. Auch wenn ein XML-Element geschrieben wird, das nicht Teil des aktuellen XML-Unterschemas ist, schließt die Schreibereinheit das aktuelle XML-Unterschema von einer geeigneten Ebene eines geeigneten Vorgänger-Schema-Elements ab. Die Automatisierung der mit dem XML-Schema befassten Buchführung eliminiert das Risiko von syntaktischen Fehlern, die bei XML-Ausgaben auftreten. Das Merkmal des verzögerten Druckens stellt einen Mechanismus zur Verfügung, wodurch ein Programm korrekte XML-Daten erzeugen kann, sogar wenn die Abfolge der Druckbefehle in dem ursprünglichen Altsystemanwendungsprogramm nicht direkt mit der Ordnung der XML-Elemente, die durch das XML-Schema vorbeschrieben sind, übereinstimmt.

[0026] Ein weiterer wichtiger Vorteil der vorliegenden Erfindung besteht darin, dass eine Unterstützung durch Werkzeuge die Komplexität des Modellierens der zugrunde liegenden Programmlogik managt, was in einer wesentlich reduzierten Zeit und Aufwand für die Modifikation eines Altcomputersystems zum Ausgeben von XML-formatierten Daten führt. Werkzeuge helfen bei: der Bestimmung des Steuerflussgraphen der Altanwendungen; der Abstraktion eines Untergraphen, der insbesondere auf das Schreiben von Ausgabezeilen bezogen ist, aus diesem Graphen; der Identifikation von Konstanten und Datenelementen, die in Druckzeilen eingehen, so dass die Elemente, die als gekennzeichnete XML geschrieben werden sollen, vollständig identifiziert werden können; und der Identifikation von domänenspezifischer Information, wie z. B. Positionen von Kopfabschnitten und Fußabschnitten. Die Automatisierung durch die Unterstützung durch Werkzeuge erhöht das Management der Programmkomplexität erheblich.

[0027] Ein weiterer wichtiger technischer Vorteil der vorliegenden Erfindung wird durch die automatische Erzeugung von Datenstrukturen aus dem XML-Schema und der Erzeugung von kontextsensitiven DOM zur Verfügung gestellt. Dies führt z. B. zu einer schnelleren Entwicklung eines neuen Codes und einer schnelleren Revision für bestehenden Altcode, um XML-Daten auszugeben. Weiterhin wird die Gelegenheit für Fehler aufgrund des automatischen Festhaltens an die Anforderungen des XML-Schemas vermindert. Auch ist die In-situ-Erzeugung von XML aus einem Altcomputersystem erleichtert, so dass die Ausgabe eines Zielschemas sogar dann ermöglicht wird, wenn sich dieses erheblich von dem der natürlichen Struktur der Ausgabe, die durch ein zugrunde liegendes Altcomputersystem vorgenommen wird, unterscheidet.

Kurzbeschreibung der Zeichnungen:

[0028] Ein umfassenderes Verständnis der vorliegenden Erfindung und deren Vorteile wird durch Bezugnahme auf die nachfolgende Beschreibung in Verbindung mit den beigefügten Zeichnungen erhalten, in denen gleiche Bezugszeichen gleiche Merkmale angeben, und in denen:

[0029] Fig. 1 ein Blockdiagramm eines Codeerzeugungssystems darstellt, das mit einem Altcomputersystem in Verbindung steht;

[0030] Fig. 2 ein Flussdiagramm der Erzeugung von modifizierten Altprogrammanwendungen angibt, um XML-Daten auszugeben;

[0031] Fig. 3 ein Flussdiagramm der Erzeugung eines Modells von Schreiboperationen einer Altprogrammanwendung angibt;

[0032] Fig. 4 eine Beispielausgabe eines Altcomputersystemberichts für eine Telefonrechnung angibt;

[0033] Fig. 5 XML-formatierte Daten, die dem in Fig. 4 angegebenen Altcomputersystembericht entsprechen, angibt;

[0034] Fig. 5A ein XML-Schema für die in Fig. 5 angegebene Ausgabe angibt;

[0035] Fig. 6 eine grafische Benutzerschnittstelle zum Mappen eines Altcomputersystemcodes in ein Extensible Mark-up Language Schema und ein Berichtsdatenmodell angibt;

[0036] Fig. 6A ein zugrunde liegender COBOL Code angibt, der durch das Berichtsdatenmodell der Fig. 6 modelliert wird;

[0037] Fig. 7 ein beispielhaftes Extensible Mark-up Language Schema zum Ausgeben von Adressdaten angibt;

[0038] Fig. 7A eine Baumstruktur für das Schema der Fig. 7 angibt;

[0039] Fig. 7B eine berechnete Datenkontexttabelle für das Schema, das in Fig. 7 angegeben ist, darstellt; und

[0040] Fig. 8 ein Flussdiagramm einer XML-Druckoperation darstellt, die die Erzeugung einer syntaktisch korrekten Ausgabe von Extensible Mark-up Language Daten sicherstellt.

[0041] Fig. 9 stellt ein Flussdiagramm einer XML-Druckoperation dar, die die Erzeugung einer syntaktisch korrekten Ausgabe von Extensible Mark-up Language Daten sicherstellt, indem diese als eine DOM-Instanz gepuffert werden.

Ausführliche Beschreibung der Erfindung

[0042] Bevorzugte Ausführungsformen der Erfindung werden in den Figuren dargestellt, wobei gleiche Bezugszeichen verwendet werden, um auf gleiche oder entsprechende Teile der verschiedenen Zeichnungen Bezug zu nehmen.

[0043] Um einen Vorteil aus den Chancen, die sich aus der Nutzung von XML als ein Medium für E-Commerc ergeben, zu ziehen, müssen Geschäfte entweder ihre bestehenden Altcomputersysteme ersetzen oder die An-

wendungen auf die Altcomputersysteme umschreiben. Jedoch haben Geschäfte erhebliche Investitionen in ihre bestehenden Altcomputersysteme vorgenommen und die darauf angepassten Anwendungen, so dass eine gesamte Ersetzung dieser Systeme und Anwendungen kurzfristig nicht praktikabel ist. Altcomputersysteme führen wesentliche Funktionen, wie z. B. Rechnungsstellung, Bestandssteuerung und Zeitplanung aus, die eine massive Verarbeitung von Online- und Los-Transaktionen benötigen. Altcomputersystemanwendungen, die in Sprachen wie z. B. COBOL geschrieben sind, bleiben für die absehbare Zukunft ein lebender Bestandteil von Unternehmensanwendungen vieler großer Organisationen. Tatsächlich stellt diese installierte Basis existierender Software die Hauptverkörperung von Geschäftsregeln vieler Organisationen dar. Obwohl diese Anwendungen im Prinzip von Hand modifiziert werden können, um Daten in einem XML-Format auszugeben, kann in der Realität die zugrunde liegende Logik sogar einer einfachen Berichtsausgabeanwendung schwierig zu verstehen und zu dechiffrieren sein.

[0044] Daher besteht eine erhebliche Herausforderung, denen viele Geschäfte gegenüberstehen, darin, die schnelle und kostengünstige Anpassung von bestehenden Computersystemen vorzunehmen, um von den durch den elektronischen Handel bestehenden Chancen in vorteilhafter Weise zu nutzen. Sogar wenn neue und aktualisierte Computersysteme installiert werden, erfordert die sich ständig verändernde Natur des elektronischen Handels, dass Geschäfte Flexibilität als eine Schlüsselkomponente für neue Computersysteme beinhalten. XML ist aufgrund der Einfachheit, mit der XML sich an essentielle E-Commerce-Funktionen anpasst, wie z. B. die Übertragung über das Internet, die Direktübertragung als ein Objekt zwischen verschiedenen Anwendungen und die Anzeige und Manipulation über eine Browsertechnologie zu einer populären Wahl für die Berichtsdaten geworden. Die Flexibilität von XML ist ein Ergebnis davon, dass sie benannte Tags umklammernde Daten, die die Beziehung der Daten innerhalb eines XML-Schemas identifizieren, einschließen. Jedoch beruht die Implementierung von XML-Datenberichten auf der akkuraten Verwendung der Tags, um die Ausgangsdaten innerhalb des XML-Schemas zu definieren. Somit halten sich Computersysteme, die XML implementieren, an das XML-Schema und verwenden eine exakte Buchführung, um genaue Berichte zu erhalten.

[0045] Die vorliegende Erfindung hilft bei der Implementierung von XML für Berichte sowohl durch die Modifikation von Altcomputersystemprogrammanwendungen, um XML-Daten auszugeben als auch durch das Nachverfolgen von XML-Ausgaben innerhalb eines XML-Schemas, um eine akkurate Ausgabe sicherzustellen, ungeachtet, ob die XML-Daten aus einem Altcomputersystem stammen oder nicht. Mit Bezug auf Fig. 1 zeigt ein Blockdiagramm ein Computersystem **10**, das ein Altcomputersystem **12** modifiziert, um Daten in einem XML-Format auszugeben. Ein Codeerzeugungssystem **14** stellt eine Schnittstelle zu dem Altcomputersystem **12** dar, um die Analyse von einer oder mehreren Altprogrammanwendungen **16** und die Erzeugung von einem oder mehreren modifizierten Altprogrammanwendungen **18** zu ermöglichen. Das Codeerzeugungssystem **14** stellt dem Altcomputersystem **12** auch eine Schreibereinheit **20** und eine Kontexttabelle **22** zur Verfügung. Das Altcomputersystem **12** ist dann in der Lage, direkt XML-formatierte Daten auszugeben, wenn die modifizierten Altprogrammanwendungen **18** die Schreibereinheit **20** in Verbindung mit der Kontexttabelle **22** aufrufen, um syntaktisch korrekte XML-Daten auszugeben.

[0046] Das Codeerzeugungssystem **14** umfasst eine Codeerzeugungseinheit **24**, eine Mapping-Einheit **26** und eine Modellierungseinheit **28**. Die Modellierungseinheit **28** stellt eine Schnittstelle zu dem Altcomputersystem **12** her, um eine Kopie der Altprogrammanwendungen **16** zur automatischen Durchsicht und Modellierung zu erhalten. Die Modellierungseinheit **28** erzeugt eine Liste von Ereignissen für Punkte in dem Programm, an denen Daten geschrieben werden. Zum Beispiel kann die Modellierungseinheit **28** den Quellcode der Altprogrammanwendung nach Berichts- oder Schreibbefehlen für ausgewählte Ausgangsdatenströme durchsuchen. Die Listen von Berichtsereignissen werden verwendet, um die Berichtsfunktionen des Altcomputersystem z. B. durch ein Berichtsdatenmodell, das die Werte und Typen von beschriebenen Datenfeldern aus den Altprogrammanwendungen **16** auflistet, zu modellieren. Die Liste der Berichtsereignisse wird dann durch eine formale Grammatik ergänzt, die verwendet wird, um das XML-Schema auf die durch die Altprogrammanwendungen als Bericht erstellte Ausgabe zu beziehen. Die Liste der Berichtsausgabeereignisse und die formale Grammatik sind zwei Komponenten des Berichtsdatenmodells für das Altsystemanwendungsprogramm. Intuitiv beschreibt ein Ereignis eine Zeile in einem Bericht und die formale Grammatik beschreibt, wie das Anwendungsprogramm diese Zeilen durchläuft, um einen Bericht zu bilden.

[0047] Die Modellierungseinheit **28** stellt ein Berichtsdatenmodell, das Berichtsausgabeereignisse in den Altprogrammanwendungen **16** identifiziert, der Mapping-Einheit **26** und der Modellierungs-/Mapping grafischen Benutzerschnittstelle **30** zur Verfügung. Die Mapping-Einheit **26** bringt die Berichtsausgabeereignisse aus dem Berichtsdatenmodell zu dem XML-Schema **32** in Übereinstimmung, und diese Beziehung zwischen dem Berichtsdatenmodell und dem XML-Schema **32** wird auf der grafischen Modellierungs-/Mapping Benutzerschnitt-

stelle **30** angezeigt. Durch Herstellung der Beziehung den Berichtsausgabeereignissen der Altprogrammanwendung **16** und dem XML-Schema **32** definiert die Mapping-Einheit **26** eine Spezifikation für die Modifikation der Altprogrammanwendungen **16** zu den ausgegebenen XML-Daten. Die grafische Modellierungs-/Mapping Benutzerschnittstelle **30** stellt eine Information für Programmierer der Modifikationsspezifikation zur Vefügung. Die grafische Modellierungs-/Mapping Benutzerschnittstelle **30** erzeugt eine Modifikationsspezifikation und eine Kontexttabelle **22**. Optional ermöglicht die grafische Modellierungs-/Mapping Benutzerschnittstelle **30** Programmierern, ein XML-Schema zu erzeugen oder zu modifizieren.

[0048] Die Codeerzeugungseinheit **24** akzeptiert die Modifikationsspezifikation, eine Kopie der Altprogrammanwendungen **16** und eine Kontexttabelle, um modifizierte Altprogrammanwendungen **18** zu erzeugen. Anhand der Modifikationsspezifikation erzeugt die Codeerzeugungseinheit **24** einen Quellcode in der Computersprache des Altcomputersystems, der in die Altprogrammanwendungen **16** eingefügt wird, um die Ausgabe von XML-Daten anzuweisen, und speichert den modifizierten Quellcode als die modifizierten Altprogrammanwendungen **18**. Die modifizierten Altprogrammanwendungen **18** können weiterhin die Altcomputersystemberichtsausgabebefehle beibehalten, so dass die modifizierten Programmanwendungen **18** zusätzlich zu dem XML-Format weiterhin Daten in dem Format des Altcomputersystems als Bericht ausgeben. Die Ausgabe beider Formate hilft bei der Qualitätskontrolle, indem ein direkter Vergleich der Daten aus dem modifizierten und nicht modifizierten Code ermöglicht wird. Alternativ können die modifizierten Befehle, die durch die Codeerzeugungseinheit **24** bereitgestellt werden, Berichtsbefehle von Altprogrammanwendungen **16** ersetzen, so dass die modifizierten Altprogrammanwendungen Daten ausschließlich in dem XML-Format berichten. Die Schreibereinheit **20** ist in einer Computersprache des Altcomputersystems **12** geschrieben und bezieht sich auf die Kontexttabelle **22**, um die geeigneten XML-Schema-Elemente für die Ausgabe von Daten aus dem Altsystem **12** zu bestimmen. Der modifizierte Code in den modifizierten Altprogrammanwendungen **18** ruft die Schreibereinheit **20** auf, wenn Daten in dem XML-Format ausgegeben werden sollen.

