# Eine verteilte Infrastruktur für typ- und diensterweiterbare orthogonale Digitale Bibliotheken

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Biologie und Informatik
der Johann Wolfgang Goethe-Universität
in Frankfurt am Main

von
Christian Alexander Mönch
aus Aachen

Frankfurt 2002
(D F 1)

vom Fachbereich Biologie und Informatik der
Johann Wolfgang Goethe-Universität als Dissertation angenommen.

| | |
|---|---|
| Dekan: | Prof. Dr. Bruno Streit |
| Gutachter: | Prof. Dr. Oswald Drobnik<br>Prof. Dr. Kurt Geihs |
| Datum der Disputation: | 28. Januar 2002 |

# Danksagung

Die vorliegende Arbeit entstand zum größten Teil während meiner Tätigkeit als wissenschaftlicher Mitarbeiter an der Professur für Telematik des Fachbereichs Biologie und Informatik der Johann Wolfgang Goethe-Universität Frankfurt am Main.

Meinen besonderen Dank möchte ich Prof. Dr. Oswald Drobnik aussprechen, der die vorliegende Arbeit betreut hat. Seine Ratschläge und Diskussionbeiträge haben wesentlich zum Gelingen dieser Arbeit beigetragen.

Bei Prof. Dr. Kurt Geihs möchte ich mich für die Übernahme des Zweitgutachtens bedanken.

Darüber hinaus danke ich den Kollegen an der Professur, insbesondere Martin Heß, für die fruchtbaren Diskussionen und für das gute Arbeitsklima.

Danken möchte ich auch Magdalena Feldhoffer, deren fachliche Hinweise und Anregungen für mich sehr wertvoll waren.

Meiner Mutter danke ich sehr für ihre großzügige Unterstützung und ihr unerschütterliches Vertrauen in meine Fähigkeiten.

Ganz herzlich bedanke ich mich bei Frauke, die mich mit großem Verständnis und grenzenloser Geduld begleitet und in vielfältiger Weise während der Erstellung dieser Arbeit unterstützt hat.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Programmverzeichnis

# Verzeichnis der Listings

# 1. Einleitung

## 1.1. Digitale Bibliotheken

Die Versorgung mit Wissen ist von zentraler Bedeutung für unserer Gesellschaft und unabdingbare Voraussetzung für erfolgreiche wissenschaftliche Arbeit. Die Gewährleistung der Wissensversorgung erfolgt zu einem großen Teil durch Bibliotheken, die die Aufgaben der Speicherung und Bereitstellung des zur Verfügung stehenden Wissens übernehmen. Allerdings stoßen traditionelle Bibliotheken bei der Ausführung dieser Aufgaben zunehmend an ihre Grenzen. So führt die große Zahl heute lebender Wissenschaftler zu einem enormen jährlichen Zuwachs an Dokumenten, die nicht nur gespeichert sondern auch erschlossen und möglichst schnell bereitgestellt werden müssen. Gleichzeitig entstehen aufgrund der weiten Verbreitung informationsverarbeitender Technologien immer mehr Dokumente, die elektronisch veröffentlicht werden, also lediglich in digitaler Form vorliegen und gleichberechtigt zur Wissensmenge beitragen.

Die ständig wachsende Zahl an wissenschaftlichen Veröffentlichungen, die Notwendigkeit zum schnellen Zugriff auf Dokumente und das Auftreten elektronisch publizierter Dokumente haben den Bedarf an Digitalen Bibliotheken geweckt. Digitale Bibliotheken sind Bibliotheken, deren Primärdaten in digitaler Form vorliegen (vgl. [65]). Sie bieten eine Reihe von Vorteilen gegenüber traditionellen Bibliotheken (vgl. [24, 49]). So lassen sich digitale Informationen auf wesentlich kleinerem Raum speichern, sie lassen sich schneller übertragen, sie sind weltweit verfügbar, sie ermöglichen die integrierte Darstellung unterschiedlicher Medien und sie erlauben den Einsatz automatischer Verfahren zur inhaltsbasierten Suche.

## 1.2. Entwicklungstendenzen

In der jüngeren Vergangenheit sind Entwicklungstendenzen sichtbar geworden, die einen großen Einfluß auf die Realisierung Digitaler Bibliotheken besitzen. So verfügen existierende Rechnergenerationen über ein Leistungspotential, das die effiziente Verarbeitung von zeitdiskreten sowie zeitkontinuierlichen Medien erlaubt. Dies ermöglicht die Erschließung neuer Anwendungsfelder und damit einhergehend die Entwicklung neuer Medien, wie Hypertext oder Hypermedia, begleitet von einer ständigen Weiterentwicklung der verwendeten Repräsentationsformate. Einige Beispiele für die schnelle Entwicklung in den letzten Jahren sind die Hypertext Markup Language (HTML) [96], die Virtual Reality Modeling Language (VRML) [43] und die Extensible Markup Language (XML) [97]. Da es sich bei XML um eine Metasprache mit der

Fähigkeit zur Definition neuer Auszeichnungselemente handelt, ist zu erwarten, daß eine große Zahl neuer spezialisierter Formate, die auch neue Medien erschließen werden, entstehen. Daß diese Entwicklung bereits begonnen hat, läßt sich an der Entstehung von MathML 2.0 [100], einem XML-basierten Austauschformat für Formeln, an der Entstehung der Beschreibungssprache für Scalable Vector Graphics (SVG) [98], an der Definition der Synchronized Multimedia Integration Language (SMIL) [101] sowie an der Entstehung von Extensible 3D (X3D) [104], einer XML-Kodierung für die VRML, ablesen.

Neben der Rechenleistung der zur Verfügung stehenden Systeme ist die zunehmende Vernetzung von Systemen maßgeblich für die Entstehung neuer Medien und Dokumente, deren Nutzung die Existenz eines Kommunikationsnetzwerks voraussetzt. Die Inhalte dieser Online-Dokumente (vgl. [24, S. 19]) sind nicht mehr statisch festgelegt, sondern werden dynamisch aus entfernten Quellen zusammengesetzt. Beispiele dafür sind Nachrichtenticker, kooperativ bearbeitbare Dokumente und Konferenzsysteme. Aufgrund der hier geschilderten Entwicklungen muß davon ausgegangen werden, daß in Zukunft eine große Zahl unterschiedlicher Dokumenttypen und Dokumentformate entstehen werden, die sich durch zunehmende strukturelle Komplexität und Dynamik auszeichnen werden (vgl. [25]).

Die Möglichkeiten der rechnergesteuerten Verarbeitung führen zu einer ständigen Entwicklung neuer Dienste, wie z. B. inhaltsbasierte Suchdienste, Profildienste oder Benachrichtungsdienste (vgl. [24, S. 82 f]). Auch hier ist eine Beschleunigung der Entwicklung zu erwarten.

Die fortschreitende Integration und Miniaturisierung der Mikroelektronik begünstigt darüber hinaus die Entstehung einer Vielfalt unterschiedlicher Geräte zur Informationsverarbeitung. Ausgerüstet mit den Mitteln zur drahtlosen Kommunikation eignen sich diese Geräte zum mobilen Zugriff auf die Ressourcen Digitaler Bibliotheken (vgl. [54, 23]). Diese Klasse von Geräten zeichnet sich aufgrund der kurzen Innovationszyklen und der hohen Innovationsrate durch eine große Heterogenität in Bezug auf Plattformen und Fähigkeiten aus.

Insgesamt läßt sich feststellen, daß die grundlegenden Elemente, mit denen Digitale Bibliotheken umgehen, einer permanenten Änderung unterworfen sind. Die Zahl der digitalen Dokumente sowie die Zahl der Anwender steigt kontinuierlich. Es entwickeln sich fortwährend neue Dokumenttypen, neue Medien und neue Dienste. Darüber hinaus entwickeln sich neue Rechnerplattformen, die zur Realisierung Digitaler Bibliotheken beitragen werden, sei es auf der Seite der Betreiber oder der Nutzer. Durch die Verwendung unterschiedlicher Plattformen wird diese Entwicklung zwangsläufig zu heterogenen Umgebungen führen, in denen Digitale Bibliotheken ihre Funktionalität bereitstellen müssen.

Um den Schutz der notwendigen hohen Investitionen zur Realisierung Digitaler Bibliotheken gewährleisten zu können, müssen Digitale Bibliotheken ihre Dienste über einen langen Zeitraum zur Verfügung stellen. Daher muß schon bei der Konzeption Digitaler Bibliotheken der Möglichkeit zur Adaption an die hier aufgeführten Entwicklungen Rechnung getragen werden.

## 1.3. Präzisierung der Problemstellung

Unter einer Digitalen Bibliothek wird in der vorliegenden Arbeit ein System verstanden, das die Speicherung einer unbegrenzten Zahl von Dokumenten unterschiedlicher Typen ermöglicht und eine unbegrenzte Zahl von Anwendern bei der Durchführung von Operationen und Diensten auf diesen Dokumenten unterstützt (vgl. [24, S. 4 f]). Unter Operationen werden hier Manipulationen oder Transformationen eines Dokuments verstanden, unter einem Dienst wird die Manipulation oder Transformation einer Menge von Dokumenten verstanden (vgl. [24]). Ein Beispiel für eine Operation ist die Präsentation eines digitalen Dokuments, ein Beispiel für einen Dienst ist die Suche über einer Dokumentsammlung.

Aufgrund der in Abschnitt 1.2 beschriebenen Entwicklungen müssen Digitale Bibliotheken so flexibel gestaltet werden, daß sie die Integration neuer Dokumenttypen und Dienste während ihrer Laufzeit erlauben, ohne daß dabei die Funktionalität der Digitalen Bibliothek beeinträchtigt wird. Insgesamt lassen sich aus diesem Ziel fünf Anforderungen an Digitale Bibliotheken ableiten: Dezentralität, Orthogonalität, dynamische Erweiterbarkeit um neue Dokumenttypen, dynamische Erweiterbarkeit um neue Dienste und Plattformunabhängigkeit.

**Dezentralität** Skalierbarkeit ist eine wesentliche Forderung an Digitale Bibliotheken. Eine Digitale Bibliothek muß in der Lage sein, alle zukünftigen digitalen Dokumente zu speichern und einer unbegrenzten Zahl von Anwendern den Zugriff auf diese Dokumente und auf die Dienste der Digitalen Bibliothek zu ermöglichen.

Die Konsequenz aus dieser Forderung ist eine dezentrale, verteilte Architektur Digitaler Bibliotheken, denn jede zentrale Komponente würde die Skalierbarkeit einer Digitalen Bibliothek letztlich begrenzen (vgl. [6, 26]).

**Orthogonalität** Die angestrebte große Zahl von Dokumenten setzt die Möglichkeit zu rechnergesteuerten Ausführung von Operationen und Diensten voraus, denn eine manuelle Bearbeitung kann aufgrund des zu erwartenden Umfangs der Dokumentmenge nicht geleistet werden (vgl. [7, 106, 83]). Eine rechnergesteuerte Ausführung von Operationen bedingt, daß Operationen unabhängig vom Format oder Inhalt eines Dokuments auf einheitliche Art und Weise ausgeführt werden können, denn andernfalls wäre eine manuelle Intervention zur Identifikation der notwendigen Prozesse zur Operationsausführung notwendig.

Die Digitale Bibliothek muß daher eine Schnittstelle zur Ausführung von Operationen und Diensten bereitstellen, die unabhängig von den Unterschieden zwischen verschiedenen Dokumenten, also von deren Format und deren Inhalt ist. Die Möglichkeit zur einheitlichen Ausführung von Operationen und Diensten auf den Dokumenten der Digitalen Bibliothek wird mit dem Begriff *Orthogona-*

*lität* bezeichnet.[1] Aufbauend auf den Operationen in einer orthogonalen Digitalen Bibliothek lassen sich dann format- und typunabhängige Dienste realisieren.

**Transparente Erweiterung um neue Dokumenttypen** Um ihre Aufgaben erfüllen zu können, müssen Digitale Bibliotheken die Integration sich neu entwickelnder Dokumenttypen unterstützen (vgl. [65, 60]). Die Integration neuer Dokumenttypen umfaßt die Speicherung der Dokumente sowie die Möglichkeit zur Ausführung von Operationen und Diensten auf diesen Dokumenten.

Ein wesentlicher Aspekt der Integration neuer Dokumenttypen ist die Integration der für die Durchführung von Operationen notwendigen Prozesse in die Digitale Bibliothek. Dabei muß die Orthogonalität der Digitalen Bibliothek gewahrt bleiben, um Dokumente neuen Typs in die rechnergesteuerte Ausführung von Operationen und Diensten einbeziehen zu können.

Die Erweiterung der Digitalen Bibliothek um neue Dokumenttypen unter Wahrung der Orthogonalität wird im folgenden als *transparente Typerweiterbarkeit* bzw. *transparente Erweiterbarkeit* bezeichnet.

**Transparente Erweiterung um neue Dienste** Neben der Integration neuer Dokumenttypen muß die Integration neuer Dienste in die Digitale Bibliothek möglich sein, um die Langlebigkeit der Digitalen Bibliothek zu gewährleisten (vgl. [105]). Neu integrierte Dienste müssen alle bereits in der Digitalen Bibliothek existierenden sowie zukünftig zu integrierenden Dokumente unterstützen.

Ein weiterer wesentlicher Aspekt der Integration neuer Dienste ist die Integration der für die Durchführung des Dienstes notwendigen Prozesse in die Digitale Bibliothek. Dabei muß die Eigenschaft der Dezentralität gewahrt bleiben.

Die Integration neuer Dienste in die Digitale Bibliothek unter Sicherstellung der genannten Eigenschaften wird als *transparente Dienstintegration* bzw. *transparente Diensterweiterung* bezeichnet.

**Plattformunabhängigkeit** Die Skalierbarkeit einer Digitalen Bibliothek kann letztlich nur durch eine Erweiterung der Kapazitäten der Digitalen Bibliothek in Form der Integration neuer Rechnerknoten erreicht werden. Die schnelle Entwicklung neuer Plattformen wird daher zwangsläufig zu einem heterogenen Rechnerumfeld der Digitalen Bibliothek führen.

Diese Heterogenität darf die Funktion der Digitalen Bibliothek nicht negativ beeinflussen. Insbesondere muß die Durchführung von Operationen und Diensten auf den existierenden Dokumenten sichergestellt werden.

[1] Der Begriff „Orthogonal“ wird hier analog zu dem Begriff des orthogonalen Befehlssatzes eines Prozessors verwendet. Er wurde gewählt, um zu verdeutlichen, daß die Operationen und Dienste in einer orthogonalen Digitalen Bibliothek unabhängig von der Art der Dokumente sind, in dem Sinn, daß jede Operation auf jedem Dokument und jeder Dienst auf allen Dokumenten der Digitalen Bibliothek in der gleichen Weise ausgeführt werden kann.

## 1.4. Ziele und Vorgehensweise

Existierende Systeme zur Verarbeitung digitaler Dokumente unterstützten die transparente Erweiterung um neue Dokumenttypen und Dienste nur ungenügend. Die Integration neuer Dokumenttypen kann meist nur unter Einschränkungen der Skalierbarkeit oder Verletzung der Orthogonalität vorgenommen werden. Die Erweiterung existierender Systeme zur Verarbeitung digitaler Dokumente um neue Dienste führt häufig zu zentralen, und infolge dessen nicht skalierbaren Strukturen.

Das Ziel dieser Arbeit liegt daher in der Entwicklung einer verteilten Infrastruktur zur Realisierung skalierbarer erweiterbarer orthogonaler Digitaler Bibliotheken. Die Infrastruktur soll die Entwicklung Digitaler Bibliotheken ermöglichen, die folgende Anforderungen erfüllen:

- Speicherung einer unbegrenzten Menge an Dokumenten und Unterstützung einer unbegrenzten Menge an Benutzern bei der Ausführung von Operationen und Diensten auf den gespeicherten Dokumenten.
- Bereitstellung von orthogonalen Schnittstellen zur rechnergesteuerten Verarbeitung von Dokumenten.
- Unterstützung transparenter Typ- und Diensterweiterungen.
- Unterstützung der Integration neuer Plattformen.

Zum Erreichen dieser Ziele wird wie folgt vorgegangen. Zunächst wird in Kapitel 2 ein Modell für die Verarbeitung digitaler Dokumente in erweiterbaren orthogonalen Digitalen Bibliotheken entwickelt. Auf der Grundlage dieses Modells erfolgt dann eine Untersuchung existierender Systeme zur Verarbeitung digitaler Dokumente im Hinblick darauf, inwieweit sie die genannten Anforderungen erfüllen. Anhand der Ergebnisse dieser Untersuchung werden diejenigen Faktoren identifiziert, die jeweils die Erfüllung der Anforderungen verhindern.

In Kapitel 3 wird ein Lösungsansatz zur Behebung der erkannten Defizite vorgestellt. Auf der Grundlage dieses Vorschlags wird dann eine verteilte Infrastruktur zur Realisierung skalierbarer erweiterbarer orthogonaler Digitalen Bibliotheken entworfen.

Kapitel 4 geht auf die Struktur der Dokumente in dieser Infrastruktur ein. Nachdem beschrieben wird, wie Dokumente als Muster zur Erstellung neuer Dokumenttypen verwendet werden können, werden Dokumentmuster zur Realisierung verschiedener Dokumenttypen vorgestellt. In Kapitel 5 werden Mechanismen zur Integration neuer Dienste in die Infrastruktur diskutiert. Dazu werden mobile Dokumente konstruiert, die eine flexible Verteilung der Dienstlogik in der Infrastruktur erlauben. An Beispielen wird die Implementierung von Katalog- und Suchdiensten beschrieben.