[0049] Mit Bezug auf Fig. 2 stellt ein vereinfachtes Flussdiagramm den Prozess der Erzeugung von modifizierten Altprogrammanwendungen dar, die Daten in einem XML-Format ausgeben. Der Prozess beginnt mit Schritt **34**, in dem der Altcode der Altprogrammanwendungen **16** dem Codeerzeugungssystem **14** verfügbar gemacht wird. Zum Beispiel lädt ein Mainframe Altcomputersystem, das einen COBOL Quellcode ausführt, eine Kopie des Quellcodes in ein Codeerzeugungssystem **14** zur Analyse und Erzeugung eines modifizierten Codes herunter.

[0050] In Schritt **36** modelliert das Codeerzeugungssystem **14** die Altprogrammanwendungen, um ein Berichtsdatenmodell der Schreibereignisse und ihrer zugrunde liegenden Grammatik aus dem Code der Altprogrammanwendungen bereitzustellen. Zum Beispiel identifiziert das Berichtsdatenmodell die Ereignisse innerhalb des Codes der Altprogrammanwendungen **16**, zu denen Daten auf ausgewählte Ausgangsgeräte geschrieben werden, einschließlich der Werte und Typen der Daten. In Schritt **38** wird das Berichtsdatenmodell verwendet, um eine Modifikationsspezifikation zu erzeugen. Die Modifikationsspezifikation wird in Verbindung mit einem XML-Schema erzeugt, das in Schritt **40** bereitgestellt wird, der die Datenstruktur für Schreibbefehle der modifizierten Altprogrammanwendungen **18** definiert, um XML-Daten auszugeben.

[0051] In Schritt **42** werden die Modifikationsspezifikationen verwendet, um automatisch modifizierten Altcode zu erzeugen, der auf dem Altcomputersystem **12** ausgeführt werden soll. Der modifizierte Altcode wird in Schritt **44** so ausgeführt, dass die modifizierten Altprogrammanwendungen eine Ausgabe von dem Altsystem **12** in einem XML-Format ausgeben, ohne zuvor eine Transformation der Ausgangsdaten zu benötigen.

[0052] Der Prozess des Modellierens des Altcomputersystems **12** wird ausführlicher mit Bezug auf Fig. 3 gezeigt. Die Modellierungseinheit **28** extrahiert ein Berichtsdatenmodell der Altprogrammanwendungen **16** durch eine automatische Analyse des zugrunde liegenden Altcodes. Die automatische Analyse stellt ein verbessertes Verständnis der Operation des Altcodes dar und reduziert die Wahrscheinlichkeit von Fehlern hinsichtlich der Operation und der Aufrechterhaltung des zugrunde liegenden Altcodes. Im Wesentlichen analysiert die Modellierungseinheit **28** den Altsoftwareprozess nach Regeln, um ihren Steuerfluss aufzuzeichnen. Eine Abstraktion des Steuerflusses erzeugt ein Berichtsdatenmodell, der das Verständnis von Datentypen und invarianten Datenwerten, die über jeden Schreibbefehl in dem Berichtsdatenmodell geschrieben werden, erlaubt. Das Berichtsdatenmodell stellt ein Modell der Altprogrammanwendungen **16** zur Verfügung, wenn dieses mit den Werten und den Typen der geschriebenen Datenfelder kombiniert wird.

[0053] Mit Bezug auf Fig. 3 startet der Modellierungsprozess in Schritt **46** durch die Bestimmung des Steuerflussgraphen des Altprogramms. Der Steuerflussgraph einer bestimmten Altprogrammanwendung ist ein gerichteter Graph (N, A) in dem N ein Knoten für jeden Ausführungspunkt der Programmanwendung erhält und

A einen arc <n_1, n_2> enthält, wobei n_1 und n_2 Elemente von N sind, wenn die Altprogrammanwendung in der Lage ist, bei einem möglichen Ausführungszustand sofort von n_1 nach n_2 zu springen.

[0054] In Schritt **48** werden die Schreiboperationen des Steuerflussgraphen bestimmt, um einen Steuergraphen für eine Datendatei zu erhalten. Im Wesentlichen wird der Steuerflussgraph ausschließlich auf Startknoten, Stoppknoten und Knoten, die ausgewählte Datendateien schreiben, abstrahiert. Dies führt zu einem Steuergraphen für eine Datendatei, die die Schreibereignisse in den Altprogrammanwendungen identifiziert. Der Steuergraph für die Datendatei, die aus einem Steuerflussgraphen (N, A) abstrahiert wird, ist ein gerichteter Graph (N_R, A_R). Ein Knoten n wird in der Gruppe der Knoten N_R festgelegt, wenn der Knoten n eine Altprogrammanwendung startet, eine Altprogrammanwendung anhält, oder auf eine Datendatei schreibt. Der arc <n_1, n_m> ist in A_R, wenn sowohl n_1 als auch n_m in der Gruppe von Knoten N_R sind und eine Abfolge von arcs <n_1, n_2>, <n, n_3> ... <n_{m-1}, n_m> in A existiert, wobei für i von 2 zu m – 1, n_i nicht in der Gruppe der Knoten N_R enthalten ist.

[0055] Wenn der Steuergraph für die Datendatei in Schritt **50** vervollständigt ist, wird eine Information über die in jeden Schreibknoten der Datendatei geschriebenen Daten dem Steuergraph für die Datendatei angehängt. Zum Beispiel werden die Werte oder der Typ jedes Datenfelds, das durch jeden Knoten geschrieben wird, statisch über den Datenfluss in dem Steuerflussgraphen bestimmt, und dem Knoten des Steuergraphen der Datendatei hinzugefügt.

[0056] In Schritt **52** werden die Pfade von dem Startknoten durch den Steuergraphen für die Datendatei zu dem Stoppknoten in einer formalen Grammatik dargestellt. Diese formale Grammatik mit den angehängten Datenfeldinformationen bildet das Berichtsdatenmodell. Dieses Modell ist eine abstrakte Darstellung der Datendateien, die durch die Altprogrammanwendungen geschrieben werden können, und stellt die Basis zur Verfügung, auf der eine Modifikationsspezifikation geschrieben werden kann.

[0057] Das Berichtsdatenmodell wird in zwei Teilen bereitgestellt. Zunächst wird jeder Schreibknoten mit seiner angehängten Datenfeldinformation als ein Ereignis dargestellt. Diese Ereignisse sind die Grundlegendsten oder Blattunterausdrücke des Berichtsdatenmodells. Zweitens werden die Nicht-Blattunterausdrücke des Berichtsdatenmodells als Regel dargestellt, die sich hierarchisch von den Ereignissen aufbauen.

[0058] Die Erzeugung und Darstellung eines Berichtsdatenmodells von Altprogrammanwendungen kann durch Betrachtung eines Telefonrechnungsbeispiels dargestellt werden. **Fig. 4** stellt die gedruckte Ausgabe aus einem COBOL Programm für eine Telefonrechnung dar. Ein typisches COBOL Programm druckt die Telefonrechnung in einem vorbestimmten Format, das z. B. eine vorbestimmte Papiergröße und Spaltenabmessungen aufweisen kann. Das Drucken der „TOTAL CALLS" Zeile in **Fig. 4** ist das Ergebnis einer Berechnung der gesamten Anzahl von Anrufen, der Gesamtzeit der Anrufe und der Gesamtkosten der Anrufe. Als ein Beispiel eines einzelnen Knotens eines Steuerflussgraphen lautet das Ereignis, das von dem COBOL Code zum Ausgeben der Total Calls-Zeile der **Fig. 4** abgeleitet ist, wie folgt:

Incident 47 loc 414 record PRTEC from RS-LINE

< LINE 2 >

0: "TOTAL CALLS:"

14: RECORDS-SELECTED-EDIT loc266 pic Z, ZZ9 size 5

19: " TOTAL TIME:

53: RS-HH loc 270 pic 99 size 2

55: ":"

56: RS-MM loc 272 pic 99 size 2

58: ":"

59: RS-SS loc 274 pic 99 size 2

61: " "

63: RS-COST loc 276 pic $$$$$.99 size 8

71: " "

[0059] Das Ereignis 47 beschreibt die Daten, die in dem geeigneten Punkt in dem Programm durch die Schreibinstruktion in Zeile 414 geschrieben werden. Die Daten umfassen die Kopfabschnitte von „TOTAL CALLS" und „TOTAL TIME", auf die die akkumulierten Werte für die Gesamtzahl von Anrufen, die Gesamtzeit der Anrufe und die Gesamtkosten der Anrufe folgen. Die konstanten Werte „TOTAL CALLS" und „TOTAL TIME" werden durch die Datenflussanalyse des Altanwendungsprogramms bestimmt.

[0060] Das Berichtsdatenmodell umfasst Grammatikregeln, die aus den Schreibereignissen aufgebaut werden. Wenn jede Grammatikregel aus den geeigneten Ereignissen und Unterregeln definiert ist, wird eine Berichtsausgabegrammatik, die die potentielle Ausgabe der Altprogrammanwendungen für die Rechnung, die in Fig. 4 gezeigt ist, beschreibt, wie folgt erzeugt:

Regel 23 [seq 3 4 5 6 7 8 9 10]

Regel 24 [? 23]

Regel 41 [seq 23 24 25]

Regel 42 [? 41]

Regel 45 [seq 0 1 2 42]

Regel 46 [? 45]

Regel 50 [seq 2449]

Regel 51 [? 50]

Regel 61 [seq 24 47 48 51 23]

Regel 62 [? 61]

Regel 63 [seq 62 2425]

Regel 64 [*63]

Regel 78 [seq 46 64 24 47 48 50 65 66]

Wurzel 79 [seq 78]

[0061] Diese Grammatikregeln zeigen, wie die Schreibereignisse kombiniert werden, um die durch das Altanwendungsprogramm geschriebene Ausgabe darzustellen. Zum Beispiel besteht die Regel 61 aus der Abfolge von Unterregeln und Ereignissen 24, 47, 48, 51 und 23. Auf die Daten, die durch jede Unterregel oder Ereignis beschrieben werden, folgen nacheinander in der Datendatei die Daten, die durch die nächste Unterregel oder Ereignis beschrieben werden. Das heißt, in Regel 61 folgen auf Daten, die durch Ereignis 47 beschrieben werden, unmittelbar die Daten, die durch Ereignis 48 beschrieben werden. Die Regel 62 ist eine bedingte Regel, die angibt, dass Daten, die durch 61 beschrieben werden, in die Datendatei geschrieben werden können oder vollständig fallen gelassen werden können. Regel 64 ist eine Wiederholungsregel, die angibt, dass Daten vorhanden sind, die durch Regel 63 beschrieben werden, die Null oder mehrere Male wiederholt wird.

[0062] Mit Bezug auf Fig. 5 sind Daten, die gemäß dem XML-Schema der Fig. 5A formatiert sind, dargestellt, die eine Datenstruktur für die Altcomputerausgabe der Fig. 4 zur Verfügung stellt. Die Daten liegen innerhalb eines Öffnungs-Tags von „<bill>" und einem Abschluss-Tag von „</bill>". Das „bill"-Schema umfasst ein „detail-list" Unterschema, das wiederum ein „detail-by-phone" Unterschema umfasst. Innerhalb des „detail-by-phone" Unterschemas werden separate Tags definiert, die die Daten der TOTAL CALLS Zeile der Fig. 4 als Bericht ausgeben. Das „total-bill-by-phone" Unterschema, das „total-time-by-phone" Unterschema und das „total-calls" Unterschema definieren die Daten, die in der TOTAL CALLS-Zeile der Ausgabe des Altcomputersystems gedruckt werden.

[0063] Fig. 5A stellt das XML-Rechnungsschema dar, das verwendet wird, um die Daten in Fig. 5 auszugeben. Das Wurzelelement des Schemas entspricht dem Elemententyp mit dem Namen „bill". Ihre Unterschemata sind Typen der Unterelemente. Das detail-by-phone Unterschema des detail-list Unterschemas von „bill" umfasst die Datenstruktur, die in der TOTAL CALLS Zeile der Fig. 4 als Bericht ausgegeben wird.

[0064] Mit Bezug zu Fig. 6 stellt ein Beispiel einer Anzeige durch die grafische Modellierungs-/Mapping Be-

nutzerschnittstelle **30** die Mapping-Beziehung zwischen dem XML-Schema, dem Berichtsdatenmodell und den zugrunde liegenden Altcomputerprogrammanwendungen, die als COBOL Code in Fig. 6A dargestellt sind, dar. Ein Grammatikfenster **54** listet die Grammatikregeln des Berichtsdatenmodells, die in dem Berichtsdatenmodell der Altprogrammanwendungen bereitgestellt werden, auf. Ein XML-Schema-Fenster **56** stellt das XML-Schema dar, das durch Fig. 5 dargestellt wird, das für die in Fig. 4 gezeigte Altcomputersystemausgabe repräsentativ ist. Ein Mapping-Fenster **58** stellt die Beziehung zwischen den Variablen der Altprogrammanwendungen und der XML-Tags des XML-Schemas dar. Zum Beispiel entspricht RS-TIME einer COBOL-Variablen, die auf das „total-time"-Tag des XML-Schemas gemappt ist, d.h. diesem zugeordnet ist. Die Regel 79 stellt die Wurzel oder den Beginn der Grammatik dar, die durch das oben gezeigte Berichtsdatenmodell bereitgestellt wird. Innerhalb des Grammatikfensters fällt Ereignis 47 unter Regel 78 als ein Ereignis, das aufgerufen wird, um die gesamten Kosten aus der Altprogrammanwendung zu berichten.

[0065] Wenn eine Beziehung zwischen dem Berichtsdatenmodell und dem XML-Schema hergestellt ist, wird eine Modifikationsspezifikation geschrieben und die Erzeugung der modifizierten Altprogrammanwendung wird automatisch durchgeführt. Die modifizierten Altprogrammanwendungen werden ausgebildet, um Daten aus dem Altcomputersystem mit XML-Schema-Tags, die die Natur der Daten beschreiben, zu berichten. Zum Beispiel entspricht im Folgenden das Ereignis 47 mit einer XML-Tag-Information und einem Datenfeldtyp und einer Werteinformation, die in diesem angegeben ist:

```
Incident 47 loc 414 record PRTEC from RS-LINE
                < LINE 2 >
0: "TOTAL CALLS :" size 14
14: RECORDS-SELECTED-EDIT loc 266 pic Z, ZZ9 size 5
                tag total-calls-by-phone
                id bill\detail-list\detail-by-phone\total- calls-by-phone
                type TAG when P
19: "TOTAL TIME :" size 34
53: RS-TIME loc 270 pic 99 size 2
                tag total-time-by-phone
                id bill\total-time
                type TAG when P
55: ":"
56: RS-MM loc 272 pic 99 size 2
58: ":" size 1
59: RS-SS loc 274 pic 99 size 2
61: ":" size 2
63: RS-COST loc 276 pic $$$$$.99 size 8
                tag total-cost
                id bill\total-cost
                type TAG when P
71: "" size 2
```

[0066] Die angegebenen Ereignisse stellen die Basis für die Modifikationsspezifikation zur Verfügung, die durch die Mapping-Einheit **26** bereitgestellt wird, um die Erzeugungseinheit **24** zur Erzeugung der modifizierten Altprogrammanwendungen zu codieren. Zum Beispiel entspricht die Modifikationsspezifikation für Ereignis 47:

node (414, XML-TOTAL-CALLS-ID, 'total-calls-by-phone', 'RECORDS-SELECTED-EDIT', 266).
node (414, XML-TOTAL-TIME-ID, 'total-time-by-phone', 'RS-TIME', 270).
node (414, XML-TOTAL-BILL-ID, 'total-bill-by-phone', 'RS-COST', 276)

[0067] Es wird angemerkt, dass die Daten RS-MM, RS-MM und RS-SS zu dem Datenelement RS-TIME kombiniert worden sind.

[0068] Die Codeerzeugungseinheit **24** wendet die Modifikationsspezifikation an, um die Modifikationen zu bestimmen, die für den Altcode notwendig sind, um geeignete Tagbezogene Daten an das XML-Schema auszugeben. Zum Beispiel wird der folgende Code durch die Codeerzeugungseinheit **24** gemäß der Modifikationsspezifikation hinzugefügt, um XML-formatierte Daten aus den modifizierten Altprogrammanwendungen auszugeben, die sich auf das Ereignis 47 beziehen:

```
MOVE    RECORDS-SELECTED-EDIT    TO XML-BUFFER
MOVE    XML-TOTAL-CALLS-ID       TO XML-UID
CALL    'XML' USING    XML-UID
                       XML-BUFFER
MOVE    RS-TIME                  TO XML-BUFFER
MOVE    XML-TOTAL-TIME-ID        TO XML-UID
CALL    'XML' USING    XML-UID
                       XML-BUFFER
MOVE    RS-COST                  TO XML-BUFFER
MOVE    XML-TOTAL-BILL-ID        TO XML-UID
CALL    'XML' USING    XML-UID
                       XML-BUFFER
```

[0069] Die modifizierte Altprogrammanwendung ruft die Schreibereinheit **20** auf, um eine Ausgabe mit Tags vorzunehmen, die von dem XML-Schema, das in der Kontexttabelle **22** gespeichert ist, bereitgestellt werden. Wenn die modifizierten Altprogrammanwendungen **18** in das Altcomputersystem **12** geladen werden, wird die Schreibereinheit **20** gemeinsam mit der Kontexttabelle **22** durch die modifizierten Altprogrammanwendungen **18** aufgerufen, um einen XML-Datenstrom auszugeben.

[0070] Die vorberechneten Daten, die notwendig sind, um das akkurate Schreiben von eingebetteten XML-Elementen zu steuern, werden durch das XML-Schema erzeugt. Die vorberechneten Daten bestehen aus einer Karte von einem Index zu Tiefe, Start-Bezeichnung, End-Bezeichnung, Eltern-Index und weiteren notwendigen Informationen, um korrektes XML zu erzeugen. Zum Beispiel stellt das XML-Schema, das durch Fig. 7 dargestellt ist, eine Datenstruktur zum Drucken eines Kundennamens, seiner Adresse und Identifikationen zur Verfügung. Fig. 7A stellt die Baumstruktur des XML-Schemas, das in Fig. 7 gezeigt ist, dar. Fig. 7B stellt die berechnete Datenstruktur des XML-Schemas, das durch Fig. 7 gezeigt wird, dar, einschließlich der Tiefe jedes Elementes, das der Position des Elementes in der Baumstruktur und ein Index für jedes Element, die sein Vorgängerelement angibt. Zum Beispiel entspricht das Element „Customer" der Wurzel des XML-Schemas und hat ein Nachfolgerelement „Address". Das „Street"-Element ist ein Nachfolger, des „Address"-Elements, wie durch die Ziffer 3, die der Identifikation des Elements „Address" entspricht, angegeben wird.

[0071] Mit Bezug auf Fig. 8 stellt ein Flussdiagramm den Prozess dar, der in der Schreibereinheit implementiert ist, um einen XML-Datenstrom auszugeben. Die berechneten Daten, die durch Fig. 7B dargestellt sind, werden beim Schreiben des XML-Datenstroms mit Bezug auf das in Fig. 7 dargestellte XML-Schema angewendet. Der Prozess beginnt mit Schritt **100**, in dem ein XML-Druckbefehl (XML PRINT Befehl) mit der Identifikation des Schema-Elementes und des zu druckenden Wertes aufgerufen wird. Zum Beispiel stellen die Befehle:

```
MOVE    '861 East Meadow'          TO XML-BUFFER
MOVE    XML-CUSTOMER-STREET        TO XML-UID
CALL    'XML' USING    XML-UID
                       XML-BUFFER
```

die Identifikation für das Element „Street" der berechneten Datenstruktur zur Verfügung.

[0072] In Schritt **102** wird ein Test vorgenommen, um zu sehen, ob der XML-Druck-Prozess gestartet worden ist, um Daten auszugeben. Wenn nicht, wird die geeignete Datenstruktur oder der aktuelle Kontext initialisiert und die identifizierte Datendatei in Schritt **104** geöffnet. Zum Beispiel würde ein XML-Druck-Befehl, der sich auf Kundendaten bezieht, zu einer Initialisierung des aktuellen Kontext führen, der als Wurzelelement „Customer" aufweist. In Schritt **106** wird ein Test vorgenommen, um festzustellen, ob alle Daten der Datenstruktur ausgegeben worden sind. Wenn alle Daten ausgegeben werden, fährt der Prozess mit Schritt **108** fort, in dem die geeigneten XML-End-Tags ausgegeben werden und die Datendatei geschlossen wird. Wenn jedoch die Koten-ID sich nicht am Ende der Datenstruktur befindet, fährt der Prozess mit Schritt **109** fort. Wenn z. B. die Knoten-ID „City" entspricht, fährt der Prozess mit Schritt **109** fort.

[0073] In Schritt **109** wird ein Test durchgeführt, um festzustellen, ob die aufgerufene Knoten-ID ein Nachfolger des aktuellen Knotens ist. Zum Beispiel entspricht das Element „Street" einem Nachfolger des Elements „Address". Wenn somit das Element „Address" dem aktuellen Element entspricht, und das Element „Street" dem aufgerufenen Element entspricht, fährt der Prozess mit Schritt **110** fort. Wenn im Gegensatz dazu das aktuelle Element „Name" entspricht und das aufgerufene Element dem Element „Street" entspricht, fährt der Prozess mit Schritt **112** fort, um die Knoten-ID des nächsten gemeinsamen Vorgängerknotens mit einer konsistenten Kardinalität zu dem aufgerufenen Element zu lokalisieren. Somit würde als der gemeinsame Vorfahre der Elemente „Name" und „Street", das Element „Customer" identifiziert werden. In Schritt **114** werden die End-Tags bis zu dem Element „Customer" abgeschlossen, und der Prozess fährt mit Schritt **110** fort. Die Überprüfung der Kardinalität in Schritt **112** stellt sicher, dass, wenn ein Vorgänger nur ein einzelnes Auftreten eines Nachfolgers erlaubt, der Nachfolger nur einmal gedruckt wird. Wenn z. B. ein Nachfolgerelement in aufeinander folgenden Ereignissen ausgegeben wird, gibt die Kardinalität an, dass zwischen jeder Ausgabe des Nachfolgers das Vorgängerelement geschlossen wird und eine neue Instanz des Vorgängers geöffnet wird.

[0074] In Schritt **110** werden Tags von dem identifizierten Vorgänger bis hinunter zum aufgerufenen Knoten geöffnet, und Attribute der Knoten entlang der Baumstruktur werden gemeinsam mit den geeigneten Werten ausgegeben. In Schritt **116** kehrt der Prozess zu dem Schritt **100** zurück, um den nächsten Wert in dem XML-Datenstrom zu akzeptieren.

[0075] Eine zusätzliche Funktion der Schreibereinheit **20** entspricht der verzögerten Verarbeitung zum Schreiben von Daten als vollständige Datenstrukturen. Zum Beispiel speichert die Schreibereinheit **20** Attribute, Werte und Textwerte in eine Datenstruktur ohne die Daten auszugeben, bis alle Attribute, Werte und Textwerte der Datenstruktur vollständig sind. Diese verzögerte Verarbeitung ermöglicht es der Schreibereinheit, sich an die Ablaufsbedingungen des XML-Schemas zu halten.

[0076] Die Beispielausgabe unten zeigt die Notwendigkeit dieser Fähigkeit.

BEISPIELAUSGABE

	Sende Scheck zahlbar an
John Doe	ABC WIRELESS
111 Mizar P1	P.0. BOX 666666
Pasadena CA 93436-1204	DALLAS TX 75263-1111

[0077] Zwei Adressen werden nebeneinander auf der Seite gedruckt. Eine ist die Kundenadresse und die andere ist die Rechnungsadresse. Somit enthält eine einzige Zeile der Ausgabe abwechselnd Elemente von zwei verschiedenen Unterschemas gemäß dem unten gezeigten Ziel-XML-Schema.

ZIEL XML SCHEMA

```
< ElementType name = "name"/ >
< ElementType name = "address"/ >
< ElementType name = "phone-number"/ >
< ElementType name = "city-state-zip"/ >
< ElementType name = "customer" >
  < element type = "name"/ >
  < element type = "address"/ >
  < element type = "city-state-zip"/ >
< /ElementType >
< ElementType name = "remitter" >
  < element type = "name"/ >
  < element type = "address"/ >
  < element type = "city-state-zip"/ >
< /ElementType >
< ElementType name = "bill-header" >
  < element type = "customer"/ >
  < element type = "remitter"/ >
< /ElementType >
```

[0078] Ein vollständiges Kundenadressen-Unterschema muss vor dem Rechnungsadress-Unterschema ausgegeben werden. Aufgrund der Struktur des Altcodes (unten gezeigt) ist es notwendig, die Komponenten der Rechnungsadresse zwischenzuspeichern, während die XML-Struktur für den Kunden geschrieben wird. Zusätzlich zu ihrer anderen Buchführungsaufgabe stellt die Kontexttabelle einen Speicher für diese Zwischenspeicheroperation zur Verfügung.

[0079] Der ursprüngliche Altcode kann nachfolgend gesehen werden:

AUSZUG AUS ALT-COBOL-DATEN-ERKLÄRUNGEN

```
05    L-BILL-HEADER-10.
      10 FILLER PIC X (49) VALUE SPACES.
      10 FILLER PIC X (32) VALUE "Sende Scheck zahlbar an".
05    HL-BILL-HEADER-11.
10    FILLER PIC X VALUE SPACES.
10    HLS-CUSTOMER-NAME PIC X(40) VALUE SPACES.
10    HLS-REMITTANCE-NAME PIC X (40) VALUE SPACES.
05    HL-BILL-HEADER-12.
10    FILLER PIC X VALUE SPACES.
10    HLS-CUSTOMER-ADDRESS PIC X(40) VALUE SPACES.
10    HLS-REMITTANCE-ADDRESS PIC X (40) VALUE SPACES.
05    HL-BILL-HEADER-13.
10    FILLER PIC X VALUE SPACES.
10    HLS-CT-ST-ZIP PIC X (40) VALUE SPACES.
10    HLS-REMITTANCE-CT-ST-ZIP PIC X (40) VALUE SPACES.
```

AUSZUG EINES ALT-COBOL-PROZESSCODES

```
WRITE BILL-RECORD FROM HL-BILL-HEADER-10 AFTER 2
WRITE BILL-RECORD FROMHL-BILL-HEADER-11
WRITE BILL-RECORD FROM HL-BILL-HEADER-12
WRITE BILL-RECORD FROM HL-BILL-HEADER-13
```

[0080] Der modifizierte Code wird nachfolgend gezeigt mit Anmerkungen, die die darauf folgenden Operationen beschreiben.