Eine Realisierung der entwickelten Infrastruktur wird in Kapitel 6 vorgestellt. Dabei wird auf die Basismechanismen der Infrastruktur und auf die Realisierung unterschiedlicher Dokumenttypen eingegangen. Abschließend faßt Kapitel 7 die Ergebnisse der Arbeit zusammen und gibt Anregungen für mögliche Erweiterungen.

# 2. Erweiterbare orthogonale Digitale Bibliotheken

Existierende Systeme zur Verarbeitung digitaler Dokumente erfüllen die in Abschnitt 1.3 genannten Anforderungen nur unvollständig. Um die Ursachen dafür ermitteln zu können bzw. die Konsequenzen abschätzen zu können, die die Anforderungen für die Architektur Digitaler Bibliotheken besitzen, wird im vorliegenden Abschnitt ein Modell Digitaler Bibliotheken entworfen, das die grundlegenden Mechanismen orthogonaler erweiterbarer Digitaler Bibliotheken identifiziert. Auf der Grundlage dieses Modells wird anschließend eine Untersuchung existierender Systeme vorgenommen.

## 2.1. Ein Modell Digitaler Bibliotheken

In Kapitel 1 wurde eine Digitale Bibliothek bereits als ein System beschrieben, das eine unbegrenzte Zahl digitaler Dokumente unterschiedlicher Formate und Medien speichert und einer unbegrenzten Zahl an Anwendern die Durchführung von Operationen und Diensten auf diesen Dokumenten ermöglicht. Im folgenden werden die Begriffe digitale Dokumente, Operationen und Dienste näher erläutert. Die Beschreibungen digitaler Dokumente und Operationen orientieren sich an [24], [65] und [39] die Beschreibung der Dokumenttypen orientiert sich an [88].

### 2.1.1. Digitale Dokumente

Die in einer Digitalen Bibliothek gespeicherten Daten liegen in Form von digitalen Dokumenten vor. Ein *digitales Dokument* ist eine in sich abgeschlossene, individuell adressierbare Informationseinheit, deren Inhalt digital kodiert und auf einem elektronischen Datenträger gespeichert ist, so daß das Dokument rechnergesteuert verarbeitet werden kann (vgl. [24, S. 15]).

Die in einem digitalen Dokument gespeicherten Informationen werden in unterschiedlicher Form kodiert.[1] Diese Kodierung wird im folgenden als Dokumentkodierung oder Dokumentformat bezeichnet. Beispiele für Kodierungen sind die Kodierung eines Buchstabens in ASCII oder die PCM-Kodierung von Audio-Informationen (vgl. [88, S. 11]). Digitale Dokumente können eine Hierarchie von Kodierungen besitzen (vgl. [24, S. 25]), deren unterste Ebene immer eine Folge von Bits ist. Ein Bei-

[1] Der Begriff „Kodierung“ wird hier analog zu dem Begriff „Repräsentationsmedium“ in [88, S. 10 f] verwendet. Auf den Terminus Medium wurde hier bewußt verzichtet, um Assoziationen mit Präsentationsmedien zu vermeiden.

spiel für eine solche Hierarchie bildet das Resource Description Framework (RDF), ein Format zur Beschreibung von Relationen. RDF-Dokumente werden in der Extensible Markup Language (XML), einem Format zur Kodierung von Baumstrukturen mit benannten Knoten, notiert. XML wiederum wird als Folge von ASCII-Zeichen repräsentiert, die letztlich als Bit-Folge dargestellt werden können. Die Bit-Folge wird als *digitales Objekt* bezeichnet (vgl. [44]). Jedes digitale Dokument läßt sich, unabhängig von Art und Umfang der Kodierungshierarchie als Bit-Folge darstellen. Auf dieser Ebene ist eine einheitliche Verarbeitung, z. B. Kopieren, aller Dokumente möglich. Diese Verarbeitung kann sich natürlich nur auf die allen Dokumenten gemeinsamen Eigenschaften beziehen und keine spezifischen Formate berücksichtigen.

Eine Verarbeitung eines Dokuments, die über die Verarbeitung von Bit-Folgen hinausgeht, also eine höhere Ebene in der Kodierungshierarchie interpretiert, benötigt zweierlei Informationen:

- Information über die Kodierungshierarchie des zu verarbeitenden Dokuments, d. h. welche Kodierungen im Dokument verwendet werden.
- Informationen über die jeweiligen Kodierungen, d. h. wie die einzelnen Kodierungen zu interpretieren sind.

Diese Informationen werden im folgenden als *Strukturkenntnis* bezeichnet. Die zur Verarbeitung von Dokumenten notwendige Strukturkenntnis muß zusätzlich zum Dokumentinhalt bereitgestellt werden, da sie sich nicht rechnergesteuert aus dem Inhalt eines digitalen Dokuments ermitteln läßt, wie das folgende Beispiel verdeutlicht (vgl. [24, S. 52 f]). Die Byte-Folge 50, 56, 48, 56, 49, 55, 52, 57 läßt sich mit Kenntnis des ASCII-Codes in den Text 28081749 umwandeln, der sich, unter Annahme eines Datumsformats, als das Datum 28. 8. 1749, den Geburtstag von Johann Wolfgang Goethe interpretieren läßt. Es gibt aber weitere sinnvolle Interpretation, z. B. die Interpretation der Ziffernfolge 28081749 als Betragswert (28.081.749,- DM) oder als geographische Ortsangabe (28 08 N 17 49 W). Welche Interpretation korrekt ist kann nicht allein anhand der Bit-Folge entschieden werden.[2]

Digitale Dokumente lassen sich also zusammenfassend durch folgende Eigenschaften charakterisieren:

- Digitale Dokumente besitzen eine hierarchische Struktur von Formaten, auf deren unterster Stufe eine Folge von Bits steht.
- Die Verarbeitung digitaler Dokumente erfordert für jede Hierarchiestufe spezifische Strukturkenntnis. Die Strukturkenntnis zur Verarbeitung der Bit-Ebene ist in allen Rechnerknoten vorhanden.
- Die notwendige Strukturkenntnis kann nicht aufgrund des Dokumentinhalts ermittelt werden, sondern muß extern bereitgestellt werden.

[2] Es gibt Formate, wie z. B. XML und SGML, die als selbstenthaltene Formate bezeichnet werden, da sie eine Möglichkeit zur Beschreibung ihrer eigenen Kodierung beinhalten, so daß keine externe Kenntnis über das Format notwendig zu sein scheint. Die Interpretation dieser Formate erfordert jedoch die Kenntnis, daß die Bit-Folge ein XML- oder SGML-Dokument darstellt.

### 2.1.2. Speicherkomponenten

Eine Speicherkomponente ist eine Softwarekomponente, die digitale Dokumente speichert. Sie ermöglicht anderen Komponenten Zugriff auf die von ihr gespeicherten Dokumente. Speicherkomponenten stellen dazu eine generische Schnittstelle, die *Speicherschnittstelle* zum Zugriff auf die in ihnen gespeicherten Dokumente zur Verfügung. Diese Schnittstelle stellt die digitalen Dokumente als Bit-Folgen bereit, also in Form digitaler Objekte. Die Ressourcen einer einzelnen Speicherkomponente, z. B. die Zahl der von ihr gespeicherten digitalen Dokumente, sind begrenzt. Alle Speicherkomponenten zusammen bilden den Speicher der Digitalen Bibliothek. Eine Skalierung der Speicherkapazität der Digitalen Bibliothek erfolgt durch die Bereitstellung zusätzlicher Speicherkomponenten. Die hier gegebene Definition der Speicherkomponenten lehnt sich an die in [44] beschriebenen Repositorien an.

### 2.1.3. Anwenderkomponenten

Anwenderkomponenten sind Softwarekomponenten, mittels derer ein Anwender die Ausführung von Operationen oder Diensten initiieren. Da jede Operation und jeder Dienst eine individuelle Schnittstelle bereitstellen kann (siehe Abschnitt 2.1.4 und Abschnitt 2.1.5), müssen Anwenderkomponenten über je eine Schnittstelle für jede Operation und jeden Dienst verfügen. Anwender der Digitalen Bibliothek sind gegenüber den anderen Elementen der Digitalen Bibliothek durch Anwenderkomponenten repräsentiert. Eine Skalierung der Anwenderzahl erfolgt durch die Bereitstellung zusätzlicher Anwenderkomponenten.

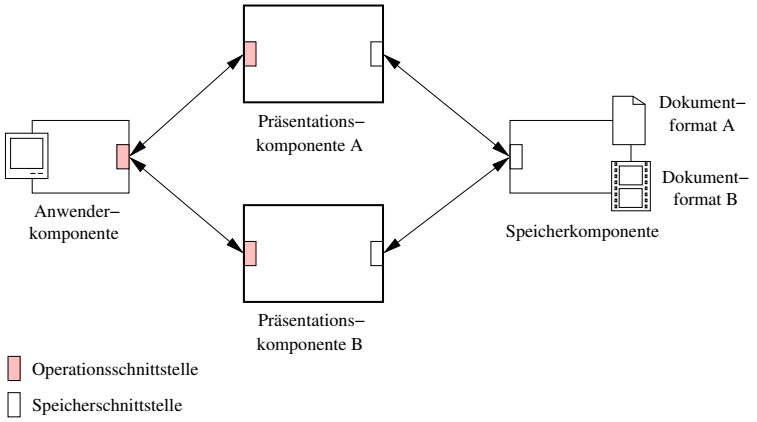
### 2.1.4. Operationen

Unter einer Operation wird die Manipulation oder Transformation eines digitalen Dokuments verstanden, für deren Durchführung Programmfunktionen verantwortlich sind (vgl. [24, S. 24]). Jede Operation ist durch ihre Schnittstellen charakterisiert. Alle Operationen verwenden die Speicherschnittstelle zum Zugriff auf digitale Dokumente. Darüber hinaus besitzen Operationen eine operationsspezifische *Operationsschnittstelle* zum Austausch von Daten mit Anwenderkomponenten. Beispiele für Operationen sind: Präsentieren eines Dokuments, Suche innerhalb eines Dokuments.

Die Durchführung der Operation besteht in der Abbildung zwischen der Speicherschnittstelle und der Operationsschnittstelle. Die Abbildung zwischen den beiden Schnittstellen ist spezifisch für die Operationsschnittstelle und die Kodierungshierarchie, u. U. sogar für den Inhalt des Dokuments, auf dem die Operation ausgeführt wird (vgl. Abschnitt 3.2.1.1). So ist z. B. die Präsentation eines ASCII-kodierten Textdokuments anders durchzuführen als die Präsentation eines EBDIC-kodierten Textdokuments oder eines JPEG-kodierten Bilddokuments.

Abbildung 2.1 illustriert die Durchführung einer Präsentations-Operation auf Dokumenten mit unterschiedlicher Kodierungshierarchie. Präsentationskomponente A enthält die Programmfunktionen zur Durchführung der Präsentation eines Textdokuments, während Präsentationskomponente B Programmfunktionen zur Präsentation eines Bilddokuments bereitstellt. Beide Komponenten nutzen die generische Speicher-

**Abbildung 2.1.** Operation in der Digitalen Bibliothek




schnittstelle zum Zugriff auf die Dokumentdaten und eine Operationsschnittstelle zur Kommunikation mit der Anwenderkomponente, diese kann z. B. im Zugriff auf geeignete Funktionsbibliotheken oder in der Verwendung des X11-Protokolls bestehen.

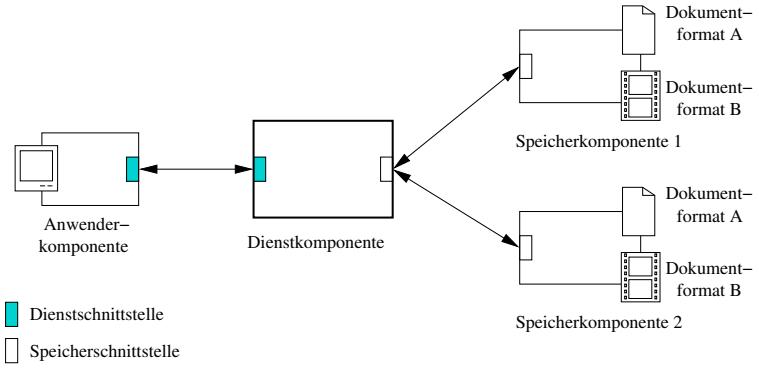
### 2.1.5. Dienste

Unter einem Dienst wird hier eine Manipulation oder Transformation der Elemente einer Teilmenge aller in einer Digitalen Bibliothek enthaltenen Dokumente verstanden. Dienste unterscheiden sich von Operationen also dadurch, daß sich ihre Berechnung nicht auf ein Dokument beschränkt, sondern mehrere Dokumente, bis hin zu allen Dokumenten der Digitalen Bibliothek umfassen kann. Beispiele für Dienste in einer Digitalen Bibliothek sind die, auch in traditionellen Bibliotheken existierenden Dienste Sammeln und Erschließen (vgl. [81, S. 15]) sowie die für Digitale Bibliotheken spezifischen Benachrichtigungs-, Profil- und Personalisierungsdienste (vgl. [39, 27], [24, S. 82–84]). Alle Dienste greifen über die Speicherschnittstelle auf digitale Dokumente zu und kommunizieren über die dienstspezifische *Dienstschnittstelle* mit Anwenderkomponenten. Analog zu Operationen sind Dienste durch ihre Dienstschnittstellen charakterisiert.

Die Erbringung des Dienstes besteht in der Abbildung zwischen Speicherschnittstellen und der Dienstschnittstelle. Die Abbildung zwischen der Speicherschnittstelle und der Dienstschnittstelle ist spezifisch für die Dienstschnittstelle sowie die Kodierungshierarchien, bzw. die Inhalte (vgl. Abschnitt 3.2.1.1), aller digitalen Dokumente. So ist beispielsweise die Suche in einem Textdokument anders durchzuführen als die Suche in einem Bilddokument.

Abbildung 2.2 illustriert die Erbringung eines Dienstes auf unterschiedlichen digitalen Dokumenten. Im Gegensatz zur Ausführung einer Operation verlangt die Erbringung eines Dienstes die Integration der Informationen aller beteiligten digitalen Dokumente, um ein Ergebnis bereitstellen zu können. Aus diesem Grund ist in Abbildung 2.2 die Erbringung des Dienstes durch eine monolithische Komponente dargestellt, die über Informationen über die Kodierungshierarchien aller Dokumente verfügt. Diese

**Abbildung 2.2.** Erbringung eines Dienstes




Komponente kann natürlich verteilt realisiert werden, sie muß aber Informationen über die Kodierungshierarchien aller Dokumente in der Digitalen Bibliothek besitzen.

In den folgenden Abschnitten wird anhand des Modells untersucht, welche Konsequenzen sich für Digitale Bibliotheken aus den Anforderungen Orthogonalität und Erweiterbarkeit ergeben.

## 2.2. Orthogonalität in Digitalen Bibliotheken

Auf die Anforderung Orthogonalität wurde in dem in Abschnitt 2.1 entworfenen Modell Digitaler Bibliotheken noch nicht eingegangen. Im vorliegenden Abschnitt wird daher erläutert, welche zusätzlichen Einschränkungen und Mechanismen zur Gewährleistung der Orthogonalität in einer Digitalen Bibliothek notwendig sind.

Orthogonalität erfordert neben der Möglichkeit zur einheitlichen Interpretation der Resultate einer Operationsausführung auch die Möglichkeit zur einheitlichen Durchführung von Operationen. Legt man das Modell zugrunde, sind drei Randbedingungen für die Gewährleistung der Orthogonalität maßgeblich:

- Die Operationsschnittstelle einer Operation muß verbindlich definiert werden, um eine rechnergesteuerte Initiierung der Operation sowie eine rechnergesteuerte Interpretation der Ausgabe durchführen zu können.
- Jede Operation muß auf jedem Dokument zur Verfügung stehen.
- Die Digitale Bibliothek muß eine Schnittstelle bereitstellen, die den Anwenderkomponenten die dokumentunabhängige Ausführung derjenigen Programmfunktionen ermöglicht, die zur Durchführung einer Operation notwendig sind.

Wenn diese Randbedingungen erfüllt sind, ist keinerlei manuelle Intervention zur Initiierung von Operationen oder zur Interpretation der Ergebnisse einer Operationsausführung mehr notwendig. Alle dokumentspezifischen Verarbeitungsschritte werden transparent für Anwenderkomponenten innerhalb der Digitalen Bibliothek ausgeführt.

Die Erfüllung der Randbedingungen erfordert, die zur Ausführung von Operationen auf unterschiedlichen Dokumenten notwendigen unterschiedlichen Programmfunktionen transparent innerhalb der Digitalen Bibliothek bereitzustellen. Dies geschieht durch die Verwendung einer Zuordnungsfunktion, die die Programmfunktionen identifiziert, die für die Ausführung einer Operation auf einem Dokument notwendig ist. Im einzelnen wurden folgende Elemente zur Bereitstellung orthogonaler Operationen in einer Digitalen Bibliothek identifiziert: orthogonale Operationen, Dokumentmethoden, Laufzeitumgebungen und die Zuordnungsfunktion. Diese Elemente werden in den folgenden Abschnitten näher beschrieben.