MODIFIZIERTER ALT-COBOL-PROZESSCODE

```
* Unverändert, da er nichts für das Schema relevantes ausgibt
      WRITE BILL-RECORD FROM HL-BILL-HEADER-10 AFTER 2
* Gebe Kundennamen aus
      MOVE HLS-CUSTOMER-NAME TO XML-VALUE
      MOVE CUSTOMER-NAME-ID TO XML-TAG
```

```
        CALL"XML" USING XML-TAG XML-VALUE
* Verhindertes Schreiben des Namens des Übersenders
        MOVE HLS-REMITTANCE-NAME TO XML-VALUE
        MOVE REMITTER-NAME-ID TO XML-TAG
        CALL"XML-SET-NODE-VALUE" USING XML-TAG XML-VALUE
        WRITE BILL-RECORD FROMHL-BILL-HEADER-11
* Gebe Kundenadresse aus
        MOVE HLS-CUSTOMER-ADDRESS TO XML-VALUE
        MOVE CUSTOMER-ADDRESS-ID TO XML-TAG
        CALL"XML" USING XML-TAG XML-VALUE
* Verhindertes Schreiben der Adresse des Übersenders
        MOVE HLS-REMITTANCE-ADDRESS TO XML-VALUE
        MOVE REMITTER-ADDRESS-ID TO XML-TAG
        CALL"XML-SET-NODE-VALUE" USING XML-TAG XML-VALUE
        WRITE BILL-RECORD FROMHL-BILL-HEADER-12
* Gebe City-State-Zip des Kunden aus
        MOVE HLS-CT-ST-ZIP TO XML-VALUE
        MOVE CUSTOMER-CITY-STATE-ZIP-ID TO XML-TAG
        CALL"XML" XML-TAG XML-VALUE
* Verhindertes Schreiben der City-State-Zip des Übersenders
        MOVE HLS-REMITTANCE-CT-ST-ZIP TO XML-VALUE
        MOVE REMITTER-CITY-STATE-ZIP-ID TO XML-TAG
        CALL"XML-SET-NODE-VALUE" USING XML-TAG XML-VALUE
        WRITE BILL-RECORD FROM HL-BILL-HEADER-13
* Schreiben des unterdrückten Übersender-Knotens mit Unterknoten.
        MOVE XML-REMITTER-ID TO XML-TAG
        CALL"XML-WRITE-NODE" USING XML-TAG
```

[0081] Die resultierende Ausgabe für dieses bestimmte Beispiel kann nachfolgend gesehen werden.

XML AUSGABE

```
< bill-header >
< customer >
```

< name > John Doe < /name >

< address > 111 Mizar Pl < /address >

< city-state-zip > Pasadena CA 93436-1204 < /city-state-zip >

< /customer >

< remitter >

< name > ABCWIRELESS < /name >

< address > P.0. BOX 666666 < /address >

< city-state-zip > DALLAS TX 75263-1111 < /city-state-zip >

< /remitter >

< /bill-header >

[0082] Ein XML-Schema kann Kardinalitätsbeschränkungen für die Komponentenelemente vorgeben. Zum Beispiel kann in dem unten gezeigten Schema C, C1 und C2 jeweils nur einmal innerhalb ihrer entsprechenden Eltern erscheinen. Es ist wichtig, diese Eigenschaft sicherzustellen, wenn eine Instanz dieses Schemas erzeugt wird.

< ElementTypename = "C1" >

< ElementTypename = "C2" >

< ElementType name = "C" >

< element type ="C1" maxOccurs ="1"/ >

< element type ="C2" maxOccurs ="1"/ >

< /ElementType >

< ElementType name = "A" >

< element type = "C" maxOccurs = "1"/ >

< /ElementType >

[0083] Wenn einige der vorberechneten Elemente der Kontexttabelle, die das Schema darstellen, das bei „A" seine Wurzel hat, werden in der nachfolgenden Tabelle gezeigt.

ID	Bezeichnung	Tiefe	Eltern	Kardinalität
1	<A>	1	0	n
2	<C>	2	1	1
3	<C1>	3	2	1
4	<C2>	3	2	1

[0084] Die ID-Spalte speichert den eindeutigen Identifizierer, der jedem Element zugeordnet ist. Die Kardinalitätsspalte gibt eine Bedingung über die Anzahl von Ereignissen eines Elements innerhalb seines Vorgängerelements an. Ein ‚n' bedeutet, dass sie Null oder mehr sein kann. ‚1' bedeutet, dass sie genau 1 ist.

[0085] Die unten gezeigte Tabelle zeigt, wie diese Information dynamisch genutzt wird, wenn XML-PRINT Befehle ausgeführt werden. (Es wird angemerkt, dass die COUNT-Spalte des CONTEXT die Änderung des Werts der Kardinalitätszahl hinsichtlich eines bestimmten Schema-Elements zeigt.)

CONTEXT

ZUSTAND	STACK	COUNT	BEFEHL	AUSGABE
0	[]	A = 1	XML-PRINT C1, V11	<A>
1	[A]	C = 1		<C>
2	[A, C]	C1 = 1		<C1> V11 </C1>
3	[A, C]	C2 = 1	XML-PRINT C2, V21	<C2> V21 </C2>
4	[A, C]	C1 = 0	XML-PRINT C1, V12	</C> C2 = 0
5	[A]	C = 0		</A>
6	[]	A = 2		<A>
7	[A]	C = 1		<C>
8	[A, C]C1 = 1			<C1> V12 </C1>

[0086] Der Anfangszustand „0" umfasst einen leeren Stack und keine Kardinalitätszahlen, die mit einem Schema-Element zugeordnet sind. Der Befehl, V11 als ein Schema-Element C1 zu drucken, führt zu einer Überprüfung des Zustands, der Ausgabe der <A> und <C> Vorgängerbezeichnungen und der Ausgabe des mit einer Bezeichnung versehenen Elements V11. Der STACK wird modifiziert, um den aktuellen Kontext eines geöffneten <A> und <C> aufzuzeichnen, und die Kardinalitätszahlen für A, C und C1 werden auf 1 gesetzt.

[0087] Der Befehl, V21 als ein Schema-Element C2 zu drucken, führt zu einer Überprüfung des Zustands. Der STACK hinsichtlich der Vorgänger von C2 ist korrekt, so dass die einzige Druckoperation die Ausgabe des mit dem Label versehenen Elements V21 ist. Der STACK ist unverändert. Die Kardinalitätszahl für C2 wird auf 1 festgelegt.

[0088] Der Befehl, V12 zu drucken, das durch das Schema-Element C1 gekennzeichnet ist, bewirkt eine Überprüfung des Zustands. Der STACK in Zustand 3 hinsichtlich der Vorgänger von C1 ist korrekt. Jedoch ist die Kardinalitätszahl für C1 gleich 1, was der erlaubten Kardinalität von Elementen dieses Typs entspricht. Wir schließen daher C und setzen die Kardinalitätszahlen für seine Kinder C1 und C2 zurück. Zu diesem Punkt kann man feststellen, dass die Kardinalitätszahl für C gleich 1 ist, was der erlaubten Kardinalität der Element dieses Typs entspricht. Wir schließen daher A und setzen die Kardinalitätszahl für C auf 0 zurück. Zu diesem Punkt (Zustand 6) ist der STACK geleert und wir geben die Vorgängerbezeichnungen <A> und <C> aus, geben das mit einer Bezeichnung versehene Element V12 aus, modifizieren den STACK, um den aktuellen Kontext eines geöffneten <A> und <C> aufzuzeichnen und legen die Kardinalitätszahlen für C und C1 auf 1 und A auf 2 fest.

[0089] Nun wird der Fall berücksichtigt, bei dem das maximale Auftreten von Elementen des Typs C keine Obergrenze aufweist. Das heißt, die Elementedefinition von C innerhalb von A wird geändert zu:
<element type = "C" max0ccurs = "n"/>

[0090] Der dritte Druckschritt wird nun einfacher, wie in der nachfolgenden Tabelle gezeigt:

CONTEXT

ZUSTAND	STACK	COUNT	BEFEHL	AUSGABE
0	[]	A = 1	XML-PRINT C1, V11	<A>
1	[A]	C = 1		<C>
2	[A, C]	C1 = 1		<C1> V11 </C1>
3	[A, C]	C2 = 1	XML-PRINT C2, V22	<C2> V22 </C2>
4	[A, C]	C1 = 0	XML-PRINT C1, V12	</C>
		C2 = 0		
5	[A]	C = 2		<C>
6	[A, C]	C1 = 1		<C1> V12 </C1>

[0091] Die ersten zwei XML-PRINT Operationen werden wie zuvor durchgeführt. Weil eine willkürliche Anzahl von C Unterelementen von A vorliegen kann, muss das A nicht geschlossen werden und ein neues geöffnet werden. Wir schließen C, setzen den STACK auf [A] und setzen die Kardinalitätszahlen für die Nachfolger von C, C1 und C2 zurück. Wir öffnen ein neues C und inkrementieren die Kardinalitätszahl von C auf 2. Schließlich wird das mit einem Label versehene Element V12 ausgegeben und die Kardinalitätszahl für C1 auf 1 festgelegt. Schließlich liegt im Gegensatz zu den vorangehenden Beispielen dieses Falles keine Obergrenze für die Ereignisse jedes Elementes vor. Das heißt, die Definitionen der Element C, C1 und C2 werden geändert zu:

< element type = "C1" maxOccurs = "n"/ >

< element type = "C2" maxOccurs = "n"/ >

< element type = "C" maxOccurs = "n"/ >

[0092] Die Zustandsänderungen können in der nachfolgenden Tabelle gesehen werden:

CONTEXT

ZUSTAND	STACK	COUNT	BEFEHL	AUSGABE
1	[]	A = 1	XML-PRINT C1, V11	<A>
2	[A]	C = 1		<C>
3	[A, C]	C1 = 1		<C1> V11 </C1>
4	[A, C]	C2 = 1	XML-PRINT C2, V22	<C2> V22 </C2>
5	[A, C]	C1 = 2	XML-PRINT C1, V12	<C1> V12 </C1>

[0093] Die ersten und zweiten Aufrufe arbeiten wie zuvor beschrieben. Der dritte Aufruf wird sogar einfacher. Weil eine willkürliche Anzahl von C1 Unterelementen von C vorliegen können, muss C nicht geschlossen werden und ein neues geöffnet werden. Das gekennzeichnete Element V12 wird ausgegeben und die Kardinalitätszahl für C1 auf 2 inkrementiert.

[0094] Wenn man Altcode modifiziert, treten bestimmte Schwierigkeiten auf, zu entscheiden, wann Schemadaten, die in Kopfabschnitten und Fußabschnitten enthalten sind, gedruckt werden sollen. Beim Betrachten des Beispiels der Telefonrechnung kann die Ausgabe eines Rechnungsstellungsprogramms eine Abfolge von Rechnungen enthalten. Jede Rechnung kann eine einzelne Seite oder mehrere Seiten einnehmen. Wenn die Rechnung mehrere Seiten belegt, wird sein Kopfabschnitt üblicherweise wiederholt. Als Ergebnis führt der Kopfabschnitt manchmal ein neues Rechnungsschema-Element ein, und zu anderen Zeiten ist es bloße Seitendekoration der für den Menschen lesbaren Ausgabe. Um die Notwendigkeit zu erkennen, den aktuellen Rechnungstag zu schließen und einen neuen zu öffnen, muss man wissen, dass es einen eindeutigen Identifizierer gibt, der jeder Rechnungsinstanz zugeordnet ist, und dass, wenn der Wert dieses ‚Key' sich ändert, die aktuelle Rechnung geschlossen wird und eine neue geöffnet wird. Um diese Berechnung zu ermöglichen, enthält die Kontexttabelle einen Boole'schen Identifizierer für Schlüsselelemente und die aktuellen Werte dieser

Elemente. Diese Überprüfung wird gleichzeitig wie die Überprüfung der Kardinalität durchgeführt.

[0095] Bei einer alternativen Ausführungsform werden von einem Computerprogramm ausgegebene Daten als eine DOM-Instanz vor ihrer Ausgabe effektiv zwischengespeichert. Zum Beispiel wird eine Altcomputeranwendung für einen Telefonausdruck, der Daten als eine Druckroutine ausgibt, wahrscheinlich nicht die Daten in einer Abfolge ausgeben, die die Erzeugung von XML gemäß einer gewünschten Schemastruktur ohne eine erhebliche Umstrukturierung der Daten nach ihrer Ausgabe ausgeben. Somit erfordert es zwei Prozessschritte zum Erzeugen einer XML-Ausgabe, erstens das Ausgeben von XML-Daten gemäß einem Schema, das die natürliche Struktur der Daten, die von dem zugrunde liegenden Altprogramm gedruckt wird, wiederspiegelt, und zweitens eine Verarbeitung der ausgegebenen Daten durch ein separates Programm, das ein XSLT-Stylesheet anwendet, um das gewünschte Format zu erzeugen. Um diesen Prozess zu vereinfachen, baut die vorliegende Erfindung das gesamte endgültige Ziel-DOM in das ursprüngliche Altprogramm ein, um somit im Ergebnis Daten zwischenzuspeichern, um die Daten, wenn diese vollständig sind, auszugeben.

[0096] Die Ausgabe einer XML-Datenstruktur mit einer DOM-Instanz umfasst die Erzeugung von vorberechneten Daten, um die Erzeugung von eingebetteten XML-Komponenten gemäß einem XML-Schema genau zu steuern, und anschließend die Anwendung der vorberechneten Daten, um eine gewünschte XML-Datenstruktur zu erzeugen. Mit Bezug auf **Fig. 9** stellt ein Flussdiagramm die folgenden Schritte dar, um vorberechnete Daten anzuwenden, um ein gewünschtes XML-Dokument auszugeben. In Schritt **120** wird ein Aufruf mit einem XML-Knoten-ID-Tag-Identifizierer vorgenommen, um den Pfad zu dem XML-Knoten, einen Knotenwert, um den einzufügenden Wert zu identifizieren, und einen optionalen Kontext zu identifizieren, der verwendet werden kann, um den Default-Kontext zu übergehen.