### 2.2.1. Orthogonale Operationen

Eine *orthogonale Operation* ist eine Operation, die vollständig definiert ist, und die auf allen Dokumenten der Digitalen Bibliothek ausgeführt werden kann. Die vollständige Definition einer orthogonalen Operation beinhaltet die Angabe des Operationsnamens, der Operationsschnittstelle und eine Beschreibung der Semantik der Operation. Dies geschieht analog zu den Forderungen, die in [47] an Services gestellt werden. Die Operationsschnittstelle einer orthogonalen Operation wird als *orthogonale Schnittstelle* bezeichnet.

### 2.2.2. Dokumentmethoden

Die Ausführung einer orthogonalen Operation auf einem Dokument erfordert die Durchführung entsprechender Programmfunktionen. Während orthogonale Operation dokumentunabhängig sind, sind die zu ihrer Durchführung benötigten Programmfunktionen spezifisch für einzelne Dokumente (siehe Abschnitt 2.1.1).

Die Programmfunktionen, die orthogonale Operationen auf Dokumenten durchführen, werden im folgenden als *Dokumentmethoden* oder einfach als Methoden bezeichnet. Jede Dokumentmethode führt eine orthogonale Operation auf allen digitalen Dokumenten mit der gleichen Kodierungshierarchie bzw. mit identisch verarbeitbarem Inhalt (vgl. Abschnitt 3.2.1.1) aus.

### 2.2.3. Laufzeitumgebungen

Dokumentmethoden werden in Laufzeitumgebungen instantiiert, die alle für die Ausführung der Methoden notwendigen Ressourcen, z. B. Speichermedien, Präsentationsmedien und Übertragungsmedien, bereitstellen. Laufzeitumgebungen unterscheiden sich voneinander in dem Programmcode, den sie ausführen können sowie in den Schnittstellen, über die sie den Dokumentmethoden Zugriff auf ihre Ressourcen erlauben. Aufgrund dieser Unterschiede sind Dokumentmethoden spezifisch für eine Laufzeitumgebung.

### 2.2.4. Zuordnungsfunktion

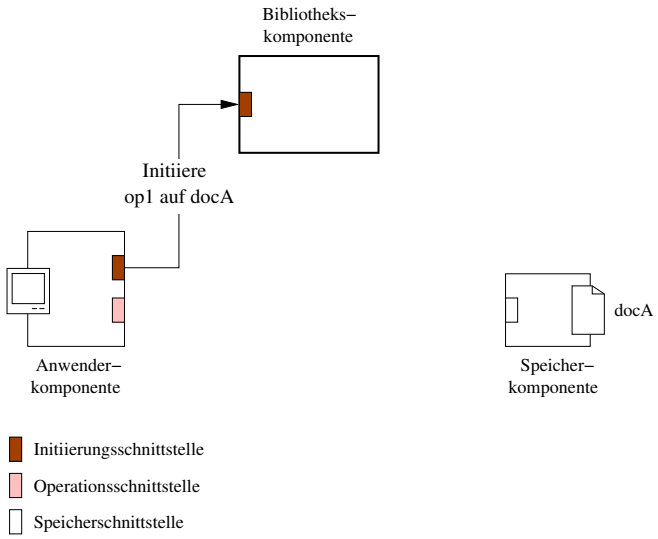
Orthogonalität erfordert, daß die Digitale Bibliothek die zur Ausführung einer Operation geeignete Dokumentmethode identifizieren kann. In Abschnitt 2.1.1 wurde deutlich, daß diese Identifikation nicht automatisch vorgenommen werden kann, sondern externe Informationen benötigt. Diese Informationen werden in orthogonalen Digitalen Bibliotheken in Form einer *Zuordnungsfunktion* bereitgestellt. Die Zuordnungsfunktion bildet eine Operation, ein Dokument und eine Laufzeitumgebung auf die Dokumentmethode ab, die die geforderte Operation auf dem angegebenen Dokument unter Verwendung der angegebenen Laufzeitumgebung realisieren kann.

### 2.2.5. Zusammenfassung

Das Modell für die orthogonale Verarbeitung von Dokumenten beinhaltet die in Abschnitt 2.1 genannten Elemente sowie die im vorliegenden Abschnitt definierten orthogonalen Operationen, Dokumentmethoden, Laufzeitumgebungen und die Zuordnungsfunktion. Die grundlegenden Konzepte zur Bereitstellung von Orthogonalität sind die einheitliche Definition von orthogonalen Operationen, ihre verbindliche Verfügbarkeit auf allen Dokumenten sowie eine transparente Zuordnung von Operationen, Dokumenten und Laufzeitumgebungen zu Methoden. Diese Zuordnung wird durch die Zuordnungsfunktion vorgenommen.

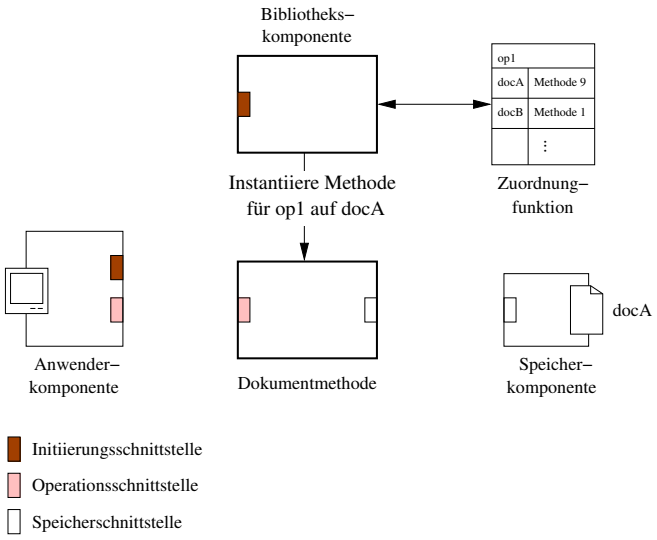
Ein Beispiel für die Ausführung einer orthogonalen Operation in einer orthogonalen Digitalen Bibliothek ist in Abbildung 2.5 dargestellt. Die Ausführung der Operation beginnt mit der Spezifikation des Dokuments und der Operation, die auf dem Dokument ausgeführt werden soll. Dazu wendet sich eine Anwenderkomponente an eine Komponente, die die Digitale Bibliothek repräsentiert und initiiert dort die Ausführung der Operation, wie Abbildung 2.3 dargestellt.

**Abbildung 2.3.** Initiierung einer orthogonalen Operation in einer orthogonalen Digitalen Bibliothek



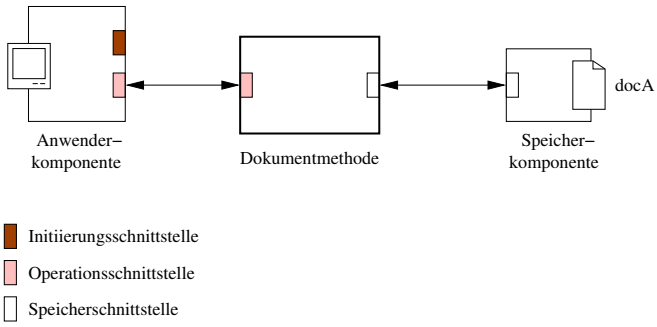
Im nächsten Schritt ermittelt die Bibliothekskomponente mit Hilfe der Zuordnungsfunktion die geeignete Dokumentmethode und instantiiert sie. Als Resultat der Instantiierung steht eine Komponente zur Verfügung, die die Dokumentmethode realisiert. Abbildung 2.4 illustriert diesen Schritt.

**Abbildung 2.4.** Instantiierung der Dokumentmethode




Der letzte Schritt zur Durchführung der Operation auf dem genannten Dokument besteht in der Ausführung der Dokumentmethode. Die Methodenkomponente kommuniziert dazu über die Speicherschnittstelle mit einer oder mehreren Speicherkomponenten und über die Anwenderschnittstelle mit der Anwenderkomponente. Dieser Vorgang ist in Abbildung 2.5 dargestellt.

**Abbildung 2.5.** Durchführung der orthogonalen Operation




Die Komponente zur Initiierung der Operationsausführung kann, wie in diesem Fall, identisch mit der Komponente zur Nutzung der Anwenderschnittstelle sein. Dies ist jedoch nicht zwingend.

Die Digitale Bibliothek ist gegenüber ihren Nutzern durch die orthogonalen Schnittstellen und die Dienstschnittstellen, die beide von den Anwenderkomponenten ge-

nutzt werden, sowie durch die Schnittstelle zur Initiierung Operationsausführung repräsentiert. Die Schnittstelle zur Initiierung von Operationen wird in Abschnitt 3.3.4 vorgestellt, die orthogonalen Schnittstellen sind in Abschnitt 3.3.2.1 beschrieben.

## 2.3. Erweiterung orthogonaler Digitaler Bibliotheken

Zwei weitere in Abschnitt 1.3 genannte und bisher nicht betrachtete Anforderungen an Digitale Bibliotheken sind: transparente Erweiterbarkeit um neue Dokumentformate und transparente Integration neuer Dienste. Die Forderung nach Orthogonalität bedingt, daß auch die Integration neuer Dokumente transparent durchgeführt werden muß, d. h. daß sie keine manuellen dokumentspezifischen Aktionen des Anwenders voraussetzen darf. Eine manuelle Bearbeitung durch die Anwender würde die Orthogonalität verletzen. Die manuelle Bearbeitung durch die Betreiber der Speicherkomponenten würde dagegen die Erweiterbarkeit und Skalierbarkeit der Digitalen Bibliothek einschränken (vgl. 2.5).

Im vorliegenden Abschnitt wird untersucht, welche Mechanismen für eine transparente Erweiterung einer orthogonalen Digitalen Bibliothek um neue Dokumentformate und neue Dienste notwendig sind.

### 2.3.1. Erweiterung um neue Dokumentformate

Die transparente Erweiterung einer orthogonalen Digitalen Bibliothek um neue Dokumentformate beinhaltet zwei Aspekte. Zum einen muß die Möglichkeit zur Durchführung orthogonaler Operationen auf Dokumenten des neuen Formats sichergestellt werden. Zum anderen muß die Bearbeitung der Dokumente mit neuen Formaten durch existierende Dienste gewährleistet sein.

#### 2.3.1.1. Bereitstellung orthogonaler Operationen

Die Integration neuer Dokumentformate in die Mechanismen zur Ausführung orthogonaler Operationen erfordert die Bereitstellung der entsprechenden Dokumentmethoden und die Erweiterung der Zuordnungsfunktion. Die Forderung nach transparenter Erweiterung bedingt dabei sowohl die transparente Bereitstellung von Dokumentmethoden als auch die transparente Erweiterung der Zuordnungsfunktion.

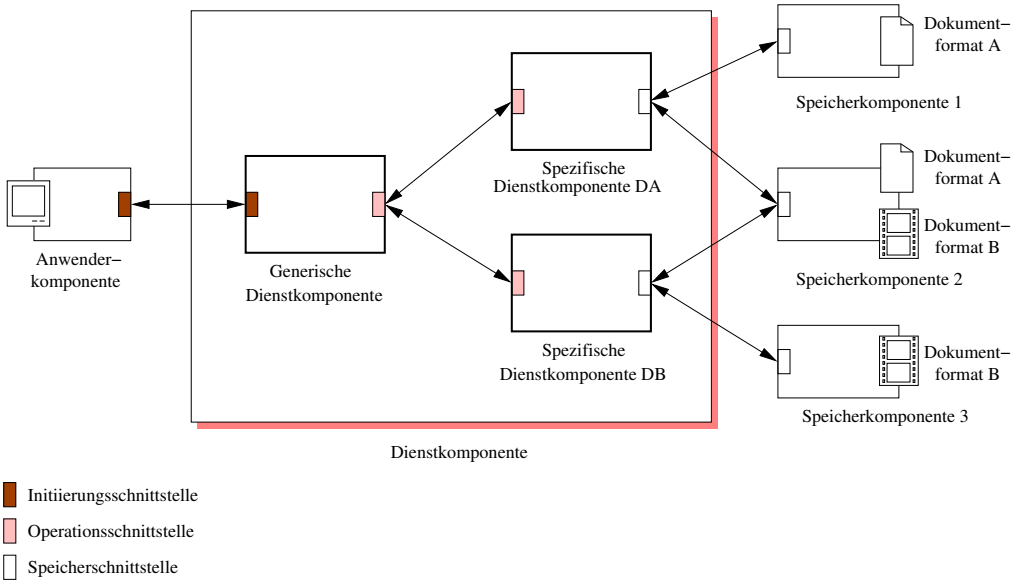
Die transparente Bereitstellung von Dokumentmethoden erfordert, daß die Elemente der Zuordnungsfunktion, Dokumente, Operationen, Laufzeitumgebungen und Dokumentmethoden systemweit eindeutig benannt werden. Nur dann ist die rechnergesteuerte Interpretation der Zuordnungsfunktion und Instantiierung einer geeigneten Dokumentmethode möglich. Die transparente Erweiterung der Zuordnungsfunktion erfordert einen Mechanismus, der die Zuordnungsfunktion in allen an der Digitalen Bibliothek beteiligten Komponenten verfügbar macht, so daß Änderungen sofort berücksichtigt werden können.

#### 2.3.1.2. Integration in existierende Dienste

Bei der Integration neuer Dokumentformate muß neben der Verfügbarkeit der orthogonalen Operationen auch sichergestellt sein, daß bereits existierende Dienste auf Dokumenten mit neuen Formaten ausgeführt werden können. Auch hier besteht die Forderung nach Orthogonalität, die Integration neuer Dokumentformate darf nicht zu einer Änderung der Dienstschnittstellen oder der Notwendigkeit manueller Intervention durch den Anwender führen. Die notwendigen Änderungen, die sich durch die Integration der neuen Dokumentformate ergeben, dürfen also nicht an der Dienstschnittstelle sichtbar werden.

Betrachtet man einen Dienst als monolithische Komponente, kann die Unterstützung eines neuen Dokumentformats nur durch den Austausch der Dienstkomponente, bzw. aller Dienstkomponenten, realisiert werden. Um diesen Aufwand zu vermeiden, ist es sinnvoll, Dienstkomponenten in dokumentspezifische und in generische Komponenten zu unterteilen. Dokumentspezifische und generische Dienstkomponenten kommunizieren über eine dienstinterne Schnittstelle miteinander, wie in Abbildung 2.6 dargestellt.

**Abbildung 2.6.** Strukturierung der Dienste in generische und spezifische Komponenten




Die generische Dienstkomponente kommuniziert mit den spezifischen Dienstkomponenten DA und DB zur Erbringung des Dienstes auf Dokumenten mit den Formaten A bzw. B. Die Erweiterung des Dienstes um die Fähigkeit zur Verarbeitung eines neuen Dokumentformats erfordert bei dieser Struktur lediglich die Bereitstellung einer entsprechenden dokumentspezifischen Dienstkomponente. Die generische Dienstkomponente entspricht zusammen mit der Menge der spezifischen Dienstkomponenten in ihrer Funktion der in Abbildung 2.2 dargestellten monolithischen Dienstkomponente.

Die in Abbildung 2.6 dargestellten dokumentspezifischen Dienstkomponenten sind aber nichts anderes als Dokumentmethoden, die von der generischen Schnittstelle der Speicherkomponenten auf die dienstinterne Schnittstelle abbilden. Die Instantiierung einer solchen Dokumentmethode läßt sich also als Ausführung einer orthogonalen Operation betrachten, deren Operationsschnittstelle der dienstinternen Schnittstelle entspricht. Dokumenttypspezifische Dienstkomponenten lassen sich also in Form von orthogonalen Operationen bereitstellen. Damit werden Dokumente in neuen Formaten sofort nach ihrer Integration in die Digitale Bibliothek von den existierenden Diensten unterstützt, denn die Dokumente stellen nach Voraussetzung orthogonale Operationen bereit.

Als Konsequenz der Nutzung orthogonaler Operationen zur Integration neuer Dokumente in existierende Dienste ergibt sich, daß die Menge der in der Digitalen Bibliothek definierten orthogonalen Schnittstellen nicht mehr allein von den Anforderungen an die Verarbeitung individueller Dokumente, z. B. der Präsentation, definiert ist. Sie ist darüber hinaus auch von den in der Digitalen Bibliothek verfügbaren Diensten abhängig.

### 2.3.2. Integration neuer Dienste

Die Integration neuer Dienste erfordert die Erstellung und Integration generischer Dienstkomponenten und, je nach Dienst, die Einführung einer neuen orthogonalen Schnittstelle, die als dienstinterne Schnittstelle fungiert. Die Erweiterung der Digitalen Bibliothek um neue Dienste kann damit auf die Erweiterung der Menge der orthogonalen Operationen und die transparente Einführung neuer generischer Dienstkomponenten zurückgeführt werden.

#### 2.3.2.1. Integration generischer Dienstkomponenten

Um die Skalierbarkeit neu integrierter Dienste gewährleisten zu können, muß eine dezentrale Realisierung der Dienstkomponente möglich sein. Die Integration der Dienste darf dabei, ähnlich wie die Integration neuer Dokumentformate, weder manuelle Intervention durch den Anwender erfordern, was die Orthogonalität beeinträchtigen würde, noch durch die Betreiber der Speicherkomponenten, was die Skalierbarkeit beeinträchtigen würde.

#### 2.3.2.2. Integration neuer orthogonaler Schnittstellen

Die dynamische Integration von neuen Diensten in die Digitale Bibliothek kann die Erweiterung um neue orthogonale Operationen zur Bereitstellung einer neuen dienstinternen Schnittstelle erfordern. Die Erweiterung der Menge der orthogonalen Operationen erfordert, neben der bereits für die Erweiterung um neue Dokumentformate notwendigen Möglichkeit zur transparenten Integration neuer Dokumentmethoden und zur transparenten Erweiterung der Zuordnungsfunktion, die Möglichkeit zur Definition neuer orthogonaler Operationen.

Die nachträgliche Erweiterung um neue orthogonale Operationen setzt die Möglichkeit

zur Erweiterung der Zuordnungsfunktion bereits existierender Dokumente sowie die Möglichkeit zur nachträglichen transparenten Einführung entsprechender Dokumentmethoden voraus. Der Grund dafür ist, daß eine Umsetzung existierender orthogonaler Schnittstellen auf eine neue orthogonale Schnittstellen in einer typerweiterbaren Digitalen Bibliothek nicht automatisiert durchgeführt werden kann (siehe Abschnitt 2.1.1). Die Bereitstellung externer Informationen ist daher immer notwendig, um eine neue orthogonale Schnittstelle zu etablieren.

## 2.4. Existierende Systeme zur Verarbeitung digitaler Dokumente

In diesem Abschnitt werden verteilte Systeme zur Verarbeitung digitaler Dokumente betrachtet, um festzustellen, inwieweit sie die in Abschnitt 1.3 genannten Anforderungen erfüllen. Dabei werden explizit Systeme zur Verarbeitung digitaler Dokumente und nicht ausschließlich Digitale Bibliotheken betrachtet. Diese beiden Kategorien sind nicht eindeutig voneinander abgegrenzt, i. a. wird für eine Digitale Bibliothek die aktive Verwaltung, Erschließung und Archivierung der Dokumente vorausgesetzt. Als Beispiel für ein System zur Verarbeitung digitaler Dokumente, das keine vollwertige Digitale Bibliothek darstellt wird u. a. das World Wide Web (WWW) angeführt (vgl. [70]). Daß das WWW in die im folgenden vorgenommenen Betrachtungen eingeschlossen wird, hat mehrere Gründe: zum einen bildet es häufig die Grundlage für die Realisierung Digitaler Bibliotheken, indem es durch entsprechende Dienste erweitert wird, zum anderen ermöglicht die Betrachtung des WWW die Analyse allgemeiner Konzepte, die in den häufig stark spezialisierten Digitalen Bibliotheken, wie z. B. der Musikbibliothek Meldex [55] oder der Bildbibliothek QBIC [31] in den Hintergrund treten.

Als Voraussetzung für die Erweiterbarkeit um neue Dokumenttypen wurden in den vorangegangenen Abschnitten die Existenz einer erweiterbaren Zuordnungsfunktion und die Möglichkeit zur Bereitstellung neuer Dokumentmethoden identifiziert. Wie die Erweiterung der Zuordnungsfunktion und die Bereitstellung neuer Dokumentmethoden vorgenommen wird hat großen Einfluß auf die Skalierbarkeit, Orthogonalität und Effizienz der Systeme, wie im vorliegenden Abschnitt gezeigt wird.

### 2.4.1. Administrative Instanzen

In Abschnitt 1.3 wurde bereits deutlich gemacht, daß eine Digitale Bibliothek nur als dezentrale verteilte Anwendung realisiert werden kann, um die geforderte Skalierbarkeit gewährleisten zu können (vgl. [6]). Unter einer verteilten Anwendung wird hier ein Zusammenschluß von Rechnerknoten verstanden, die getrennte Adreßräume besitzen und über ein Kommunikationsnetzwerk verbunden sind. Die Rechnerknoten stellen eine oder mehrere Ablaufumgebungen für Softwarekomponenten bereit, die zur Erfüllung einer gemeinsamen Aufgabe miteinander kommunizieren.

Digitale Bibliotheken bestehen daher aus einer Menge kommunizierender Rechnerknoten. In jeder Digitalen Bibliothek lassen sich zwei administrative Instanzen identi-

fizieren, deren Rechnerknoten an der Ausführung einer Operation oder eines Dienstes beteiligt sind: Anwender und Betreiber der Digitalen Bibliothek.

Anwender der Digitalen Bibliothek betreiben Rechnerknoten, auf denen Anwenderkomponenten plaziert sind. Diese Rechnerknoten werden im folgenden als *anwenderseitige Rechnerknoten* bezeichnet. Komponenten, die auf anwenderseitigen Rechnerknoten instantiiert werden besitzen lokalen Zugriff auf die Anwenderkomponenten, d. h. sie befinden sich entweder auf dem gleichen Rechnerknoten oder kommunizieren über ein lokales Netzwerk mit der Anwenderkomponente.

Betreiber der Digitalen Bibliothek betreiben Rechnerknoten, auf denen Speicherkomponenten plaziert sind. Diese Knoten werden im folgenden als *speicherseitige Rechnerknoten* bezeichnet. Komponenten, die auf speicherseitigen Rechnerknoten instantiiert werden besitzen lokalen Zugriff auf die Speicherkomponenten.

Je nach Dokumenttyp und Operation können mehrere anwenderseitige und/oder speicherseitige Rechnerknoten in die Ausführung einer Operation involviert sein. Es ist die Aufgabe der Administration, die zur Ausführung einer Dokumentmethode notwendigen Voraussetzungen, z. B. Verfügbarkeit der benötigten Methoden, zu schaffen. Beinhaltet dies eine manuelle Intervention, muß diese vom Anwender, vom Betreiber oder von beiden geleistet werden und ist damit nicht mehr transparent.

### 2.4.2. Integration neuer Dokumentmethoden

Existierende Systeme zur Verarbeitung digitaler Dokumente unterstützen meist keine transparente Integration neuer Dokumentmethoden. Der Grund dafür liegt darin, daß diese Systeme über keine systemweit eindeutige Benennung aller Elemente der Zuordnungsfunktion verfügen. Dadurch enthält der Arbeitsablauf zur Installation neuer Dokumentmethoden mindestens einen manuell auszuführenden Schritt und ist infolge dessen nicht transparent.

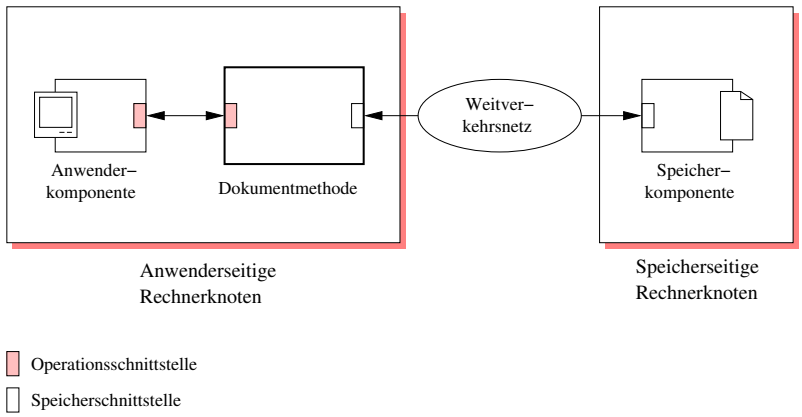
Die fehlende Transparenz hat großen Einfluß auf die Skalierbarkeit, Orthogonalität und Effizienz der Systeme, wie im folgenden gezeigt wird. In Abhängigkeit davon, ob die manuelle Bearbeitung auf der Seite der Anwenderkomponenten oder auf der Seite der Speicherkomponenten erfolgt ergeben sich unterschiedliche Einschränkungen.

#### 2.4.2.1. Plazierung von Dokumentmethoden

Integriert man neue Dokumentmethoden in eine Digitale Bibliothek, stellt sich die Frage nach der Plazierung der Methoden. Die Alternative besteht in der Installation auf anwenderseitigen Rechnerknoten, der Installation auf speicherseitigen Rechnerknoten oder der Installation auf beiden Klassen von Rechnerknoten. Jede dieser Möglichkeiten besitzt spezifische Vor- und Nachteile, die im folgenden eingehender untersucht und bewertet werden. Die Kriterien zur Bewertung der Plazierungsalternativen sind die zur Ausführung der Operation benötigte Kommunikationsbandbreite sowie die Gewährleistung der Skalierbarkeit und der Orthogonalität. Zusätzlich wird der zur Bereitstellung der Dokumentmethoden erforderliche Aufwand zur Bewertung herangezogen.

**Anwenderseitige Plazierung** Viele erweiterbare Systeme zur Verarbeitung digitaler Dokumente beschränken sich auf die Erweiterung anwenderseitiger Rechnerknoten, wie in Abbildung 2.7 dargestellt.

**Abbildung 2.7.** Anwenderseitige Plazierung von Dokumentmethoden




Die Erweiterung erfolgt dabei dezentral durch die Installation neuer Dokumentmethoden durch die Anwender. Die Vorteile dieses Ansatzes sind:

- Dokumentmethoden besitzen lokalen Zugriff auf anwenderseitige Komponenten, z. B. zur Präsentation eines Dokuments. Präsentationsorientierte und interaktive Operationen können daher mit geringem Bandbreitenaufwand durchgeführt werden.
- Dokumentspeicher müssen lediglich die generische Speicherschnittstelle anbieten. Dadurch kann jedes Dokumentformat in jedem Dokumentspeicher abgelegt werden. Die Speicherkapazität der Digitalen Bibliothek ist damit nur von der Gesamtzahl der Dokumentspeicher abhängig.

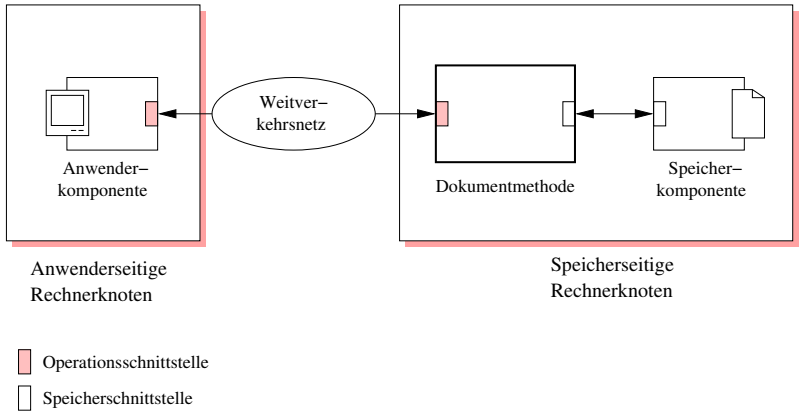
Diesen Vorteilen stehen folgende Nachteile der anwenderseitigen Plazierung gegenüber:

- Die manuelle Installation durch den Anwender verletzt die Orthogonalität, da der Anwender dokumentspezifische Handlungen vornehmen muß.
- Dokumentmethoden, die Operationen auf den Dokumentdaten, z. B. Volltextsuche, ausführen, greifen entfernt auf die speicherseitigen Komponenten zu. Dabei entsteht ein hoher Bandbreitenbedarf, vor allem, wenn Operationen auf mehreren Dokumenten ausgeführt werden, wie es für die Realisierung von Diensten notwendig ist.
- Durch die große Heterogenität der anwenderseitigen Rechnerknoten erfordert die Bereitstellung neuer Dokumentmethoden für alle anwenderseitigen Rechner einen hohen Implementierungsaufwand. Die fortgesetzte Migration von anwenderseitigen Plattformen erfordert die fortgesetzte Portierung existierender Methoden.

Dieses Modell wird in den unterschiedlichsten Digitalen Bibliotheken verwendet, z. B. im Informedia Projekt [102], und ist das grundlegende Paradigma der Kahn/Wilensky-Architektur für Digitale Bibliotheken [44]. Darüber hinaus ist die anwenderseitige Erweiterung die vorherrschende Art der Integration neuer Dokumentformate in das WWW, bei dem die anwenderseitige Erweiterung häufig durch sogenannte Hilfsanwendungen (Helper Applications) erfolgt.

**Speicherseitige Plazierung** Es gibt einige Systeme, die die Möglichkeit zur Erweiterung der speicherseitigen Rechnerknoten um neue Dokumentmethoden, wie sie in Abbildung 2.8 dargestellt ist, vorsehen.

**Abbildung 2.8.** Dokumentseitige Plazierung von Dokumentmethoden




Die speicherseitigen Rechnerknoten stellen in dieser Plazierungsvariante die orthogonalen Operationsschnittstellen bereit. Die Installation der Dokumentmethoden fällt bei dieser Plazierung in den Aufgabenbereich der Betreiber der Speicherkomponenten. Die Vorteile dieses Ansatzes sind:

- Die Orthogonalität wird nicht verletzt, da auf den anwenderseitigen Rechnerknoten keine weiteren Installationen mehr vorgenommen werden müssen.
- Dokumentmethoden besitzen lokalen Zugriff auf Dokumentdaten. Damit wird die bandbreitenschonende Realisierung datenintensiver Operationen und Dienste ermöglicht.
- Die Heterogenität der anwenderseitigen Plattformen hat keinen Einfluß auf die Zahl der notwendigen Implementierungen von Dokumentmethoden.

Die Nachteile der speicherseitigen Plazierung von Dokumentmethoden sind:

- Die Notwendigkeit der manuellen Bereitstellung der Dokumentmethode durch die Betreiber führt zu einer begrenzten Menge an speicherseitig vorhandenen Dokumentmethoden. Als Folge davon können Dokumentspeicher nur noch eine

begrenzte Zahl von Dokumentformaten unterstützen, sie werden formatspezifisch.

Die Existenz formatspezifischer Speicherkomponenten schränkt die Skalierbarkeit der Digitalen Bibliothek ein, denn Dokumente müssen auf Dokumentspeichern gespeichert werden, die über die notwendigen formatspezifischen Dokumentmethoden verfügen. Die Speicherkapazität der Digitalen Bibliothek ist damit nicht mehr ausschließlich durch die Zahl der Speicherkomponenten bestimmt sondern auch durch die Übereinstimmung der zu speichernden und der von den Speichern unterstützten Dokumentformate.

Um die Skalierbarkeit der Digitalen Bibliothek langfristig sicherzustellen ist die permanente Anpassung der installierten Dokumentmethoden an das Formatprofil der Dokumente notwendig.

- Die Kommunikation zwischen der Anwenderkomponente und den Dokumentmethoden erfolgt entfernt. Dies kann, abhängig von Präsentationsmedium des Dokuments und der Güte der Präsentation, zu einem hohen Bandbreitenbedarf bei präsentationsorientierten Operationen führen.

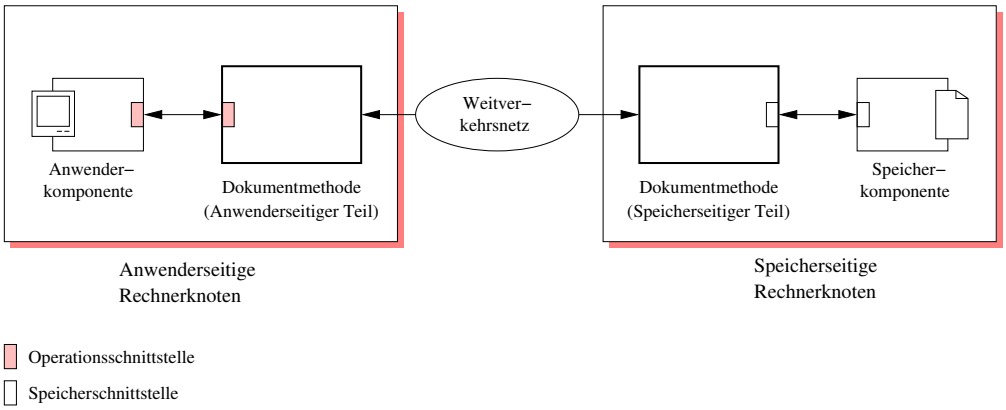
Ein verbreitetes Beispiel für diese Struktur sind speicherseitige Erweiterungen im WWW, die Zugriffe auf Datenbanken ermöglichen oder eine Gateway-Funktion bereitstellen. Die verwendeten Mechanismen beinhalten u. a. CGI-Skripte, Active Server Pages und Servlets. Der verteilte Indexierungsdienst Harvest (vgl. [9]) verwendet speicherseitige Komponenten zur lokalen Indexierung digitaler Dokumente. Weitere Beispiele für diese Art von dokumentverarbeitenden Systemen bieten Host-basierte Systeme, Z39.50-Server und alle Systeme, die eine entfernte Präsentation mittels des X11-Protokolls oder des Audiofile-Protokolls, einem Protokoll zur entfernten Präsentation zeitkontinuierlicher Daten, vornehmen. Die letzten beiden Beispiele illustrieren, daß multimediale Präsentationen zwar auch hier möglich sind, diese führen aber aufgrund der niedrigen Abstraktionsebene der präsentationsorientierten Schnittstellen zu einem hohen Bandbreitenbedarf. Aus diesem Grund ist ihre Anwendung i. a. auf lokale Netzwerke beschränkt.

**Verteilte Plazierung** Die ineffiziente Nutzung der Bandbreite in den oben beschriebenen Plazierungsmöglichkeiten entsteht durch die willkürliche Verteilung der Bearbeitung der Operation an der Speicherschnittstelle oder an der orthogonalen Schnittstelle. Eine effizientere Ausnutzung der Bandbreite läßt sich durch eine methodenspezifische bzw. formatspezifische Verteilung der Dokumentmethoden erreichen. Eine verteilte Plazierung der Dokumentmethode erlaubt eine Trennung der Arbeitsschritte an einer Schnittstelle, über die relativ wenige Daten fließen. Diese Plazierungsvariante ist in Abbildung 2.9 dargestellt.