[0097] In Schritt **122** stellt ein Test fest, ob ein Kontextwert bereitgestellt wurde. Wenn nicht, wird in Schritt **126** der Kontext auf den Default-Kontext festgelegt.

[0098] In Schritt **128** stellt ein Test fest, ob der zu erzeugende Knoten ein Nachfolger des aktuellen Kontextes ist. Wenn nicht, wird in Schritt **130** ein Vorgängerknoten gefunden, der der kleinste Vorgänger sowohl von dem aktuellen Kontext als auch dem aufgerufenen Knoten-ID ist, der einer Kardinalitätsüberprüfung genügt, und der aktuelle Kontext wird auf den gemeinsamen Vorgänger festgelegt. Wenn ein geeigneter Vorgänger gefunden wird, werden in Schritt **132** Knoten aus dem aktuellen Kontext zu dem aufgerufenen Knoten-ID mit Attributen und Text, soweit notwendig, erzeugt. In Schritt **134** stellt ein Test fest, ob ein Kontextwert als Teil des Aufrufs bereitgestellt wurde. Wenn nicht, wird in Schritt **136** der Default-Kontext auf den aktuellen Kontext festgelegt. Das Verfahren kehrt dann zu Schritt **138** zurück.

[0099] Als ein Beispiel dient die folgende Abfolge von Aufrufen:
CALL XML-GEN XML-CURRENT-ADDRESS, "true"
CALL XML-GEN XML-STREET-ADDRESS, "861 East Meadow"

erzeugt die Baumstruktur, die folgendes enthält:

XML:

```
< Customer >
    < Address current="true" >
    < Street > 861 EastMeadow < /Street >
    < /Address >
< /Customer >
```

[0100] Somit steigert die automatische Erzeugung von Datenstrukturen von einem XML-Schema und einer kontextsensitiven Erzeugung von DOM-Instanzen die Einfachhheit der Verwendung von XML sowohl mit neuen Anwendungen als auch mit Anwendungen, die aus Altsystemen konvertiert wurden. Die Automatisierung reduziert die Zeit zur Entwicklung eines neuen Codes und die Durchsicht von Altanwendungen und reduziert auch die Wahrscheinlichkeit von Fehlern aufgrund des Festhaltens an XML-Schema-Anforderungen. Weiterhin wird die Erzeugung von XML-Daten aus einem Altsystem mit einem Zielschema, das von der natürlichen Struktur von Daten, die von dem Altsystem ausgegeben werden, verschieden ist, durch die Transformation des DOM mit einem XSLT-Stylesheet vereinfacht. Im Ergebnis dient die DOM-Instanz als ein Zwischenspeicher, der Daten, die von dem zugrunde liegenden Programm ausgegeben werden, speichert, bis eine gewünschte

Ausgabe vorbereitet ist, ohne erhebliche Revision der Struktur des zugrunde liegenden Programms.

[0101] Der Aufbau einer DOM-Instanz wird durch das folgende Beispiel dargestellt. Ein Altprogramm gibt Noteberichte für Untergraduierten und Graduierten-Programme aus. Der natürliche Steuerfluss des ursprünglichen Altprogramms entspricht der folgenden XML-Ausgabe:

```
< courseList >
        < course >
                < name > Math 101 < /name >
                < type > undergrad < /type >
        < /course >
        < course >
                < name > Math 395 < /name >
                type > grad < /type > < /course >
        < course >
                name > CS 101 < /name >
                type > undergrad < /type >
        < /course >
        < course >
                name > CS 600 < /name >
                type > grad < /type >
        < /course >
< /courseList >
```

[0102] Das Ziel-XDR-Schema für die Datenausgabe aus dem Altprogramm ist:

```
SCHEMA: courseList2. xml
< ElementType name = "course" >
< ElementType name = "undergrad"/ >
        < element type = "course"/ >
< /ElementType >
< ElementType name = "grad"/ >
        < element type = "course"/ >
< /ElementType >
< ElementType name = "courseList" >
        < element type = "undergrad" maxOccurs = "1"/ >
        < element type = "grad"maxOccurs="1"/ >
< /ElementType >
< /Schema >
```

[0103] Die Daten, die gemäß dem Ziel-XDR-Schema formatiert sind, im Gegensatz zu dem ‚natürlichen' Pro-

grammsteuerfluss sind:

OUTPUT 2

```
< courseList >
    < undergrad >
        course > Math 101 < /course >
        course > CS 101 < /course >
    < /undergrad >
    < grad >
        < course > Math 395 < /course >
        < course > CS 600 < /course >
    < /grad >
< /courseList >
```

[0104] Der Arbeitsspeicherabschnitt und die Verfahrensunterteilung des Altprogramms werden überarbeitet, um Daten gemäß dem Zielschema anstelle der ‚natürlichen' Darstellung gemäß dem SCHEMA courselist2.xml auszugeben, wie z. B.:
Arbeitsspeicherabschnitt.

01 xmlvars.

```
*     Handles
      05    gradHandle pic 9 (4) comp-5.
      05    undergradHandle pic 9 (4) comp-5
      05    gradCourseHandle pic 9 (4) comp-5.
      05    undergradCourseHandle pic 9 (4) comp-5.
*     Contexts
      05    context pic 9 (4) comp-5.
      05    gradContext pic 9 (4) comp-5.
```

Prozedurunterteilung.

```
*     Öffne und verarbeite Schema
      Call "xmlOpenSchema" "courseSchema2. xml"
*     Aufbaue Handles
      Call "xmlPathToHandle" using "grad" gradHandle
      Call"xmlPathToHandle"using"undergrad" undergradHandle
      Call "xmlPathToHandle" using "grad/course" gradCourseHandle
      Call "xmlPathToHandle" using "undergrad/course" undergradCourseHandle
```

```
* Erzeuge Wurzel und untergrad-Knoten and etabliere Kontext an diesem Knoten
      Call "xmlCreateNode" using undergradHandle " " context
*     Aufbaue gradnode aber ändere nicht den Kontext
      Call "xmlCreateNodeincontext" using context gradHandle "" gradContext
*     Aufbaue die Knoten --- wir vermischen die XML-Zeilenausdrucke mit
*     Pseudocode, der eine hypothetische courselist erzeugt
*     WRITE "Math 101""undergrad"
      Call "xmlCreateNode" using undergradCourseHandle "Math 101"
*     WRITE "Math 395" "grad"
      Call "xmlCreateNodeincontext" using gradContext gradCourseHandle
           "Math 395" gradContext
*     WRITE "CS 101" "undergrad"
      Call "xmlCreateNode" using undergradCourseHandle "CS 101"
*     WRITE"CS 600""grad"
      Call "xmlCreateNodeincontext" using gradContext gradCourseHandle
           "CS 600" gradContext
*     Schreibe die XML_Ausgabedatei gemäß dem Eingabeschema
      Call "xmlWriteFile""basic. xml"
```

[0105] Das modifizierte Altprogramm im Arbeitsspeicher erzeugt eine Wurzel und einen untergraduierten Knoten und stellt einen Kontext an dem untergraduierten Knoten zur Verfügung. Der graduierte Knoten wird dann so erzeugt, dass der Wurzelknoten der kleinste gemeinsam genutzte Vorgänger der untergraduierten und graduierten Knoten ist, der Kontext bleibt jedoch unverändert. Ein Zeiger gradHandle, der dem graduierten Knoten zugeordnet ist, ermöglicht das Schreiben von Daten auf diesem Knoten, ohne den Kontext von dem des untergraduierten Knoten zu ändern. Zum Beispiel werden durch Aufrufen „xmlCreateNode" innerhalb des Default (untergraduierten) Kontextes die untergraduierten Kurse von „Math101" und „CS101" mit dem untergraduierten und Kurstags geschrieben. Durch Aufrufen „xmlCreateNodein Context" richten ein Schreiben der graduierten Kurse „Math395" und „CS600" mit grad- und course-Tags. Somit werden Daten gemäß einem Schema geschrieben, das von der natürlichen Ausgabe des zugrunde liegenden Programms verschieden ist.

[0106] Die vorliegende Erfindung weist eine Anzahl von wichtigen Geschäftsanwendungen auf, die sich auf E-Commerce beziehen und auf eine effizientere Benutzung von Altcomputerberichten durch Stein und Mörtel Geschäfte. Ein Beispiel ist, dass interne Berichte ansonsten auf Papier für die manuelle Durchsicht gedruckt werden lind ansonsten zum Speichern in einer Datenbank in einem XML-Format zur Verfügung stehen. Sind die Berichte elektronisch gespeichert, sind diese als elektronische Informationsquellen für die Durchsicht mit einem Browser oder einer anderen elektronischen Analyse verfügbar. Die Berichte sind auch einfacher in einer Datenbank (data warehouse) zu speichern.

[0107] Eine weitere kommerzielle Anwendung ist als Enterprise Application Integration (EAI) Middleware zur Übertragung von Daten zwischen Anwendungen. Das Vornehmen einer Übertragung von Daten von strukturierten Datenbanken, die XML-Format verwenden, ist relativ gradlinig, da Datendefinitionen als semantische Tags behandelt werden können. Im Gegensatz dazu sind typische Altcomputersystemberichte unstrukturiert, da sie Daten darstellen, die gemäß einer Geschäftslogik anstelle einer Datenstruktur erzeugt werden. Durch Modifizieren von zugrunde liegenden Altanwendungen, um direkt XML-formatierte Daten auszugeben, werden die ausgegebenen Daten einfacher als die strukturierten Datendateien zur Integration in einer Folge von Unternehmensanwendungen behandelt.

[0108] Eine weitere kommerzielle Anwendung ist als die Electronic Bill Presentment and Payment (EBPP).

Um eine elektronische Rechnungsstellung von typischen Altcomputersystemen bereitzustellen, wird im Allgemeinen ein Parser verwendet, um nicht mit Tag versehene Rechnungsdatendateien zu durchsuchen und dann die Datendateien mit semantisch bedeutsamen Identifizierern zu kennzeichnen. Parser sind aufwendig und schwierig zu integrieren und zu warten. Im Gegensatz dazu erspart die Modifikation von zugrunde liegenden Altcomputersystemcode, um direkt XML-formatierte Daten auszugeben, Zeit, erfordert weniger Kenntnisse und Aufwand und stellt Daten in einem anerkannten Format für E-Commerce zur Verfügung. Somit können Geschäfte mit Altcomputersystemen XML-formatierte Berichte ausgeben, die es dem Geschäft ermöglichen, an den Fortschritten im E-Commerce Vorteile zu ziehen, wie z. B. die automatische Rechnungsbezahlung. Zum Beispiel könnten individuelle Telefonkunden ihre Telefonrechnung über E-Mail, die einen Weblink zu einer Seite enthält, die die individuellen Rechnungsdetails bereitstellt, erhalten.

[0109] Eine weitere kommerzielle Anwendung ist die Archivierung von Rechnungen. Zum Beispiel Banken halten große Archive von Kundenrechnungen als verkleinerte fotografische Kopien auf Microfiche oder als Druckströme auf optischen Plattensystemen aufrecht. Abrufsysteme für diese Archive sind komplex und schwierig aufrecht zu erhalten. Die Datenextraktion aus den Druckströmen ist eine jüngste Verbesserung, wie in US-Patentnummer 6,031,625 (US 6,031,625) offenbart ist, jedoch erfordert ein solches System die Verarbeitung von Druckströmen, nachdem sie von der Altanwendung ausgegeben worden sind. Im Gegensatz dazu macht die Modifizierung des zugrunde liegenden Altcomputercodes, so dass er direkt XML-formatierte Rechnungen erzeugt, die Archivierung und den Abruf von Rechnungen viel einfacher. Zum Beispiel können XML-Ausdrücke in einer relationalen Datenbank für die einfachen Abrufe gespeichert werden. Zusätzlich werden die abgerufenen Ausdrücke, weil sie eine XML-Darstellung aufweisen, direkt ansehbar, z. B. mit Hilfe einer Browsertechnologie.

[0110] Eine weitere kommerzielle Anwendung ist als Informationsdienst, der versucht, elektronische Informationsspeicher zu analysieren, um geschäftsbezogenes Verhalten zu bestimmen, wie z. B. das Kauf- oder Verkaufsverhalten. Verbundene Datenanbieter erhalten Daten für die Intelligenzanalyse über Berichte, die auf einer Vertreiber- oder Erwerberbasis analysiert werden. Dieses ausführliche Durchsuchen kann sogar noch komplizierter sein als das Durchsuchen, das verwendet wird, um eine EBPP-Funktion zu unterstützen. Somit ist eine direkte Erzeugung von XML-formatierten Daten aus einem Altcomputersystem, das Rechnungsberichte bereitstellt, sogar effizienter in der Rolle als Informationsdienst als bei der elektronischen Rechnungsstellung und anderen Anwendungen, da eine ausführliche Datenanalyse möglich ist, ohne detaillierte Durchsuchungssysteme anzuwenden.

[0111] Insgesamt reduziert die direkte Erzeugung von XML-formatierten Daten aus Altcomputersystemen die Reibung in Informationsnetzwerken, indem die Übertragung von Informationen einfacher gemacht wird. Dies reduziert die Kosten des Nachverfolgens einer Information, den manuellen Aufwand, eine Geschäftsinformation auszutauschen und zu analysieren und reduziert die Zeit, die zum Erhalten von wertvollen Geschäftsinformationen aus bestehenden Datenquellen aufgewendet wird. Durch Bereitstellen von Daten in einer semantisch bedeutsamen Form können Kunden automatisch ihre Zulieferer für ein Verkäuferbeziehungsmanagement analysieren, Zulieferer können automatisch ihre Kunden hinsichtlich eines Kundenbeziehungsmanagements analysieren und Hersteller können automatisch Märkte für ihre Produkte hinsichtlich einer Vermarktungsintelligenz analysieren.