Beispiele für diese Art der Erweiterbarkeit sind Streaming-fähige Systeme wie die RealAudio- und RealVideo-Server, der Video-Charger der IBM-Digital Library oder der Quicktime Video-Server. Darüber hinaus ist die verteilte Plazierung typisch für zentralisierte kooperative Systeme, wie z. B. den Internet Relay Chat (IRC).

Der Vorteil der verteilten Plazierung ist, daß sowohl datenorientierte als auch

**Abbildung 2.9.** Verteilte Plazierung von Dokumentmethoden




präsentationsorientierte Operationen im Hinblick auf die benötigte Bandbreite optimal ausgeführt werden können.

Diesem Vorteil stehen allerdings die kumulierten Nachteile der anwenderseitigen und der speicherseitigen Plazierung gegenüber. Die Orthogonalität ist durch die Installation von Dokumentmethoden durch den Anwender verletzt. Die Skalierbarkeit ist durch die Entstehung formatspezifischer Speicherkomponenten, infolge der Installation von Dokumentmethoden durch die Betreiber, beeinträchtigt. Die Bereitstellung von Dokumentmethoden auf anwenderseitigen Rechnerknoten erfordert die Berücksichtigung der heterogenen anwenderseitigen Plattformen.

Die Plazierung von Dokumentmethoden muß sich nicht auf eine Anwenderkomponente bzw. auf eine Speicherkomponente beschränken. So ist IRC ein Beispiel für die Plazierung von Dokumentmethoden auf einem speicherseitigen und vielen anwenderseitigen Rechnerknoten. Die genannten Nachteile für die Installation von Dokumentmethoden auf anwenderseitigen Rechnerknoten vervielfachen sich in diesem Fall allerdings.

#### 2.4.2.2. Erweiterung der Zuordnungsfunktion

Die Realisierung orthogonaler Operationen erfordert die Existenz einer Zuordnungsfunktion, die von den Komponenten des Systems zur Ermittlung geeigneter Dokumentmethoden verwendet wird. Diese Zuordnungsfunktion kann explizit oder implizit repräsentiert werden. Eine explizite Repräsentation der Zuordnungsfunktion ist außerhalb des Kontextes der Komponenten gespeichert und wird von den Komponenten des Systems zur Laufzeit gelesen. Eine implizite Repräsentation der Zuordnungsfunktion wird bei der Erstellung der Komponenten festgelegt und in den Programmtext der Komponenten integriert.

Die Einführung neuer Operationen, Dokumenttypen, Laufzeitumgebungen und Methoden erfordert eine Erweiterung der Zuordnungsfunktion. Diese Erweiterung der Zuordnungsfunktion kann in existierenden Systemen zur Verarbeitung digitaler Do-

kumente i. a. nicht transparent durchgeführt werden. Dafür gibt es zwei Gründe: die implizite Repräsentation von Teilen der Zuordnungsfunktion und die fehlende systemweit eindeutige Benennung der Elemente der Zuordnungsfunktion.

**Erweiterung impliziter Zuordnungsfunktionen** In vielen Systemen sind zumindest Teile der Zuordnungsfunktion implizit repräsentiert. Ein System, dessen Zuordnungsfunktion vollständig implizit repräsentiert wird ist die Networked Computer Science Techical Reports Library (NCSTRL) [20, 21]. NCSTRL verwendet das Dienst-Protokoll [46, 19], das eine feste Menge von sieben Dokumentformaten unterstützt, zur Kommunikation zwischen den Komponenten. Jede Modifikation der Zuordnungsfunktion in NCSTRL kann nur durch die Installation neuer Komponenten erfolgen, und ist daher nicht ohne die Beteiligung von Anwendern und Betreibern möglich.

Im Stanford Information Bus und im WWW ist die Zahl der orthogonalen Operationen implizit festgelegt. Der Stanford Information Bus [80, 69] definiert orthogonale Operationen zur Verwaltung von Dokumenten und Sammlungen (Digital Library Interoperability Protocol, DLIOP), zur Verwaltung von Metadaten (Stanford Digital Library Metadata Architecture, SMA), zur Suche (Stanford Protocol Proposal for Internet Retrieval and Search, STARTS), zur Abrechnung (Universal Payment Application Interface, UPAI), zur Verwaltung von Rechten (Stanford Framework for Interoperable Rights Management, FIRM) sowie zur Präsentation (Digital Library Integrated Task Environment, DLITE). Das WWW stellt eine orthogonale Operation bereit, die sich mit dem Namen „Präsentation“ bezeichnen läßt. Die Erweiterung dieser Infrastrukturen um neue Operationen erfordert die Bereitstellung neuer Komponenten und damit eine manuelle Intervention von Anwendern und Betreibern.

**Erweiterung expliziter Zuordnungsfunktionen** Die Erweiterung expliziter Zuordnungsfunktionen bzw. der Teile von Zuordnungsfunktionen, die explizit gespeichert werden erfordert die Benennung der zu integrierenden Elemente. Existiert keine systemweit eindeutige Benennung für diese Elemente müssen diejenigen Elemente, deren Benennungen lokal variieren manuell integriert werden.

Im WWW besteht der explizite Teil der Zuordnungsfunktion i. a. aus einer anwenderseitig gespeicherten Abbildung von MIME-Typen (Multipurpose Internet Mail Extension) [8] auf lokal signifikante Namen von Dokumentmethoden. Die Integration neuer Dokumentmethoden kann nur durch Intervention des Anwenders geschehen, da Dokumentmethoden und Laufzeitumgebungen im WWW nicht systemweit eindeutig benannt sind. Die Identifikation der geeigneten Methode muß daher manuell identifiziert werden.

In NCSTRL+ [63, 53] werden zur Repräsentation von Dokumenten Buckets [62] verwendet. Buckets weisen eine interne Struktur auf, die sich mit definierten Operationen bearbeiten läßt (vgl. [61]). Zur Integration neuer Dokumenttypen in einen Bucket nimmt der Betreiber der Speicherkomponente eine manuelle Erweiterung der Zuordnungsfunktion vor. Buckets stellen keine orthogonalen Operationen bereit, da die Resultate der Operationsausführung nicht definiert sind. In FEDORA [71, 70] werden Dokumente zusammen mit Disseminatoren, Methoden zu ihrer Verarbeitung, als so-

genannte *Digital Objects* gespeichert. Digital Objects unterstützen die manuelle Installation von Disseminatoren für neue Dokumenttypen durch die Betreiber der Digitalen Bibliothek.

Es gibt einige Systeme, die eine Erweiterung der Zuordnungsfunktion durch die Autoren der Dokumente ermöglichen. Im WWW wird dazu das Konzept der Applets verwendet. Es beruht auf der verteilten Speicherung einer speziellen Zuordnungsfunktion in den zu präsentierenden HTML-Seiten. Diese Zuordnungsfunktion beschränkt sich auf eine Laufzeitumgebung, bei Applets ist dies Java, auf ein Dokument, das Dokument in dem das Applet referenziert wird und auf eine orthogonale Operation, die Präsentation, die gleichzeitig die einzige orthogonale Operation im WWW ist. Aufgrund dieser Beschränkung kann die Zuordnungsfunktion in Form eines Verweises auf den Uniform Resource Locator (URL) [5] des Applets dargestellt werden, der in einem Applet-Element des entsprechenden HTML-Dokuments gespeichert ist. Die Speicherung im HTML-Dokument erlaubt den Transport der dokumentspezifischen Zuordnungsfunktion zum Anwender. Aufgrund der eindeutigen Benennung des Applets und der impliziten Kenntnisse der anderen Parameter der Zuordnungsfunktion kann das Applet transparent bereitgestellt werden. Mit der Verwendung des Object-Elements, das seit HTML 4.0 [96] unterstützt wird, kann eine erweiterte Zuordnungsfunktion im Dokument gespeichert werden, die zusätzlich die Spezifikation von Laufzeitumgebungen erlaubt. Die orthogonale Operation ist auch bei Verwendung des Object-Elements auf die Präsentation festgelegt.

Ein weiteres Konzept, das eine Erweiterung der Zuordnungsfunktion durch den Autor unterstützt sind Multivalente Dokumente (Multivalent Documents, MVDs) [74, 73]. MVDs bündeln verschiedene semantische Sichten eines Dokuments zusammen mit den Methoden zu ihrer Darstellung (vgl. [72]). Ein MVD ermöglicht dem Autor die Einführung neuer Sichten durch Modifikation des Dokuments. MVDs beschränken sich ebenfalls auf Java als Laufzeitumgebung und auf eine orthogonale Operation, bei der es sich ebenfalls um die Präsentation handelt.

### 2.4.3. Integration neuer Dienste

In Abschnitt 2.3.2 wurde ein Konzept entworfen, das Dienste in Form generischer Komponenten und orthogonaler Operationen modelliert. Damit wird sichergestellt, daß neue integrierte Dokumenttypen durch existierende Dienste bearbeitet werden können. Die Anwendung dieses Konzepts zur transparenten Integration neuer Dienste setzt allerdings voraus, daß die Digitale Bibliothek die Verwendung mehrerer orthogonaler Operationen sowie die transparente Integration neuer orthogonaler Operationen unterstützt.

Eine weitere Anforderung an die Dienste einer Digitalen Bibliothek ist ihre dezentrale Realisierung. Daraus ergibt sich für die Integration neuer Dienste die Forderung nach der Verteilung diensterbringender Komponenten auf die Rechnerknoten der Digitalen Bibliothek. Ist diese Verteilung nicht möglich, werden zwangsläufig zentralisierte Dienste, wie z. B. die Indexierungsmaschinen Altavista und Google, entstehen, deren Realisierung mit einem großen Ressourceneinsatz verbunden ist.

Im folgenden wird diskutiert, inwieweit existierende Systeme die transparente Inte-

gration neuer orthogonaler Operationen und die Verteilung dienstspezifischer Komponenten auf speicherseitige Rechnerknoten unterstützen und damit die beiden soeben genannten Anforderungen erfüllen.

#### 2.4.3.1. Unterstützung orthogonaler Operationen

Die Unterstützung orthogonaler Operationen durch existierende Systeme und die Möglichkeiten zur Integration neuer orthogonaler Operationen sind durch die Möglichkeiten zur Integration neuer Dokumentmethoden und zur Erweiterung der Zuordnungsfunktion bestimmt, die in den Abschnitt 2.4.2 und 2.4.2.2 untersucht wurden. Das dort gewonnene Resultat besagt, daß die transparente Integration neuer Dienste in existierende Systeme nicht möglich ist. Ein Grund dafür ist die in den meisten Systemen notwendige manuelle Integration neuer Dokumentmethoden. Ein weiterer Grund ist die fehlende Unterstützung zur transparenten Erweiterung der Menge der orthogonalen Operationen in existierenden Systemen.

#### 2.4.3.2. Verteilung diensterbringender Komponenten

Da existierende Systeme die transparente Bereitstellung von Komponenten auf speicherseitigen Rechnerknoten nicht unterstützen, erlauben sie keine transparente Verteilung diensterbringender Komponenten. Die Etablierung diensterbringender Komponenten auf speicherseitigen Rechnerknoten muß daher von deren Betreibern manuell vorgenommen werden. Ein Beispiel für die manuelle Erweiterung von Dokumentspeichern zum Zweck der verteilten Realisierung eines Dienstes bildet Harvest [9]. Harvest nimmt die verteilte Berechnung eines Indexes durch die Integration von Informationen, die durch lokale Zugriffe auf die Dokumente in den speicherseitigen Rechnerknoten ermittelt werden, vor.

Eine manuelle Integration diensterbringender Komponenten führt, wie bei der Integration von Dokumentmethoden, dazu, daß Speicherkomponenten nur eine begrenzte Zahl an Diensten unterstützten können, also dienstspezifisch werden. Dokumente, die der Bearbeitung durch einen Dienst zugänglich sein sollen müssen dann auf einer entsprechend ausgerüsteten Speicherkomponente gespeichert sein.

## 2.5. Fazit

Erweiterbare orthogonale Digitale Bibliotheken sollen langfristig die Speicherung und Nutzung digitaler Dokumente gewährleisten. Dazu müssen sie in der Lage sein, sich neu entwickelnde Dokumentformate und Dienste zu unterstützen. Dies setzt die Möglichkeit zur Integration neuer Dokumentmethoden in die Digitale Bibliothek, zur Erweiterung der Zuordnungsfunktion, zur Unterstützung mehrerer orthogonaler Operationen und zur Erweiterung der Digitalen Bibliothek um neue orthogonale Operationen voraus. Diese Möglichkeiten werden von existierenden Digitalen Bibliotheken häufig nicht in dem geforderten Umfang bereitgestellt.

### 2.5.1. Integration neuer Dokumentformate

Die Integration neuer Dokumentformate kann in existierenden Digitalen Bibliotheken i. a. nicht rechnergesteuert ausgeführt werden, sondern erfordert eine manuelle Intervention. Die Gründe für die Notwendigkeit der manuellen Intervention sind in der fehlenden systemweit eindeutigen Benennung der Elemente der Zuordnungsfunktion sowie in ihrer nicht automatisierten Verteilung zu suchen. Damit erfordert der Prozeß der Integration neuer Dokumentformate an mindestens einer Stelle eine intellektuelle Bearbeitung, die in der Interpretation der Zuordnungsfunktion, der Verteilung der Zuordnungsfunktion oder der Installation von Dokumentmethoden bestehen kann. Eine weitere Ursache für die Notwendigkeit manueller Bearbeitung sind Sicherheitsbedenken, die selbst bei einer vollständig rechnergesteuert interpretierbaren Zuordnungsfunktion eine automatische Bereitstellung von Dokumentmethoden verbieten.

In einer orthogonalen Digitalen Bibliothek, die sich nur manuell um neue Dokumentformate erweitern läßt, können aber Orthogonalität, Skalierbarkeit und effiziente Bandbreitennutzung nicht gleichzeitig gewährleistet werden. Der Grund dafür ist, daß entweder die Anwender oder die Betreiber der Digitalen Bibliothek die manuell auszuführenden Schritte durchführen müssen. Im einzelnen ergeben sich durch die manuelle Bearbeitung folgende Einschränkungen bei Skalierbarkeit, Orthogonalität und Bandbreitennutzung:

**Skalierbarkeit** Um die Skalierbarkeit sicherzustellen, muß eine Typisierung der Speicherkomponenten (siehe Abschnitt 2.4.2.1) verhindert werden, indem die manuell durchzuführenden Schritte zur Installation von Dokumentmethoden vom Anwender durchgeführt werden. Dadurch wird aber die Forderung nach Orthogonalität verletzt, da Anwender dokumentspezifische Aktionen durchführen müssen. Gleichzeitig können datenintensive Operationen nur mit hohem Bandbreiteneinsatz durchgeführt werden.

**Orthogonalität** Um die Orthogonalität zu gewährleisten, müssen die manuell durchzuführenden Schritte zur Installation von Dokumentmethoden von den Betreibern durchgeführt werden. Dadurch kommt es aber zu einer Typisierung der Dokumentspeicher, wodurch die Skalierbarkeit der Digitalen Bibliothek nicht mehr gewährleistet ist. Gleichzeitig können präsentationsorientierte Operationen bei hoher Qualität nur mit hohem Bandbreiteneinsatz durchgeführt werden.

**Bandbreitennutzung** Eine Optimierung des Bandbreitenbedarfs erfordert i. a. die Plazierung von Dokumentmethoden auf anwenderseitigen und speicherseitigen Rechnerknoten. Die manuelle Bereitstellung der Dokumentmethoden führt in diesem Fall zum Bruch der Orthogonalität und der Beeinträchtigung der Skalierbarkeit der Digitalen Bibliothek.

### 2.5.2. Zahl der orthogonalen Operationen

Viele Systeme beschränken sich auf die Unterstützung einer orthogonalen Operation, der Präsentation, ohne die Möglichkeit zur Erweiterung um neue orthogonale Operationen zu bieten. Ein Grund dafür ist in der herausragenden Bedeutung der

Präsentation zu suchen. Nahezu jedes Dokument läßt sich unter Verwendung einer generischen Präsentationsschnittstelle mittels der Präsentationsmedien Bildschirm und Lautsprecher darstellen. Eine weitere rechnergesteuerte Verarbeitung der präsentierten Dokumente ist dabei nicht vorgesehen, vielmehr findet die Interpretation durch den Anwender statt, der über ausreichendes Kontextwissen verfügt, um die unterschiedlichen Präsentationsarten korrekt interpretieren zu können. Eine rechnergesteuerte Verarbeitung der Ausgabe der Präsentationsschnittstelle würde umfangreiche externe Kenntnisse erfordern. Beispiele für den notwendigen Aufwand bilden Systeme zur optischen Zeichenerkennung (Optical Character Recognition, OCR) oder zur intelligenten Dokumenterkennung (Intelligent Document Recognition, IDR).

Der zweite Grund für die Beschränkung auf die Präsentation besteht in der allgemeinen Übereinkunft über die zur Präsentation benötigten Mechanismen und in ihrer allgemeinen Verfügbarkeit. Dadurch wird eine explizite Definition der Präsentationsschnittstelle überflüssig. In Systemen, die sich auf die Verwendung der Präsentationsschnittstelle beschränken, kann daher auf die Vereinbarung weiterer Schnittstellen verzichtet werden, was die Konzeption und Realisierung der Systeme stark vereinfacht.