[0112] Obwohl die vorliegende Erfindung ausführlich beschrieben worden ist, ist es selbstverständlich, dass vielfältige Änderungen, Ersetzungen und Abwandlungen hierin vorgenommen werden können, ohne von dem Bereich der Erfindung, der durch die beigefügten Ansprüche definiert ist, abzuweichen.

Patentansprüche

1. Verfahren zum Ausgeben von Daten von einem Altcomputer-System (**12**) mithilfe einer Extensible-Markup-Sprache mit folgenden Schritten:
Erzeugen eines Modells von ausgegebenen Daten, die von einer Anwendung, die sich auf dem Altcomputer-System (**12**) befindet, indem Vorgänge der Anwendungen des Altcomputer-Systems (**12**), die Daten ausgeben, identifiziert werden;
Zuweisen der Vorgänge zu dem Extensible-Markup-Sprachen-Schema;
Definieren eines Steuerungsfluss-Graphen der Ausgabevorgänge;
Erzeugen einer Spezifikation, um die Anwendungen des Altcomputer-Systems zu modifizieren, um eine Ausgabe von einer Dokumentenobjektmodell-Instanz in der Extensible-Markup-Sprache bereitzustellen; und
Automatisches Modifizieren der Anwendungen des Altcomputersystems gemäß der Spezifikation;
Zuordnen des Modells des Altcomputersystems (**12**) zu einem Extensible-Markup-Sprachen-Schema; und

Automatisches Modifizieren einer oder mehrerer Anwendungen (**16**) des Altcomputersystems (**12**), in dem das Altcomputersystem (**12**) mit einer Schreibereinheit (**20**), wobei die Schreibereinheit (**20**) das Extensible-Markup-Sprachen-Schema als eine Datendatei geladen hat; und
Aufrufen der Schreibereinheit (**20**) mit den modifizierten Anwendungen (**18**), wobei die Schreibereinheit (**20**) in dem Dokumentenobjektmodell gemäß dem Extensible-Markup-Sprachen-Schema vorgesehen wird, indem eine Dokumentenobjektmodell-Instanz mit einem oder mehreren Kontexten aufgebaut wird, wobei die modifizierte Anwendung (**18**) ausgebildet ist, um Daten, die mithilfe eines Dokumentenobjektmodells geschrieben werden, von dem Altcomputersystem (**12**) in einer Extensible-Markup-Sprache auszugeben, indem eine Beziehung der ausgegebenen Daten zu einem oder mehreren Dokumentenobjektmodell-Kontexten in, der Extensible-Markup-Sprache erstellt wird;
Aufbauen einer Dokumentenobjektmodell-Instanz mit dem einen oder den mehreren Kontexten; und
Ausgeben der Daten der Dokumentenobjektmodell-Instanz in der Extensible-Markup-Sprache.

2. Verfahren nach Anspruch 1, mit dem weiteren Schritt:
Anwenden eines oder mehreren XSLT Stylesheets, um die Dokumentenobjektmodell-Instanz zum Ausgeben von Daten in einem vorbestimmten Format umzustrukturieren.

3. Verfahren nach Anspruch 1 oder 2, wobei das Erstellen einer Beziehung weiterhin den Schritt umfasst:
Aktivieren von mehreren Kontexten gleichzeitig, um Daten für die Ausgabe als eine vollständige Dokumentenobjektmodell-Instanz zu Puffern.

4. Verfahren nach Anspruch 3, wobei das Erstellen einer Beziehung weiterhin umfasst:
Erzeugen eines Knotens für ein Ausgangsdatum; und
Gewährleisten der korrekten Kardinalität des erzeugten Knotens.

5. Verfahren nach einer der vorangehenden Ansprüche, mit den weiteren Schritten
Erzeugen von Ausgangsdaten mit der Anwendung (**18**);
Aufrufen der Schreibereinheit (**20**) mit der Anwendung;
Bereitstellen der erzeugten Ausgangsdaten an die Schreibereinheit (**20**);
Ausgeben von Daten von einer Dokumentenobjektmodell-Instanz durch die Schreibereinheit (**20**) gemäß des Extensible-Markup-Sprachen-Schemas.

6. Verfahren nach Anspruch 5, wobei die Schreibereinheit (**20**) eine Anwendung umfasst, die in der Computersprache der Anwendung des Altcomputer-Systems (**12**) betrieben wird.

7. Verfahren nach einem der vorangehenden Ansprüche mit den weiteren Schritten:
Modifizieren der Anwendung (**16**) des Altcomputer-Systems (**12**), um Daten mit einem Schema-Element auszugeben;
Erzeugen von Daten aus der modifizierten Anwendung (**18**);
Ausrichten des Schema-Elements und des aktuellen Kontexts;
Schreiben des Schema-Elements der Ausgangsdaten auf einen aktuellen der mehreren Kontexte eines Extensible-Markup-Sprachen-Schemas; und
Vorsehen eines Dokumentenobjektmodells mit den Daten, um eine Extensible-Markup-Sprach-Instanz auszugeben.

8. Verfahren nach Anspruch 7, wobei das Ausrichten des Schema-Elements die weiteren Schritte umfasst:
Feststellen, dass das Schema-Element ein Nachfahre des aktuellen Kontextes ist; und
Erzeugen des Extensible-Markup-Sprachen-Markers herunter bis zu dem Schema-Element.

9. Verfahren nach Anspruch 8, wobei das Ausrichten des Schemaelements die weiteren Schritte umfasst:
Feststellen eines minimalen gegenseitigen Vorfahrens des Schema-Elements und des aktuellen Kontextes;
Durchqueren der Extensible-Markup-Sprachen-Marker für den aktuellen Kontext bis zu dem gegenseitigen Vorfahren; und
Erzeugen der Extensible-Markup-Sprachen-Marker für das Schema-Element herunter von dem gegenseitigen Vorfahren.

10. System (**10**) zum Ausgeben von Daten aus einem Altcomputer-System (**12**) in einem Extensible-Markup-Sprachen-Format, wobei das System (**10**) umfasst:
eine Modelliereinheit (**28**), die mit dem Altcomputer-System (**12**) verbunden ist, wobei die Modelliereinheit (**28**) ausgebildet ist, um ein Modell der ausgegebenen Daten, die durch eine Anwendung, die sich auf dem Altcom-

puter-System (**12**) befindet, zu erzeugen, wobei die Modelliereinheit (**28**) über eine Schnittstelle mit dem Altcomputer-System verbinden ist, wobei die Modelliereinheit (**28**) ausgebildet ist, um eine auf dem Altcomputer-System (**12**) geladene Anwendung zu analysieren, um Vorgänge innerhalb der Anwendung zu identifizieren, die Daten von dem Altcomputer-System (**12**) ausgeben;
wobei das System (**10**) weiter umfasst:
einen Steuerungsfluss-Graphen für Ausgabeoperationen in den Anwendungen, wobei der Steuerungsfluss-Graph mehrere Knoten umfasst, wobei jeder Knoten einem Ausgabevorgang zugeordnet ist;
eine grafische Benutzerschnittstelle (**30**), die mit der Modelliereinheit (**28**) verbunden ist, wobei die grafische Benutzerschnittstelle (**30**) ausgebildet ist, um den Steuerungsfluss-Graphen und die Vorgänge anzuzeigen, wobei die grafische Benutzerschnittstelle (**30**) die Vorgänge der Anwendungen zu dem Steuerungsfluss-Graphen und einem Extensible-Markup-Sprachen-Schema zuordnen;
eine Zuordnungseinheit (**26**), die mit der Modelliereinheit (**28**) verbunden ist, wobei die Zuordnungseinheit (**26**) ausgebildet ist, um eine Modifikationsspezifikation zu erzeugen, indem das Modell einem Extensible-Markup-Sprachen-Schema zugeordnet wird;
eine Kontexttabelle (**22**), die dem Altcomputer-System (**12**) zugeordnet ist, wobei die Kontexttabelle (**22**) das Extensible-Markup-Sprachen-Schema dem Altcomputer-System (**12**) zur Verfügung stellt; und
eine Schreibereinheit (**20**), die auf das Altcomputer-System (**12**) geladen ist und ein als Datendatei gespeichertes Extensible-Markup-Sprachen-Schema aufweist, wobei die Schreibereinheit (**20**) mit den modifizierten Anwendungen des Altcomputer-Systems in Verbindung steht, um Daten in mehreren Kontexten in dem Dokumentenobjektmodell für die Ausgabe als Extensible-Markus-Sprache zu Puffern, und
eine Code-Erzeugungseinheit (**24**), die mit der Zuordnungseinheit (**26**) und dem Altcomputer-System (**12**) in Verbindung steht, wobei die Code-Erzeugungseinheit (**24**) ausgebildet ist, um den Code der Anwendung des Altcomputer-Systems zu modifizieren, um direkt Daten von einem Dokumentenobjektmodell in einer Extensible-Markup-Sprache auszugeben, wobei eine Schreibereinheit (**20**) auf das Altcomputer-System (**12**) geladen ist und mit der Anwendung über eine Schnittstelle in Verbindung steht, wobei die Schreibereinheit (**20**) ein Extensible-Markup-Sprachen-Schema als eine Datendatei aufweist und die Schreibereinheit (**20**) ausgebildet ist, um die Ausgangsdaten in mehreren aktiven Kontexten zu schreiben;
wobei die Anwendung (**18**) die Schreibereinheit (**20**) aufruft, wenn die Anwendung Daten ausgibt, wobei die Schreibereinheit (**20**) ausgebildet ist, um eine Dokumentenobjektmodell-Instanz für die Ausgabe der Daten gemäß dem Extensible-Markup-Sprachen-Schema aufzubauen.

11. System nach Anspruch 10, wobei die Schreibereinheit (**20**) in der Computersprache des Altcomputer-Systems (**12**) codiert ist.

12. System nach Anspruch 10 oder 11, wobei die Schreibereinheit (**20**) ein Dokumentenobjektmodell als ein Schema-Element vorsieht, das zu dem aktuellen Kontext ausgerichtet ist, indem durch Extensible-Markup-Sprache markierte Knoten bis herunter zu dem Schema-Element der Ausgangsdaten erzeugt werden, wenn das Schema-Element der Ausgangsdaten dem Nachfahren des aktuellen Kontexts entspricht.

13. System nach Anspruch 12, wobei die Schreibereinheit (**20**) weiterhin ausgebildet ist, einen kleinsten gegenseitigen Vorfahren des Schema-Elementes und des aktuellen Kontextes zu bestimmen und um die durch die Extensible-Markup-Sprache markierten Knoten für den aktuellen Kontext bis herauf zu dem kleinsten gegenseitigen Vorfahren zu durchqueren und um Extensible-Markup-Sprachen-Marker für das Schema-Element herunter von dem gegenseitigen Vorfahren zu erzeugen.

14. System nach einem der Ansprüche 10 bis 13, wobei die Anwendung eine Anwendung eines Altcomputer-Systems umfasst, die modifiziert ist, um ein Schema-Element in der Extensible-Markup-Sprache mit Ausgangsdaten auszugeben.

15. System nach Anspruch 14, wobei die Schreibereinheit (**20**) in dem Code des Altcomputersystems (**12**) geschrieben ist.

16. System nach Anspruch 15, wobei der Code COBOL umfasst.

Es folgen 9 Blatt Zeichnungen

Anhängende Zeichnungen

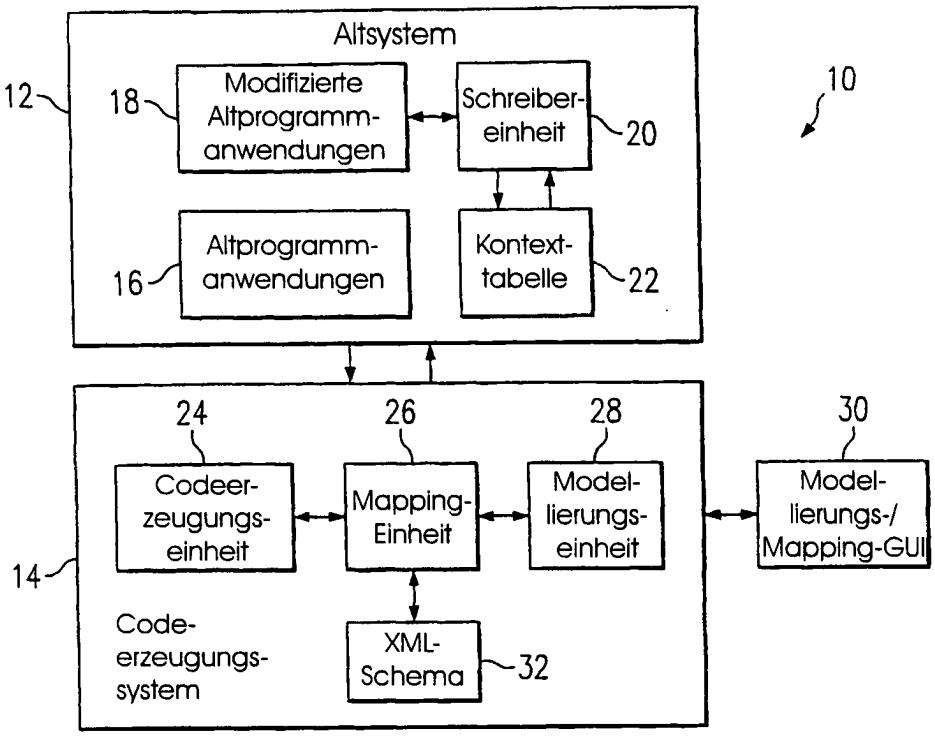


FIG. 1

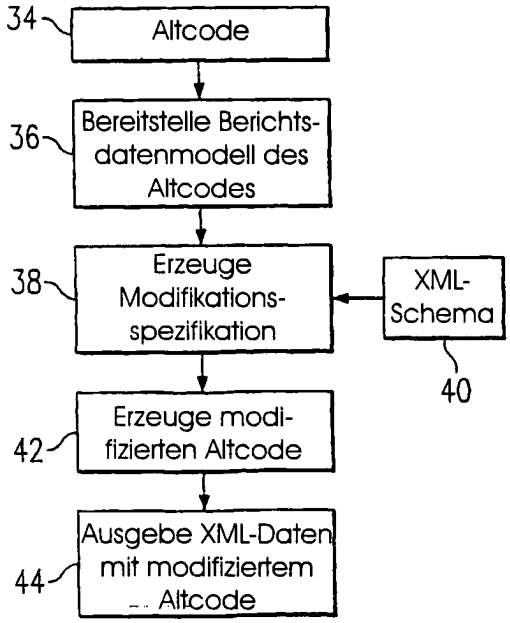


FIG. 2

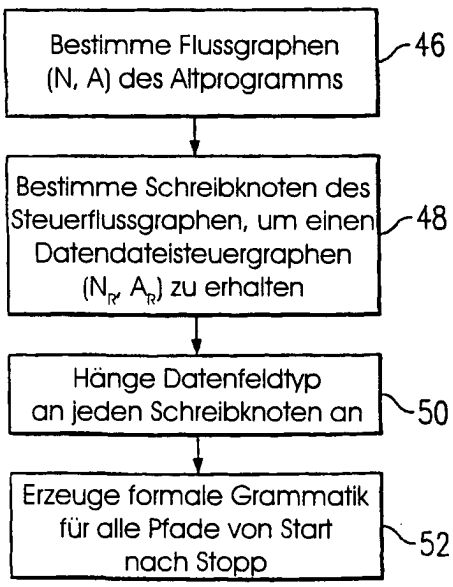


FIG. 3

Date 01/25/2000 American Telephone Company

Monthly Statement for Account Number: 1111111111
COMPANY: EDS

PHONE NUMBER: (214) 999-1212

PERSON: Suzie Q Sample-Data

Date	Time	Rate	Number	City and State		Duration	Cost
12/01/1999	07:15A	B	210-404-6690	San Antoni	TX	00:12.2	$1.22
12/02/1999	01:00A	A	210-404-6690	San Antoni	TX	00:01.0	$.05
12/02/1999	17:45P	D	919-416-1212	Kill Devil	NC	00:00.3	$.06
12/03/1999	15:01P	C	210-404-6690	San Antoni	TX	00:02.0	$.30
12/03/1999	20:23P	D	919-462-1212	Kill Devil	NC	03:02.3	$36.46
12/04/1999	06:06A	B	615-655-1122	Nashville	TN	00:00.5	$.05
12/07/1999	04:00A	A	210-404-6690	San Antoni	TX	01:00.0	$3.00
12/07/1999	15:05P	C	615-655-1122	Nashville	TN	00:40.5	$6.07
12/07/1999	15:45P	C	205-555-1234	Dothan	AL	00:04.3	$.64
12/11/1999	02:13A	A	615-655-1122	Nashville	TN	00:30.0	$1.50
12/11/1999	08:08A	B	210-404-6690	San Antoni	TX	02:20.5	$14.05
12/13/1999	08:00A	B	210-404-6690	San Antoni	TX	01:50.0	$11.00
12/21/1999	00:26A	A	210-404-6690	San Antoni	TX	00:31.3	$1.56
12/21/1999	04:12A	A	919-416-1212	Kill Devil	NC	00:32.0	$1.60
12/21/1999	18:23P	D	615-655-1122	Nashville	TN	00:23.3	$4.67
12/21/1999	19:01P	D	210-404-6690	San Antoni	TX	03:02.4	$36.48
12/22/1999	08:04A	B	205-555-1234	Dothan	AL	00:43.2	$4.32
12/27/1999	12:01A	C	205-555-1234	Dothan	AL	00:13.6	$2.04
12/27/1999	21:12P	D	205-555-1234	Dothan	AL	01:03.0	$12.60

TOTAL CALLS: 19 TOTAL TIME: 16:12:49 $137.67

FIG. 4

```
<bill>
  <account>1111111111</account>
  <customer-name>EDS</customer-name>
  <calling-date>1/billing-date26/2000></>
  <total-bill-by-phone>69.66</total-bill-by-phone>
  <total-time-by-phone>515.6</total-time-by-phone>
  <total-calls>34</total-calls>
  <detail-list>
     <detail-by-phone>
        <phone-number>999-1212</phone-number>
        <area-code>214</area-code>
        <phone-user>Suzie Q Sample-Data</phone-user>
        <calls-by-phone>
          <call>
             <date>12-01-1999</date>
             <time>07.15A/time>
             <code>B</code>
             <phone-number>404-6690</phone-number>
             <area-code>210</area-code>
             <called-city>San Antonio</called-city>
             <called-state>TX</called-state>
             <duration>12.2</duration>
             <charge>1.22</charge>
          </call>
          <call>
             <date>12-01-1999</date>
             <time>01.00A</time>
             <code>A</code>
             <phone-number>404-6690</phone-number>
             <area-code>210</area-code>
             <called-city>San Antonio</called-city>
             <called-state>TX</called-state>
             <duration>01.0</duration>
             <charge>0.05</charge>
          </call>
          <call>
             <date>12-01-1999</date>
             <time>17.45P</time>
             <code>D</code>
             <phone-number>416-1212</phone-number>
             <area-code>919</area-code>
             <called-city>Kill Devil</called-city>
             <called-state>NC</called-state>
             <duration>0.3</duration>
             <charge>0.06</charge>
          </call>
        </calls-by-phone>
        ...
     </detail-by-phone>
     ...
  </detail-list>
</bill>
```

FIG. 5

```
<Schema xmlns="urn: schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <ElementType name="accountno" content="textonly"/>
 <ElementType name="customer-name" content="textonly"/>
 <ElementType name="billing-date" content="textonly"/>
 <ElementType name="total-cost" content="textonly"/>
 <ElementType name="total-number-calls" content="textonly"/>
 <ElementType name="total-time" content="textonly"/>
 <ElementType name="phone-number" content="textonly"/>
 <ElementType name="phone-user" content="textonly"/>
 <ElementType name="date" content="textonly"/>
 <ElementType name="charge" content="textonly"/>
 <ElementType name="called-city" content="textonly"/>
 <ElementType name="called-state" content="textonly"/>
 <ElementType name="duration" content="textonly"/>
 <ElementType name="code" content="textonly"/>
 <ElementType name="area-code" content="textonly"/>
 <ElementType name="time" content="textonly"/>
 <ElementType name="call">
   <element type="area-code" maxOccurs="1" />
   <element type="phone-number" maxOccurs="1" />
   <element type="code" maxOccurs="1" />
   <element type="date" maxOccurs="1" />
   <element type="time" maxOccurs="1" />
   <element type="duration" maxOccurs="1" />
   <element type="charge" maxOccurs="1" />
   <element type="called-city" maxOccurs="1" />
   <element type="called-state" maxOccurs="1" />
 </ElementType>
 <ElementType name="total-bill-by-phone" content="textonly"/>
 <ElementType name="total-time-by-phone" content="textonly"/>
 <ElementType name="total-calls-by-phone" content="textonly"/>
 <ElementType name="calls-by-phone">
   <ElementType name="call"/>
 </ElementType>
 <ElementType name="detail-by-phone">
   <element type="phone-number"/>
   <element type="area-code"/>
   <element type="phone-user"/>
   <element type="total-bill-by-phone"/>
   <element type="total-time-by-phone"/>
   <element type="total-calls-by-phone"/>
   <element type="calls-by-phone"/>
 </ElementType>
 <ElementType name="detail-list">
   <element type="detail-by-phone"/>
 </ElementType>
 <ElementType name="bill">
   <element type="accountno"/>
   <element type="customer-name"/>
   <element type="billing-date"/>
   <element type="total-cost"/>
   <element type="total-number-calls"/>
   <element type="total-time"/>
   <element type="detail-list"/>
 </ElementType>
</Schema>
```

FIG. 5A

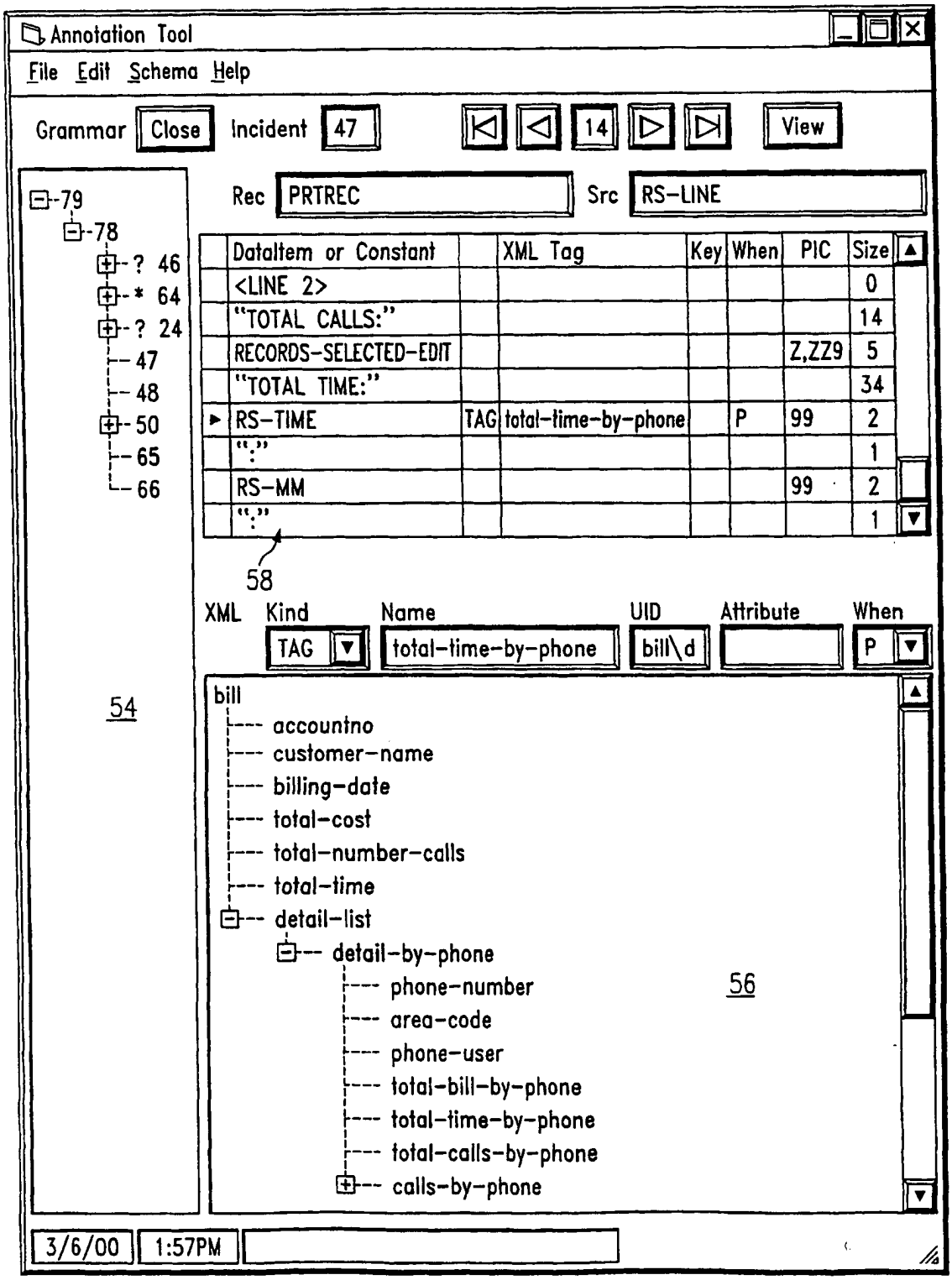

Annotation Tool
File Edit Schema Help
Grammar Close Incident 47 14 View
-79
-78
? 46
* 64
? 24
47
48
50
65
66
Rec PRTREC Src RS-LINE
DataItem or Constant | XML Tag | Key | When | PIC | Size
<LINE 2> | 0
"TOTAL CALLS:" | 14
RECORDS-SELECTED-EDIT | Z,ZZ9 | 5
"TOTAL TIME:" | 34
RS-TIME | TAG | total-time-by-phone | P | 99 | 2
":" | 1
RS-MM | 99 | 2
":" | 1
58
XML Kind TAG Name total-time-by-phone UID bill\d Attribute When P
54
bill
accountno
customer-name
billing-date
total-cost
total-number-calls
total-time
detail-list
detail-by-phone
phone-number
area-code
phone-user
total-bill-by-phone
total-time-by-phone
total-calls-by-phone
calls-by-phone
56
3/6/00 1:57PM