### 2.5.3. Integration neuer Dienste

Die Beschränkung auf eine statische Menge von Operationen, die zudem meist nur ein Element besitzt, führt dazu, daß die in Abschnitt 2.3.2 vorgestellte Architektur, die die transparente Unterstützung neuer Dokumentformate in existierenden Diensten erlaubt, für neu integrierte Dienste nicht verwendet werden kann. Die Integration neuer Dokumentformate in existierende Dienste läßt sich in diesem Fall nur durch manuelle Modifikation der entsprechenden Komponenten verwirklichen. Dadurch kommt es, im Hinblick auf neu integrierte Dienste, ebenfalls zu einer Typisierung der Speicherkomponenten, analog zur Situation bei der manuellen Integration von Dokumentmethoden durch die Betreiber von Speicherkomponenten.

Die manuelle Integration hat noch einen zweiten Nachteil. Befinden sich nicht alle Speicherkomponenten in der administrativen Domäne des Dienstgestalters, kann dieser auf den Komponenten keine Methoden etablieren. Dies kann zu entfernten Zugriffen auf Dokumentdaten und damit zu einer ineffiziente Bandbreitennutzung führen.

In den folgenden Kapiteln wird eine Infrastruktur für erweiterbare orthogonale Digitale Bibliotheken vorgestellt, die die hier erkannten Ursachen für den Trade-off zwischen Orthogonalität, Skalierbarkeit und effizienter Bandbreitennutzung behebt.

# 3. Indigo — Infrastruktur für Digitale Bibliotheken

In diesem Kapitel wird eine Infrastruktur für Digitale Bibliotheken entworfen, die die Realisierung Digitaler Bibliotheken erlaubt, mit denen sich alle in Kapitel 1 genannten Anforderungen erfüllen lassen. Diese Infrastruktur wird im folgenden mit dem Namen Indigo (Infrastruktur für Digitale Bibliotheken) bezeichnet.

## 3.1. Lösungsansatz

In Kapitel 2 wurden die Gründe für die mangelnde Erweiterbarkeit, Skalierbarkeit und Orthogonalität existierender Systeme herausgearbeitet. Diese bestanden zum einen im Fehlen einer systemweit eindeutigen Benennung der Zuordnungsfunktion. Dies führt zu der Notwendigkeit der manuellen Intervention von Anwendern und Betreibern bei der Bereitstellung von Dokumentmethoden, woraus die genannten Einschränkungen resultieren. Zum anderen bieten viele Systeme zur Verarbeitung digitaler Dokumente lediglich Unterstützung für eine orthogonale Operation, die Präsentation. Damit wird die Erweiterung existierender Dienste um die Fähigkeit zur Bearbeitung neuer Dokumentformate sowie die transparente Integration neuer Dienste in die Digitale Bibliothek erschwert.

Das Ziel von Indigo liegt in der für Betreiber und Anwender transparenten Bereitstellung von Dokumentmethoden zur Durchführung orthogonaler Operationen. Dabei soll eine dokumentspezifische Verteilung der Operationsausführung möglich sein, um eine effiziente Bearbeitung der Dokumente zu gewährleisten. Das Gesamtsystem soll ferner skalierbar und erweiterbar sein, was neben der Zahl der Dokumente und Anwender, die Speicherung und Bereitstellung von Dokumentmethoden sowie die Speicherung und Erweiterung der Zuordnungsfunktion einschließt. Indigo erreicht die genannten Ziele, indem sie:

- Eine eindeutige Benennung für die in Abschnitt 2.2.4 beschriebenen Elemente der Zuordnungsfunktion: Dokumente, Operationen, Laufzeitumgebungen und Methoden einführt.
- Die verteilte Speicherung der Zuordnungsfunktion in der Digitalen Bibliothek ermöglicht.
- Einen Mechanismus zum transparenten Zugriff der an der Digitalen Bibliothek beteiligten Komponenten auf die Zuordnungsfunktion bereitstellt.

- Eine transparente Bereitstellung von Dokumentmethoden durch die Verwendung von mobilen Programmen und damit die rechnergesteuerte Interpretation der Zuordnungsfunktion ermöglicht.

Infolge dieser Maßnahmen kann die manuelle Bearbeitung der Zuordnungsfunktion auf den Autor eines Dokuments beschränkt werden, der diese bei der Konzeption des Dokuments initial vornehmen muß. Diese Zuordnung kann dann, transparent für Anwender und Betreiber, über die Infrastruktur an alle beteiligten Komponenten verteilt werden. Zusammen mit der transparenten Bereitstellung von Dokumentmethoden lassen sich damit sowohl anwenderseitige als auch speicherseitige Dokumentmethoden ohne manuelle Intervention und damit ohne den Verlust von Orthogonalität oder Skalierbarkeit realisieren. Ferner ermöglicht die Einführung der eindeutigen Benennung, neben der rechnergesteuerten Nutzung der Zuordnungsfunktion, die Verwendung mehrerer orthogonaler Operationen sowie eine verteilte Speicherung von Dokumenten und Methoden. Schließlich erlaubt die transparente Bereitstellung von Dokumentmethoden auf allen beteiligten Rechnerknoten die Realisierung von Diensten gemäß der in Abschnitt 2.3.1.2 vorgestellten Architektur und damit die transparente Unterstützung neuer Dokumentformate durch existierende Dienste.

Im vorliegenden Kapitel wird eine Infrastruktur entworfen, die diese Konzepte realisiert. Die Anwendung dieser Infrastruktur zur Verarbeitung von Dokumenten wird in Kapitel 4, die Möglichkeit zur Realisierung von Diensten und zur Integration neuer Dienste wird in Kapitel 5 behandelt.

## 3.2. Grundlegende Entwurfsentscheidungen

Die in diesem Kapitel vorgestellte Infrastruktur orientiert sich an dem in Kapitel 2 vorgestellten Modell erweiterbarer orthogonaler Digitaler Bibliotheken. Die dort identifizierten Elemente Operationen, Dokumentmethoden und Zuordnungsfunktion finden ihre Entsprechung in der Infrastruktur. Bei der Umsetzung des Modells existiert eine große Zahl von Freiheitsgraden, die sich auf die Speicherung und Erweiterung der Zuordnungsfunktion, auf die Speicherung und Erweiterung der Dokumentmethoden sowie auf die transparente Bereitstellung von Methoden beziehen. Im folgenden werden die bestehenden Alternativen erläutert und die beim Entwurf getroffene Auswahl begründet.

### 3.2.1. Konzeption und Verteilung der Zuordnungsfunktion

Die Abbildung von einem Dokument, einer Operation und einer Laufzeitumgebung auf die Dokumentmethode, die die genannte Operation auf dem angegebenen Dokument in der genannten Laufzeitumgebung durchführt, wird von der Zuordnungsfunktion vorgenommen. Aufgrund der Forderung nach Erweiterbarkeit der Digitalen Bibliothek um neue Dokumentformate und neue orthogonale Operationen ist die Größe der Zuordnungsfunktion unbeschränkt, was eine dezentrale verteilte Speicherung erfordert. Es soll möglich sein, die Erweiterung der Zuordnungsfunktion allein durch den Autor eines Dokuments vornehmen zu lassen. Schließlich müssen alle an der Digita-

len Bibliothek beteiligten Komponenten lesenden Zugriff auf die Zuordnungsfunktion besitzen.

#### 3.2.1.1. Strukturierung der Zuordnungsfunktion

Es gibt zwei prinzipielle Möglichkeiten, eine Zuordnungsfunktion zu realisieren: unter Verwendung eines Typsystems, das Dokumente gleichen Typs zusammenfaßt, oder ohne ein solches Typsystem. Unter Dokumenten des gleichen Typs sollen hier die Dokumente verstanden werden, auf denen alle orthogonalen Operationen durch jeweils identische Dokumentmethoden durchgeführt werden können. So besitzen, z. B. GIF-Bilder im WWW denselben Typ, da die einzige orthogonale Operation, die Präsentation, auf allen GIF-Bildern mittels ein und desselben Programms durchgeführt werden kann.

Dokumenttypen und Dokumentformate müssen bei dieser Definition nicht übereinstimmen. Es kann der Fall eintreten, daß eine Operation auf zwei Dokumenten mit gleichem Dokumentformat aber unterschiedlichem Inhalt nicht identisch ausgeführt werden kann. Der Grund dafür ist, daß der funktionale Zusammenhang zwischen der Eingabe und der Ausgabe nicht angegeben werden kann, z. B. bei der Umsetzung eingescannter Textseiten in Textdokumente mittels OCR. Obwohl die Dokumentformate aller eingescannten Textseiten identisch sein können, die Seiten können z. B. im TIFF-Format vorliegen, können unterschiedliche Eingaben für eine korrekte Erkennung notwendig werden. Im Fall einer rechnergesteuerten Ausführung bedeutet dies die Implementierung unterschiedlicher Aufrufe der OCR-Software. Die Dokumente lassen sich in diesem Fall trotz identischer Eingabe- und Ausgabeformate nicht mit den gleichen Dokumentmethoden bearbeiten, besitzen nach obiger Definition also nicht den gleichen Typ. Weitere Beispiele für inhaltsspezifische Dokumentmethoden sind Methoden zur inhaltsorientierten Suche in nicht-textuellen Dokumenten, z. B. zur Volltextsuche in Ton- und Bilddokumenten.

**Zuordnungsfunktion ohne Typen** Eine Zuordnungsfunktion ohne Typen bildet ein 3-Tupel bestehend aus Dokument, Operation und Laufzeitumgebung auf eine Dokumentmethode ab. Der Aufwand für die Speicherung der Zuordnungsfunktion ohne Typen liegt in der Größenordnung $O(|\text{Dokumente}| * |\text{Operationen}| * |\text{Laufzeitumgebungen}|)$. Nimmt man an, daß die Zahl der Dokumente groß gegenüber der Zahl der Operationen und der Zahl der Laufzeitumgebungen ist, trägt vor allem die Zahl der Dokumente zum Speicheraufwand bei.

**Zuordnungsfunktion mit Typen** Eine Zuordnungsfunktion mit Typen verwendet eine zusätzliche Abbildung, die *Typabbildung*, die jedes Dokument auf einen Dokumenttyp abbildet. Die Zuordnungsfunktion mit Typen bildet ein 3-Tupel bestehend aus Dokumenttyp, Operation und Laufzeitumgebung auf eine Dokumentmethode ab. Der Aufwand für die Speicherung der Zuordnungsfunktion mit Typen liegt in der Größenordnung $O(|\text{Dokumenttypen}| * |\text{Operationen}| * |\text{Laufzeitumgebungen}|)$. Zusätzlich muß Speicher in der Größenordnung $O(|\text{Dokumente}|)$ für die Speicherung der Typabbildung bereitgestellt werden. Durch das Zusammenfassen der Dokumente

gleichen Typs verringert sich der Aufwand der Speicherung der Typfunktion um den Faktor

$$\frac{|\text{Dokumenttypen}|}{|\text{Dokumente}|}.$$

Die Verwendung einer Typabbildung ist nur dann sinnvoll, wenn die Zahl der Dokumenttypen klein gegenüber der Zahl der Dokumente ist. Das bedeutet, daß alle Operationen auf einer großen Zahl von Dokumenten durch jeweils identische Dokumentmethoden realisierbar sein müssen. Jedes Dokument, das eine oder mehrere spezifische Dokumentmethoden benötigt, erfordert die Erzeugung eines neuen Typs. Abhängig von der Häufigkeit spezifischer Methoden kann sich die Zahl der Typen der Zahl der Dokumente annähern und so eine Typabbildung überflüssig machen.

Aufgrund der geforderten Skalierbarkeit der digitalen Bibliothek, der Möglichkeit zur Erweiterung des Systems um neue Dokumentformate und der Möglichkeit zur Integration neuer orthogonaler Operationen kann die Inhaltstransparenz der Dokumentmethoden nicht gewährleistet werden und damit eine Typanzahl, die in der Größenordnung der Dokumentzahl liegt, nicht ausgeschlossen werden. Aus diesem Grund wird in dem vorliegenden Modell auf ein Typsystem verzichtet und eine Zuordnungsfunktion ohne Typen verwendet.

Ein weiterer Grund für den Verzicht auf ein Typsystem ist die Möglichkeit zur einfachen Erstellung und Integration neuer Dokumenttypen durch Dokumentautoren, wenn die im folgenden Abschnitt beschriebene „Verteilung nach Dokument“ zur verteilten Speicherung der Zuordnungsfunktion verwendet wird.

#### 3.2.1.2. Verteilte Speicherung der Zuordnungsfunktion

Die Zuordnungsfunktion muß in den Dimensionen Dokumente, Operationen und Laufzeitumgebungen erweiterbar sein, um die Integration neuer Dokumente, neuer Operationen und neuer Laufzeitumgebungen unterstützen zu können. Durch die erweiterbare Zahl an Dokumenten und Operationen ist die Zuordnungsfunktion in ihrer Größe unbeschränkt, was ihre dezentrale verteilte Speicherung notwendig macht. Die dezentrale verteilte Speicherung erfordert ein Konzept zur Partitionierung der Zuordnungsfunktion. Gleichzeitig muß sichergestellt werden, daß alle Komponenten der Infrastruktur auf die Zuordnungsfunktion zugreifen können.

Die Verteilung der Zuordnungsfunktion kann nach unterschiedlichen Gesichtspunkten erfolgen, in der vorliegenden Arbeit wird die Verteilung entlang der Dimensionen der Bildmenge der Zuordnungsfunktion untersucht. Im einzelnen wird die verteilte Speicherung der Zuordnungsfunktion entlang der Dimension Dokument, entlang der Dimension Operation sowie entlang der Dimension Laufzeitumgebung näher betrachtet.

**Verteilung nach Laufzeitumgebung** Die Speicherung der Zuordnungsfunktion kann entlang der Dimension Laufzeitumgebung verteilt vorgenommen werden. Dazu wird als Bestandteil jeder Laufzeitumgebung eine Abbildung von Dokumenten und

Operationen auf Dokumentmethoden gespeichert. Diese umgebungsspezifische Abbildungsfunktion ordnet jedem Dokument und jeder Operation die Methode zu, die die gewünschte Operation auf dem angegebenen Dokument in der vorhandenen Laufzeitumgebung ausführt. Ein Beispiel für diese Verteilung der Zuordnungsfunktion bildet das WWW mit den MIME-Typen, das eine umgebungsspezifische Abbildung von Dokumenttypen[1] und der Operation Präsentieren auf umgebungsspezifische Dokumentmethoden erlaubt.[2] Der Nachteil der nach Laufzeitumgebungen verteilten Speicherung liegt zum einen in der Größe der umgebungsspezifischen Abbildungsfunktion, die bei Verzicht auf ein Typsystem von der Zahl der Dokumente abhängig ist, und zum anderen in dem hohen Aktualisierungsaufwand bei der Einführung neuer Dokumente und/oder neuer Operationen, der die umgebungsspezifischen Abbildungen aller Laufzeitumgebungen betrifft.

**Verteilung nach Operation** Die Verteilung entlang der Dimension der orthogonalen Operationen resultiert in operationsspezifischen Abbildungsfunktionen, die ein Dokument und eine Laufzeitumgebung auf die Methode abbilden, die die Operation auf dem Dokument in der angegebenen Laufzeitumgebung ausführt. Diese Art der Verteilung besitzt, analog zur Verteilung nach Laufzeitumgebungen den Nachteil, daß die operationsspezifische Abbildungsfunktion aufgrund der großen Zahl an Dokumenten sehr groß werden kann. Darüber hinaus existiert kein kanonischer Ort zur Speicherung der operationsspezifischen Abbildungsfunktionen, da Operationen nicht durch Komponenten repräsentiert sind, sondern ein übergreifendes Konzept der Infrastruktur bilden und lediglich in Form einer Definition und durch ihre Namen in Erscheinung treten.

**Verteilung nach Dokument** Um die Zuordnungsfunktion entlang der Dimension Dokumente verteilt zu speichern, wird zu jedem Dokument eine dokumentspezifische Zuordnungsfunktion assoziiert, die eine Operation und eine Laufzeitumgebung auf eine Dokumentmethode abbildet. Ein Beispiel für diese Art der Verteilung findet sich ebenfalls im WWW, z. B. in der Form von Applets. Die dokumentspezifische Zuordnungsfunktion besteht in diesem Fall aus der Angabe des entsprechenden Java-Programms.

Diese Art der Verteilung besitzt mehrere Vorteile. Sie ist skalierbar in Hinsicht auf die Zahl der Dokumente, da für jedes Dokument nur eine kleine dokumentspezifische Abbildungsfunktion gespeichert werden muß. Der Grund dafür ist, daß die Zahl der orthogonalen Operationen i. a. klein ist (siehe Abschnitt 3.3.2.1) und die Zahl der Laufzeitumgebungen in Indigo auf wenige Laufzeitumgebungen beschränkt ist (siehe Abschnitt 3.2.2.3).

Ein weiterer Vorteil dieser Art der Verteilung ist die Möglichkeiten zur dezentralen Erweiterung der Zuordnungsfunktion durch die Dokumentautoren. Damit ist die intellektuelle Bearbeitung der Dokumente an eine Instanz übertragen, die bereits mit dem Dokument vertraut ist. Darüber hinaus besitzt der Autor vollständige Freiheit in der

[1] Die im WWW verwendete Zuordnungsfunktion verwendet Typen.

[2] Da im WWW nur eine orthogonale Operation, die Präsentation, unterstützt wird, ist der zweite Parameter der Abbildungsfunktion konstant.

Festlegung der Dokumentmethoden, da keine weiteren Aktivitäten von dritter Seite notwendig sind.

Aufgrund der genannten Vorteile wird die Zuordnungsfunktion in Indigo verteilt nach Dokumenten gespeichert.

### 3.2.2. Transparente Bereitstellung von Methoden

Die transparente Bereitstellung von Dokumentmethoden beinhaltet die Sicherstellung der Verfügbarkeit der Dokumentmethoden auf den entsprechenden Rechnerknoten. Dies umfaßt die Bereitstellung des Programmtextes und seine Instantiierung in einer entsprechenden Laufzeitumgebung. Aufgrund der geforderten Erweiterbarkeit kann die Bereitstellung der Dokumentmethoden nicht statisch erfolgen, denn ihre Zahl ist potentiell unbegrenzt. Daher können nicht alle Dokumentmethoden in jeder Komponente der Digitalen Bibliothek vorgehalten werden. In Indigo wird die transparente Bereitstellung durch die Verwendung von mobilen Programmen zur Implementierung von Dokumentmethoden realisiert.

#### 3.2.2.1. Mobile Dokumentmethoden

Die Verwendung mobiler Programme zur Implementierung von Dokumentmethoden ermöglicht den bedarfsgesteuerten Transport von Methoden zu den entsprechenden Rechnerknoten. Unter mobilen Programmen werden hier Programme verstanden, die unverändert auf heterogene Rechnerknoten transportiert werden können und auf jedem dieser Rechnerknoten mit identischer Semantik ausgeführt werden (vgl. [1]).

Durch die rechnergesteuert interpretierbare Zuordnungsfunktion kann der Transport der mobilen Programme transparent für den Betreiber des entsprechenden Rechnerknotens durchgeführt werden. Dadurch wird der Bruch der Orthogonalität auf der Seite des Anwenders und die eingeschränkte Skalierbarkeit durch die Typisierung von speicherseitigen Rechnerknoten vermieden.

Die Mobilität der Dokumentmethoden erlaubt den Transport von Strukturkenntnis an die Orte, an denen diese Kenntnis am vorteilhaftesten eingesetzt werden kann. Diese Vorteile können z. B. in der bandbreiteneffizienten Durchführung von Operationen oder in Bereitstellung zuvor nicht unterstützter Funktionalität bestehen. Der Einsatz mobiler Programme zum Transport von Strukturkenntnis ist in den letzten Jahren, u. a. in Folge der Einführung der Programmiersprache Java, zunehmend attraktiv geworden. (vgl. [68]). Mobile Strukturkenntnis wird z. B. in der Stanford Digital Library [67] zum Realisierung der mobilen Anwenderschnittstelle DLITE [15, 16] sowie in Multivalenten Dokumenten (MVD) [74, 72] zum Transport spezifischer Präsentationsmethoden zum Anwender eingesetzt. Uniform Resource Agents (URAs) [18] transportieren Strukturkenntnis auf Datenspeicher, um datenintensive Berechnungen lokal zu den Daten ausführen zu können. Für Multimedia-Systeme wird ein analoges Konzept mit dem Namen Mobile Information Manager (MIM) bezeichnet (vgl. [92]). Ein weiteres Beispiel für die Verwendung mobiler Strukturkenntnis ist die Architektur von Jini [103], in der Java-Programme zum Transport der zur Ansteuerung von Geräten notwendigen Information verwendet werden.

#### 3.2.2.2. Speicherung der Dokumentmethoden

Die unbegrenzte Zahl an Dokumentmethoden erfordert ein Konzept zur dezentralen, verteilten Speicherung der Dokumentmethoden. In Indigo wird dazu auf existierende verteilte Speicher zurückgegriffen, die eine global eindeutige Benennung der Methoden mittels der Verwendung eines Uniform Resource Identifiers (URI) [4] erlauben. Die Identifikation der URIs durch Präfixe erlaubt auch den Einsatz ortstransparenter, persistenter Uniform Resource Names (URN) [94], falls die Dienste für ihre Abbildung auf URLs etabliert werden.

Speicher für Dokumentmethoden können z. B. aus HTTP- oder ftp-Servern bestehen, die Benennung der Methoden kann in diesem Fall durch die entsprechenden URLs geschehen.

#### 3.2.2.3. Plattformunabhängigkeit

Plattformunabhängigkeit kann durch Bereitstellung von Dokumentmethoden für jede neue Plattform oder durch die Beschränkung auf einige wenige homogene Laufzeitumgebungen erreicht werden. Während der erste Ansatz die Migration aller Dokumentmethoden auf jede neue Plattform verlangt, erfordert die Integration einer homogenen Laufzeitumgebung lediglich die einmalige Bereitstellung der Laufzeitumgebungen auf einer neuen Plattform. In Indigo wird daher der Weg der Beschränkung auf wenige homogene Laufzeitumgebungen gegangen, indem interpretierende Umgebungen, z. B. Java-Interpreter, zur Ausführung von Dokumentmethoden verwendet werden. Diese weisen neben der Plattformunabhängigkeit einen weiteren Vorteil auf: sie ermöglichen die Durchsetzung von Sicherheitsanforderungen und damit die sichere Ausführung von transparent bereitgestellten mobilen Programmen. Der Nachteil dieses Ansatzes besteht in der geringeren Effizienz interpretierter Programme im Vergleich zu Programmen, die in nativem Code vorliegen. Dieser Nachteil wird hier in Kauf genommen, um den Aufwand der Portierung von Dokumentmethoden auf neue Plattformen bzw. den Verzicht auf die Unterstützung neuer Plattformen zu vermeiden.

#### 3.2.2.4. Ausführungsort und Methodenverteilung

Mit der Möglichkeit zur transparenten Bereitstellung von Dokumentmethoden auf allen an der Digitalen Bibliothek beteiligten Rechnerknoten stellt sich die Frage nach der Wahl des Ausführungsortes einer Methode. Prinzipiell kann eine Dokumentmethode auf einem Rechnerknoten ausgeführt werden, der lokalen Zugriff auf die Dokumentdaten bietet, sie kann auf einem Rechnerknoten ausgeführt werden, der lokalen Zugriff auf die anwenderseitigen Ressourcen besitzt und sie kann auf einem dritten an der Digitalen Bibliothek beteiligten Rechnerknoten ausgeführt werden.

Die in Indigo getroffene Wahl des Ausführungsortes soll eine effiziente Bandbreitennutzung ermöglichen. Dabei stellt sich das Problem, daß es nicht möglich ist, für alle Dokumente a priori zu entscheiden, auf welchem Rechnerknoten eine Dokumentmethode am bandbreiteneffizientesten ausgeführt werden würde, und ob ein Transport des Dokuments bzw. der Dokumentmethoden zu diesem Rechner durch die Einsparung gerechtfertigt werden würde. Der Transport des Dokuments müßte vor der Ausführung

der Methode stattfinden und kann daher nur von einem nicht dokumentspezifischen Teil der Infrastruktur vorgenommen werden. Dieser Teil kann aber keine Entscheidungen über den optimalen Ausführungsort für Methoden treffen, denn der optimale Ausführungsort einer Dokumentmethode hängt von Faktoren wie dem Dokumenttyp, dem Dokumentinhalt und der Struktur der Dokumentmethode selbst ab. Daher sollte die Entscheidung über den Ausführungsort einer Dokumentmethode individuell für jedes Dokument getroffen werden können.

Indigo unterstützt ein Modell, das die Kontrolle über die Durchführung einer Operation so früh wie möglich an eine Dokumentmethode übergibt, und dieser erlaubt, den Ausführungsort für die weitere Durchführung der Operation selbst zu bestimmen. Die Instantiierung der Dokumentmethode erfolgt dabei auf dem speicherseitigen Rechnerknoten, der lokalen Zugriff auf die Dokumentdaten besitzt. Dies erlaubt die Ausnutzung von Lokalität, wo dies möglich ist. Die Methoden entscheiden dann selber über eine mögliche Verteilung auf andere Rechnerknoten und können eine für das Dokument optimale Verteilung realisieren, da sie alle notwendigen Informationen über das Dokument besitzen.

## 3.3. Die Elemente der Infrastruktur

Indigo implementiert die im Modell orthogonaler Bibliotheken definierten Elemente: Dokumente, orthogonale Operationen, Methoden, Laufzeitumgebungen und Zuordnungsfunktion (siehe Kapitel 2). Dabei werden die in Abschnitt 3.2 getroffenen Entwurfsentscheidungen zugrundegelegt.

### 3.3.1. Überblick

Wesentlich für die Vermeidung der in Abschnitt 2.5 identifizierten Nachteile ist in Indigo die Erstellung der Zuordnungsfunktion durch den Dokumentautor sowie ihre transparente Verteilung und Interpretation durch die Infrastruktur. Um dieses Ziel zu erreichen, wird die Zuordnungsfunktion verteilt nach Dokumenten gespeichert. Zur Speicherung der Zuordnungsfunktion werden in Indigo Metadokumente eingeführt, welche die den Dokumenten zugeordneten, dokumentspezifischen Zuordnungsfunktionen und das entsprechende Dokument zusammenfassen. Eine dokumentspezifische Zuordnungsfunktion besteht aus der Abbildung von Operationen und Laufzeitumgebungen auf Dokumentmethoden (siehe Abschnitt 3.2.1.2).

Die Ausführung von Dokumentmethoden in Indigo geschieht in Ausführungs-Servern, die auf jedem Rechnerknoten der Digitalen Bibliothek existieren. Ausführungs-Server speichern und tauschen Metadokumente aus, stellen Laufzeitumgebungen für Dokumentmethoden bereit und ermöglichen über eine entsprechende Schnittstelle die Initiierung orthogonaler Operationen auf den von ihnen gespeicherten Dokumenten. Eine auf Indigo basierende Digitale Bibliothek besteht aus einem Verbund von Ausführungs-Servern.

Zur Durchführung einer orthogonalen Operation interpretiert der Ausführungs-Server das Metadokument des entsprechenden Dokuments, kopiert die notwendige Dokumentmethode in seinen lokalen Speicher und instantiiert die Methode in einer ent-

sprechenden Laufzeitumgebung. Laufzeitumgebungen bieten den Methoden Schnittstellen zur Nutzung von Ressourcen des Rechnerknotens sowie zur Initiierung orthogonaler Operationen an Dokumenten in Ausführungs-Servern an. In Kapitel 4 wird erläutert, wie Dokumentmethoden diese Mechanismen nutzen können, um die optimale Durchführung orthogonaler Operationen zu gewährleisten. In den folgenden Abschnitten werden die hier skizzierten Elemente ausführlicher beschrieben.

### 3.3.2. Operationen in Indigo

#### 3.3.2.1. Orthogonale Operationen

Die Identifikation orthogonaler Operationen ist schwierig, da eine orthogonale Operation sinnvoll auf jedes Dokument anwendbar sein muß. Die Festlegung der Menge der orthogonalen Operationen geschieht über ein soziales Protokoll, also eine Vereinbarung zwischen den an der Digitalen Bibliothek beteiligten Personen. In Indigo wurde eine kleine Menge orthogonaler Operationen definiert, die keinen Anspruch auf Vollständigkeit erhebt. Sie wurde anhand der Erfahrungen mit den in [57, 58] beschriebenen prototypischen Implementierungen von Indigo identifiziert.

Als orthogonale Operationen wurden nicht nur Operationen für Vorgänge wie Präsentation oder Suche identifiziert, sondern auch Operationen für Vorgänge, die vollständig auf der Ebene der Bit-Folge hätten durchgeführt werden können, wie z. B. Kopieren oder Verschieben eines Dokuments. Diese orthogonalen Operationen wurden eingeführt, da es für einige Dokumenttypen sinnvoll sein kann, das Kopieren einer Dokumentmethode zu übertragen. Diese kann dann z. B. Sicherheitsanforderungen prüfen, über die Version des Dokuments Buch führen oder in einem sich dynamisch verändernden Dokument die Übertragung auf die statischen Inhalte beschränken. Die folgende Liste enthält eine Beschreibung der identifizierten orthogonalen Operationen:

**Present** Diese Operation erhält als Argument die Adresse des Ausführungs-Servers, auf dem das Dokument präsentiert werden soll. Sie führt daraufhin alle Operationen durch, die zur Präsentation des Dokuments notwendig sind. Die Methode liefert eine boolesche Statusmeldung zurück, die anzeigt, ob die Präsentation initiiert werden konnte.

**Describe** Diese Operation besitzt kein Argument und liefert eine Inhaltsbeschreibung des Dokuments in einem einheitlichen Format zurück. In Indigo wird dazu eine Liste von Schlüsselwörtern verwendet. Diese Operation ermöglicht die inhaltsbasierte Suche im jeweiligen Dokument und bildet die Basis für eine Indexierung der Dokumente.

**Move** Diese Operation dient dazu, ein Dokument auf den Ausführungs-Server zu verschieben, der als Argument übergeben wird, und es vom ursprünglichen Ausführungs-Server zu entfernen. Die Operation liefert eine boolesche Statusmeldung zurück.

**Copy** Die Operation Copy führt die gleiche Funktion wie Move aus, mit dem Unterschied, daß das Dokument nicht vom ursprünglichen Ausführungs-Server entfernt wird.

Die orthogonalen Operationen werden in optionale und obligatorische Operationen unterteilt. Zu den obligatorischen Operationen gehören Present und Describe. Diese Operationen müssen von jedem Dokument unterstützt werden. Die übrigen orthogonalen Operationen sind optional, d. h. Dokumente müssen keine Dokumentmethoden für diese Operationen bereitstellen. Sind diese Operationen nicht definiert, wird eine in die Ausführungs-Server integrierte generische Implementierung genutzt, die die entsprechende Operation auf Ebene der Bit-Folge durchführt. Dies ist notwendig, da andernfalls die Bedingung der Orthogonalität verletzt wäre.

#### 3.3.2.2. Private Operationen

Die Mechanismen der Infrastruktur ermöglichen die transparente Ausführung orthogonaler Operationen auf einem Dokument. Wie in Abschnitt 3.2.2 erläutert wurde, können Dokumentmethoden eine Operation durch die Instantiierung weiterer Programme, die spezifisch für den Dokumenttyp sind, auf anderen Rechnerknoten verteilt ausführen. Diese Vorgehensweise setzt zweierlei voraus: die Möglichkeit zum Transport der Programme auf andere Rechnerknoten sowie die Möglichkeit zur Ausführung der transportierten Programme. Diese Voraussetzungen lassen sich durch die Verwendung von Dokumentmethoden zur Realisierung der dokumentspezifischen Programme erfüllen. Daher liegt es nahe, die in der Infrastruktur vorhandenen Mechanismen zum Transport und zum entfernten Aufruf von Methoden ebenfalls zur Ausführung dokumentspezifischer Programme auf entfernten Rechnerknoten zu verwenden. Dies wird durch die Einführung privater Operationen ermöglicht.

Private Operationen unterscheiden sich von orthogonalen Operationen dadurch, daß sie einen eigenen Namensraum besitzen, der sie von den orthogonalen Operationen unterscheidbar macht, und daß ihre Signatur nicht systemweit festgelegt ist, sondern beliebig gewählt werden kann. Die Trennung der Namensräume wird dadurch erreicht, daß alle privaten Operationsnamen das Präfix *private.* besitzen, das für die Namen orthogonaler Operationen verboten wird. Die Trennung der Namensräume von orthogonalen und privaten Operationen ermöglicht eine Erweiterung der Menge der orthogonalen Operationen, ohne daß in den einzelnen Dokumenten die Gefahr eines Namenskonflikts zwischen orthogonalen und privaten Operationen besteht.

### 3.3.3. Metadokumente

Metadokumente speichern den Inhalt eines Dokuments zusammen mit der dokumentspezifischen Zuordnungsfunktion, die Operationen und Laufzeitumgebungen auf diejenigen Dokumentmethoden abbildet, die die Operationen auf dem jeweiligen Dokument ausführen. Darüber hinaus speichern Metadokumente Attribute, die der Verwaltung der Dokumente in Indigo dienen. Metadokumente setzen sich also aus drei Teilen zusammen:

- Der Beschreibung der Attribute.
- Der Beschreibung der dokumentspezifischen Zuordnungsfunktion.
- Dem Dokumentinhalt.

Metadokumente bilden ein standardisiertes Austauschformat für Dokumente, das zur Kommunikation mit Ausführungs-Servern verwendet wird. Jedes Dokument in Indigo wird durch ein Metadokument repräsentiert.

Metadokumente werden von den Dokumentautoren erstellt, die damit die Teile der Zuordnungsfunktion definieren, die die entsprechenden Dokumente beinhalten. Nach der Erstellung eines Metadokuments versendet der Autor das Metadokument an einen Ausführungs-Server, womit es Bestandteil der Dokumentmenge der Digitalen Bibliothek wird und einen systemweit eindeutigen Namen erhält, der sich aus dem systemweit eindeutigen Namen des Ausführungs-Servers sowie aus einem innerhalb des Ausführungs-Servers eindeutigen Dokumentnamen zusammensetzt. Durch den Verzicht auf ein Typsystem sind keine Erweiterung einer Typabbildung zur Einführung eines neuen Dokumenttyps notwendig. Alle weiteren an der Verarbeitung des Dokuments beteiligten Instanzen benötigten lediglich die im Metadokument gespeicherten Informationen, um Operationen auf dem neuen Dokument bzw. Dokumenttyp ausführen zu können.