FIG. 6

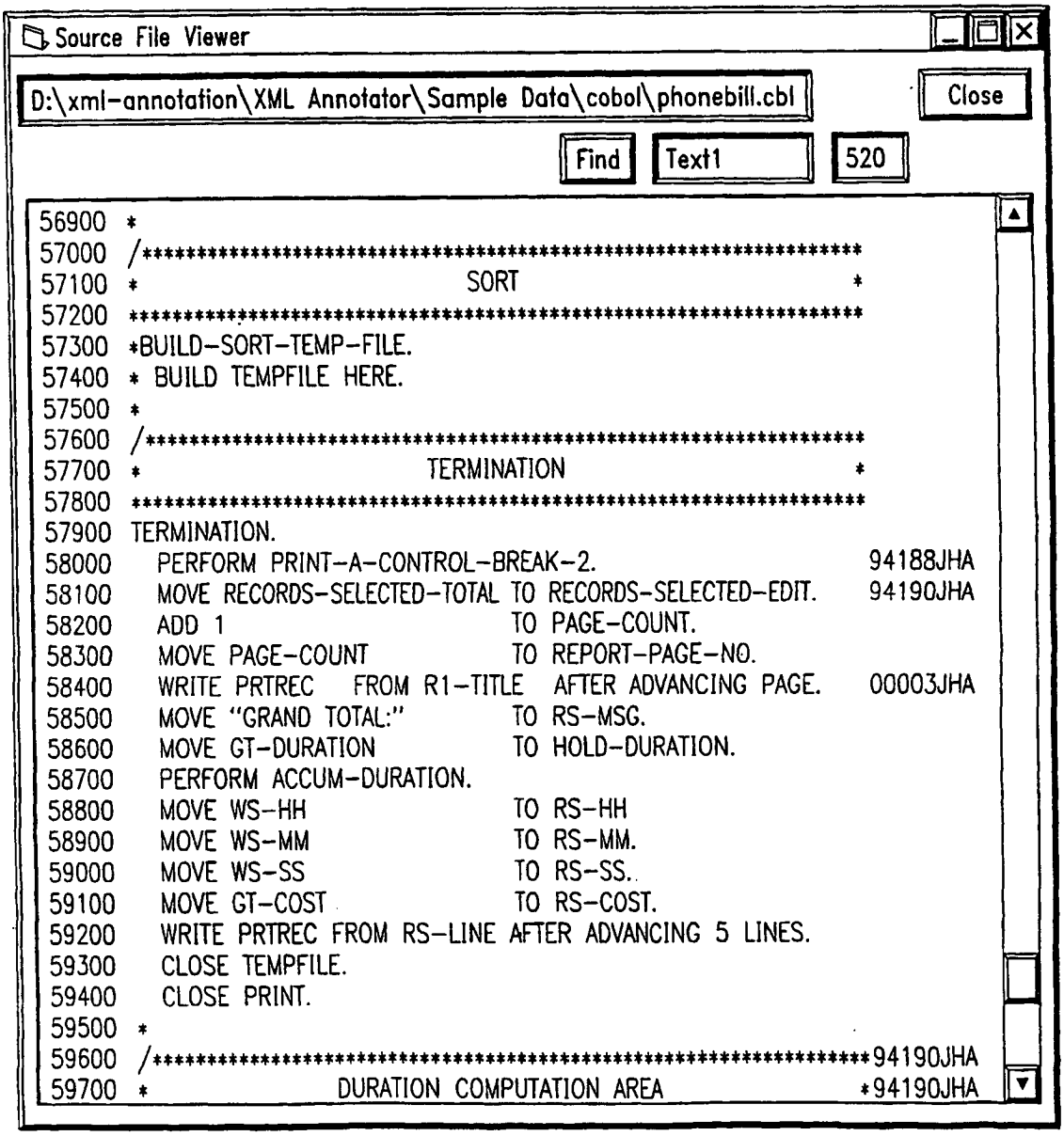

Source File Viewer
D:\xml-annotation\XML Annotator\Sample Data\cobol\phonebill.cbl
Close
Find
Text1
520
56900 *
57000 /***
57100 * SORT *
57200 ***
57300 *BUILD-SORT-TEMP-FILE.
57400 * BUILD TEMPFILE HERE.
57500 *
57600 /***
57700 * TERMINATION *
57800 ***
57900 TERMINATION.
58000 PERFORM PRINT-A-CONTROL-BREAK-2. 94188JHA
58100 MOVE RECORDS-SELECTED-TOTAL TO RECORDS-SELECTED-EDIT. 94190JHA
58200 ADD 1 TO PAGE-COUNT.
58300 MOVE PAGE-COUNT TO REPORT-PAGE-NO.
58400 WRITE PRTREC FROM R1-TITLE AFTER ADVANCING PAGE. 00003JHA
58500 MOVE "GRAND TOTAL:" TO RS-MSG.
58600 MOVE GT-DURATION TO HOLD-DURATION.
58700 PERFORM ACCUM-DURATION.
58800 MOVE WS-HH TO RS-HH
58900 MOVE WS-MM TO RS-MM.
59000 MOVE WS-SS TO RS-SS.
59100 MOVE GT-COST TO RS-COST.
59200 WRITE PRTREC FROM RS-LINE AFTER ADVANCING 5 LINES.
59300 CLOSE TEMPFILE.
59400 CLOSE PRINT.
59500 *
59600 /***94190JHA
59700 * DURATION COMPUTATION AREA *94190JHA

FIG. 6A

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <AttributeType name="Current" content="boolean"/>
 <ElementType name='Name" content="textonly"/>
 <ElementType name="ID" content="textonly"/>
 <ElementType name="Street" content="textonly"/>
 <ElementType name="City" content="textonly"/>
 <ElementType name="State" content="textonly"/>
 <ElementType name="ZIP" content="textonly"/>
 <ElementType name="Address">
    <Attribute type="Current"/>
    <Element type="Street"/>
    <Element type="City"/>
    <Element type="State"/>
    <Element type="ZIP"/>
 </ElementType>
 <ElementType name="Customer">
    <Element type="Name"/>
    <Element type="Address"/>
    <Element type="ID"/>
 </ElementType>
</Schema>
```

FIG. 7

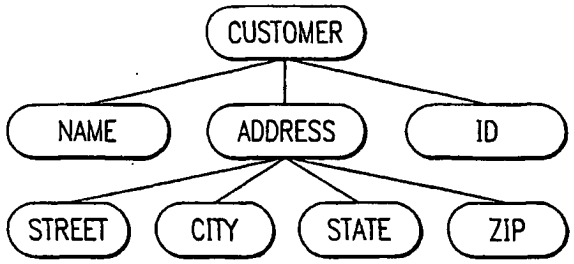


FIG. 7A

ID	Bezeichnung	Ende-Bezeichnung	Tiefe	Eltern	Kardinalität
1	<Customer>	</Customer>	1	0	n
2	<Name>	</Name>	2	1	1
3	<Address>	</Address>	2	1	1
4	<ID>	</ID>	2	1	1
5	<Street>	</Street>	3	3	1
6	<City>	</City>	3	3	1
7	<State>	</State>	3	3	1
8	<ZIP>	</ZIP>	3	3	1

FIG. 7B

FIG. 8

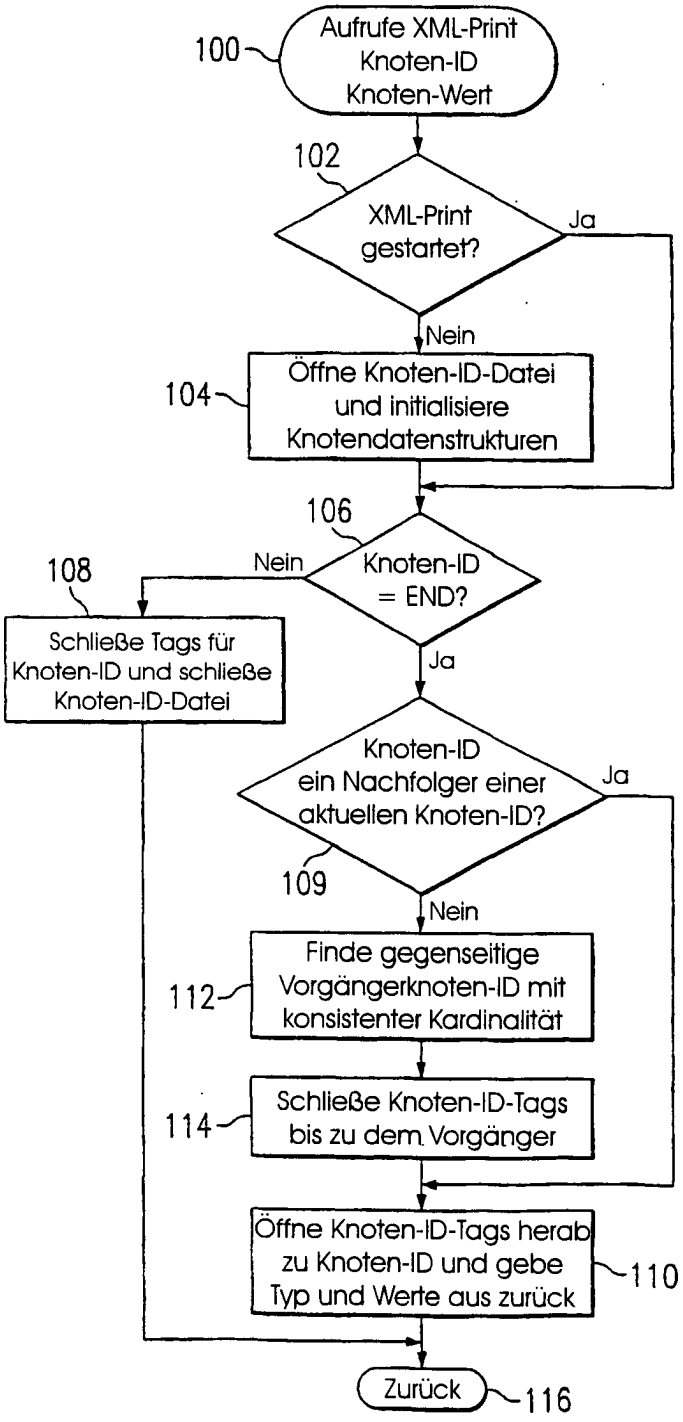
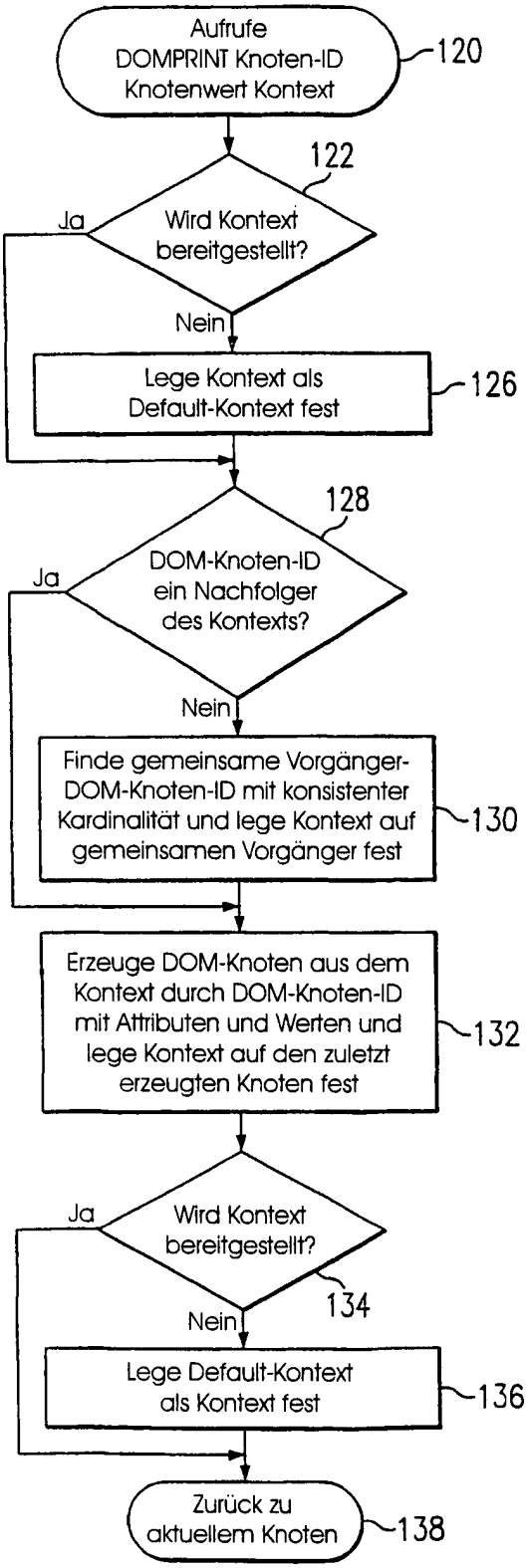

100
Aufrufe XML-Print
Knoten-ID
Knoten-Wert
102
XML-Print gestartet?
Ja
Nein
104
Öffne Knoten-ID-Datei und initialisiere Knotendatenstrukturen
106
Knoten-ID = END?
Nein
108
Schließe Tags für Knoten-ID und schließe Knoten-ID-Datei
Ja
Knoten-ID ein Nachfolger einer aktuellen Knoten-ID?
Ja
109
Nein
112
Finde gegenseitige Vorgängerknoten-ID mit konsistenter Kardinalität
114
Schließe Knoten-ID-Tags bis zu dem Vorgänger
Öffne Knoten-ID-Tags herab zu Knoten-ID und gebe Typ und Werte aus zurück
110
Zurück
116

FIG. 9


Aufrufe
DOMPRINT Knoten-ID
Knotenwert Kontext
120
122
Wird Kontext bereitgestellt?
Ja
Nein
Lege Kontext als Default-Kontext fest
126
128
DOM-Knoten-ID ein Nachfolger des Kontexts?
Ja
Nein
Finde gemeinsame Vorgänger-DOM-Knoten-ID mit konsistenter Kardinalität und lege Kontext auf gemeinsamen Vorgänger fest
130
Erzeuge DOM-Knoten aus dem Kontext durch DOM-Knoten-ID mit Attributen und Werten und lege Kontext auf den zuletzt erzeugten Knoten fest
132
Wird Kontext bereitgestellt?
Ja
Nein
134
Lege Default-Kontext als Kontext fest
136
Zurück zu aktuellem Knoten
138