Die drei Bestandteile des Metadokuments werden in den folgenden Abschnitten näher beschrieben.

#### 3.3.3.1. Zuordnungsfunktion

In diesem Teil des Metadokuments wird die dokumentspezifische Zuordnungsfunktion gespeichert. Sie bildet Operationsnamen und Laufzeitumgebungen auf Dokumentmethoden, bzw. deren URIs ab. Abgesehen von den Namen der Operationen besteht in der Zuordnungsfunktion kein Unterschied zwischen orthogonalen und privaten Operationen.

Dokumentmethoden haben weder schreibenden noch lesenden Zugriff auf diesen Teil des Metadokuments. Die Zuordnungsfunktion wird von den Ausführungs-Servern interpretiert.

#### 3.3.3.2. Dokumentinhalt

Dieser Teil des Metadokuments enthält den Inhalt des Dokuments. Er wird vom Ausführungs-Server als opake Bit-Folge betrachtet. Dokumentmethoden haben sowohl schreibenden als auch lesenden Zugriff auf den Dokumentinhalt.

#### 3.3.3.3. Attribute

In diesem Teil des Metadokuments werden Informationen hinterlegt, die über den Inhalt des Dokuments und die dokumentspezifische Zuordnungsfunktion hinausgehen. Zu diesen Informationen gehören die Identität des Autors, Informationen zur Integrität und Authentizität der Zuordnungsfunktion, des Inhalts und der Methoden. Allen Attributen ist gemein, daß sie nur vom Dokumentautor und von Ausführungs-Servern modifiziert werden können. Dokumentmethoden besitzen keinen schreibenden Zugriff auf die Attribute.

Attribute können verwendet werden, um Informationen vom Dokumentautor zu den Ausführungs-Servern der Digitalen Bibliothek zu übermitteln. Damit Attribute von Ausführungs-Servern interpretiert werden können, müssen ihre Namen und das Format der in ihnen enthaltenen Daten systemweit eindeutig festgelegt werden. Folgende systemweiten Attribute werden in Indigo definiert:

**Author** Ist dieses Attribut vorhanden, enthält es ein Zertifikat des Dokumentautors.

**ContentSignature** Ist dieses Attribut vorhanden, enthält es eine Signatur des Autors über den Dokumentinhalt. Diese Signatur dient der Kontrolle der Integrität des Dokumentinhalts.

**MethodDigests** Ist dieses Attribut vorhanden, enthält es vom Dokumentautor erstellte Prüfsummen der Dokumentmethoden. Diese dienen zur Kontrolle der Integrität der Dokumentmethoden und ermöglichen so die Plazierung von Methoden in vom Autor nicht kontrollierten Dokumentspeichern sowie die Verwendung nicht kontrollierter Kommunikationsinfrastrukturen. Letztere beinhalten z. B. Systeme zur Abbildung ortstransparenter Namen auf ortsabhängige Namen, wie sie bei der Abbildung von URLs mit Hilfe des Domain Name Systems (DNS) vorgenommen wird. Die Kontrolle der Integrität der Dokumentmethoden ist wichtig, um z. B. eine unerkannte Veränderung der Semantik des Dokuments bei der Präsentation zu vermeiden (vgl. [75]).

Zusätzlich zu den definierten Attributen, kann der Autor private Attribute im Metadokument speichern. Diese bleiben für die Infrastruktur transparent, werden aber beim Transport des Metadokuments kopiert. Private Attribute werden analog zu privaten Operationen realisiert, indem sie durch das Präfix *private.* gekennzeichnet werden. Diese Maßnahme erlaubt die spätere Erweiterung der Menge der Attribute, die vom Ausführungs-Server interpretiert werden, ohne daß die Gefahr einer Namenskollision mit privaten Attributen besteht.

### 3.3.4. Ausführungs-Server

Die Verarbeitung von Dokumenten in Indigo beruht auf der Interpretation des Metadokuments, der transparenten Bereitstellung von Dokumentmethoden und ihrer sicheren Ausführung. Diese Aufgaben werden von Ausführungs-Servern wahrgenommen. Ausführungs-Server stellen die standardisierten Laufzeitumgebungen bereit, führen in ihnen Dokumentmethoden aus und ermöglichen diesen den gesicherten Zugriff auf ihre Ressourcen. Zusätzlich speichern Ausführungs-Server Metadokumente und fungieren daher sowohl als anwenderseitige als auch als speicherseitige Komponenten.

Ausführungs-Server stellen ihre Funktionalität über eine Schnittstelle anderen Komponenten der Digitalen Bibliothek zur Verfügung. Diese Schnittstelle umfaßt Befehle zur Initiierung von Operationen sowie zum Austausch von Metadokumenten. Im einzelnen finden sich folgende Befehle in der Schnittstelle:

**Documents** Dieser Befehl besitzt keine Parameter und weist den Ausführungs-Server an, eine Liste der lokalen Namen der auf ihm gespeicherten Dokumente zurückzuliefern.

**Runtimes** Dieser Befehl besitzt keine Parameter und weist den Ausführungs-Server an, die Namen der verfügbaren Laufzeitumgebungen an den Aufrufer zurückzuliefern.

**Attributes** Dieser Befehl erhält einen lokalen Dokumentnamen als Argument und liefert die Attribute dieses Dokuments an den Aufrufer zurück.

**Operations** Dieser Befehl erhält ebenfalls einen lokalen Dokumentnamen als Argument und liefert eine Liste der Operationen zurück, die dieses Dokument unterstützt. Die Liste enthält sowohl die orthogonalen Operationen als auch die privaten Operationen.

**Invoke** Dieser Befehl erlaubt die Ausführung einer Operation auf einem Dokument und liefert deren Ergebnis an den Aufrufer zurück. Der Befehl erwartet den lokalen Dokumentnamen, den Operationsnamen und die Argumente für den Operationsaufruf als Argumente. Die Ausführung einer Operation beinhaltet die Interpretation der Zuordnungsfunktion, die transparente Bereitstellung der Dokumentmethode sowie ihre Ausführung in einer geeigneten Laufzeitumgebung. Der Invoke-Befehl erlaubt eine optionale Authentisierung des Initiators durch Signierung des Aufrufs durch den Initiator.

**Deliver** Dieser Befehl dient dem Transport eines Metadokuments auf einen Ausführungs-Server. Er erhält ein Metadokument als Argument. Der Ausführungs-Server weist dem Dokument einen im Kontext des Ausführungs-Servers eindeutigen Namen zu und liefert diesen als Ergebnis an den Aufrufer zurück. Optional kann nach der Speicherung des Metadokuments die Ausführung einer Operation erfolgen. In dem Fall werden als zusätzliche Parameter der Operationsname und die Argumente der Operation übergeben. Die Möglichkeit zur Authentisierung besteht hier ebenso wie bei dem Befehl Invoke.

**AddMethod** Dieser Befehl ermöglicht die Erweiterung bzw. Modifikation der dokumentspezifischen Zuordnungsfunktion eines Dokuments. Er erhält den lokalen Dokumentnamen, den Operationsnamen, den Methodennamen, den Namen der Laufzeitumgebung, in der die Methode ausgeführt werden kann und optional eine Prüfsumme über die Methode als Argumente. Der Befehl AddMethod erfordert die Authentisierung des Initiators durch Signierung des Aufrufs und wird nur durchgeführt, wenn der Initiator mit dem Autor identisch ist.

Die Digitale Bibliothek besteht aus einer Menge untereinander kommunizierender Ausführungs-Server (siehe Abschnitt 3.4), es gibt keine zentrale Komponente in Indigo.

### 3.3.5. Laufzeitumgebungen

Laufzeitumgebungen bilden die Umgebungen, in denen Dokumentmethoden ausgeführt werden. Sie sind durch zwei Merkmale charakterisiert, den Programmcode, den sie ausführen können, und die Schnittstellen, die sie bereitstellen. Die Schnittstellen dienen dem Zugriff auf Ressourcen der Ausführungs-Server. In Indigo können

daher nur Laufzeitumgebungen verwendet werden, die zum einen Zugriff auf alle notwendigen Ressourcen bieten und zum anderen die Durchsetzung von Sicherheitsanforderungen bei diesem Zugriff ermöglichen. Aus diesem Grund wurden in Indigo interpretierende Laufzeitumgebungen für die Ausführung von Dokumentmethoden ausgewählt. Die Vorteile der Verwendung interpretierter Sprachen sind die Homogenisierung der Plattformen sowie die Möglichkeit zur Durchsetzung von Sicherheitsanforderungen durch die Laufzeitumgebungen (siehe Abschnitt 3.2.2.3).

Die von Dokumentmethoden verwendeten Ressourcen lassen sich in zwei Klassen einteilen. Zum einen in die Ressourcen, die Bestandteil aller Laufzeitumgebungen sind. Dazu zählen z. B. Speichermedien, Kommunikationsmedien und Präsentationsmedien. Diese Ressourcen werden im folgenden als *elementare Ressourcen* bezeichnet. Jede Laufzeitumgebung in Indigo muß eine sichere Schnittstelle zu den elementaren Ressourcen bieten, so daß die Verfügbarkeit der notwendigen Ressourcen gewährleistet ist.

Die zweite Klasse von Ressourcen sind die Ressourcen des Ausführungs-Servers, die spezifisch für Indigo sind. Diese Ressourcen werden im folgenden als *Indigo-spezifische Ressourcen* bezeichnet. Zu ihnen zählt der Zugriff auf Dokumentattribute und -inhalte sowie die Möglichkeit zur Initiierung von Befehlen an der Schnittstelle eines Ausführungs-Servers. Die Laufzeitumgebungen müssen sichere Schnittstellen zum Zugriff auf Indigo-spezifische Ressourcen zusätzlich zu den Schnittstellen zu elementaren Ressourcen bereitstellen. Beim Entwurf von Indigo wurden Indigo-spezifische Ressourcen identifiziert, die sich in folgende Gruppen zusammenfassen: Server-Funktionen, Dokument-Funktionen und Traversierungs-Funktionen.

#### 3.3.5.1. Server-Funktionen

Diese Gruppe von Funktionen ermöglicht Dokumentmethoden den Zugriff auf die Schnittstelle des lokalen oder eines entfernten Ausführungs-Servers, also auf die in Abschnitt 3.3.4 beschriebenen Befehle Documents, Runtimes, Attributes, Operations, Invoke, Deliver und AddMethod. Die in den Laufzeitumgebungen bereitgestellten Funktionen entsprechen im wesentlichen diesen Befehlen. Einzig die Funktion zur Ausführung des Deliver-Befehls bildet hier eine Ausnahme.

Der Transport eines Dokuments auf einen entfernten Ausführungs-Server wird durch eine Dokumentmethode des Quelldokuments initiiert, indem diese die entsprechenden Funktionen in der Laufzeitumgebung aufruft. Der Transport selbst wird, transparent für die Dokumentmethoden, von den Ausführungs-Servern durchgeführt. In Indigo sind zwei Möglichkeiten zum Transport eines Dokuments vorgesehen, der Server-basierte Transport und der methodenbasierte Transport.

**Server-basierter Transport** Ein Dokument kann vollständig, d. h. mit seinem aktuellen Inhalt, der Zuordnungsfunktion und seinen Attributen von einem Ausführungs-Server auf einen anderen Ausführungs-Server übertragen werden. Nach der Übertragung existiert in dem entfernten Ausführungs-Server eine vollständige Kopie des Quelldokuments. Diese Art des Transports wird im folgenden als *Server-basierter Transport* bezeichnet, da der Transport vollständig vom Ausführungs-Server vorgenommen wird.

**Methodenbasierter Transport** Die zweite Möglichkeit zum Transport eines Dokuments auf einen entfernten Ausführungs-Server besteht in der Übertragung eines Metadokuments mit leerem Dokumentinhalt. Nach dieser Übertragung existiert in dem entfernten Ausführungs-Server eine Kopie der Zuordnungsfunktion und der Attribute des Quelldokuments. Der Zweck des methodenbasierten Transports besteht in der Übertragung der dokumentspezifischen Zuordnungsfunktion auf den entfernten Ausführungs-Server. Damit wird die in den Abschnitten 3.2.2.3 und 3.3.1 beschriebene dokumentspezifische Verteilung der Dokumentmethoden realisiert, denn nach dem Transport des inhaltslosen Metadokuments können Operationen des übertragenen Dokuments auf dem entfernten Ausführungs-Server instantiiert werden. Sollte die Übertragung des Dokumentinhalts gewünscht sein, kann dies unter Kontrolle der Dokumentmethoden, z. B. durch eine Kooperation zwischen anwenderseitiger und speicherseitiger Methode geschehen. Diese Art des Transports wird im folgenden als *methodenbasierter Transport* bezeichnet, da eine eventuelle Übertragung des Dokumentinhalts durch die Dokumentmethoden erfolgen muß.

Beispiele für die Nutzung des Server-basierten Transports bzw. des methodenbasierten Transports finden sich in Abschnitt 4.2.1 respektive in Abschnitt 4.2.2.

Im einzelnen finden sich folgende Funktionen in der Gruppe der Server-Funktionen:

- documents, runtimes — Diese Funktionen der Laufzeitumgebung rufen die jeweils entsprechenden Befehle an der Schnittstelle des Ausführungs-Servers auf, dessen Adresse als Argument übergeben wird, und liefern das Ergebnis an die aufrufende Dokumentmethode zurück.

- attributes, operations — Diese Funktionen erwarten die Adresse eines entfernten Ausführungs-Servers und den lokalen Namen eines Dokuments als Argumente, rufen die entsprechenden Befehle des entfernten Ausführungs-Servers auf und liefern das Ergebnis an die aufrufende Dokumentmethode zurück.

- invoke — Diese Funktion erwartet als Eingabe die Adresse eines entfernten Ausführungs-Servers, einen lokalen Dokumentnamen, einen Operationsnamen und eine Liste von Argumenten für die Operation. Die Funktion initiiert den Befehl Invoke an der Schnittstelle des entfernten Ausführungs-Servers mit den entsprechenden Argumenten und liefert das Ergebnis an die aufrufende Dokumentmethode zurück.

- invoke_async — Diese Funktion dient dem asynchronen Ausführen des Invoke-Befehls auf einem entfernten Ausführungs-Server. Sie verhält sich analog zu der invoke-Funktion, kehrt jedoch sofort nach der Initiierung des Invoke-Befehls zurück und übergibt dem Aufrufer eine Referenz, die den Aufruf identifiziert. Mit Hilfe dieser Referenz kann der Aufrufer den Zustand der Befehlsausführung prüfen und nach dem Ende der Befehlsausführung das Ergebnis ermitteln.

- add_method — Diese Funktion erwartet als Eingabe die Adresse eines entfernten Ausführungs-Servers, einen lokalen Dokumentnamen, einen Operationsnamen, den URI einer Dokumentmethode, den Namen einer Laufzeitumgebung und optional eine Prüfsumme über die Dokumentmethode. Die Funktion initiiert den

Befehl AddMethod an der Schnittstelle des entfernten Ausführungs-Servers mit den entsprechenden Argumenten und liefert das Ergebnis an die aufrufende Dokumentmethode zurück.

- s_deliver — Erwartet die Adresse eines Ausführungs-Servers als Argument und führt einen Server-basierten Transport des Dokuments, dessen Methode die Funktion s_deliver aufgerufen hat, zu dem angegebenen Server durch.

- m_deliver — Erwartet die Adresse eines Ausführungs-Servers als Argument und führt einen methodenbasierten Transport des Dokuments, dessen Methode die Funktion m_deliver aufgerufen hat, zu dem angegebenen Server durch.

- get_initiator — Diese Funktion ermöglicht der aufrufenden Dokumentmethode, die Identität des Initiators der entsprechenden Operation zu lesen, falls dieser sich authentisiert hat. Die Identität des Initiators wird zwischen den Ausführungs-Servern, die sich gegenseitig vertrauen (siehe Abschnitt 3.4.3), weitergereicht und ermöglicht so die Authentisierung des Initiators auch über den Aufruf mehrerer Operationen hinweg.

#### 3.3.5.2. Dokument-Funktionen

Diese Gruppe von Funktionen stellt Dokumentmethoden den Zugriff auf den Inhalt und die Attribute von Dokumenten sowie auf persistenten Speicher des Ausführungs-Servers bereit. Die Gruppe umfaßt folgende Funktionen:

- open, close, read, write — Diese Funktionen stellen eine lokale Schnittstelle zum Dokumentinhalt auf der Ebene der Bit-Folge bereit. Das zu bearbeitende Dokument wird durch seinen lokalen Namen identifiziert. Die Ausführung dieser Operationen ändert den im Ausführungs-Server gespeicherten Inhalt des Dokuments und damit auch den Inhalt, der beim Transport des Dokuments an einen entfernten Ausführungs-Server übertragen wird.

  Die Verwendung dieser Funktionen entspricht im Modell der orthogonalen Digitalen Bibliotheken einem Zugriff auf die Speicherschnittstelle.

- delete — Die delete-Funktion erlaubt einer Dokumentmethode das Dokument, auf dem die Methoden ausgeführt wird, zu löschen.

- attribute, attribute_list — Diese Funktionen ermöglichen den Zugriff auf die Attribute eines durch seinen lokalen Namen identifizierten Dokuments und auf deren Inhalt. Die Funktionen attribute und attribute_list sind lokale Varianten der Funktion attributes aus der Gruppe der Server-Funktionen.

- set_static, get_static, remove_static — Diese Funktionen erlauben Dokumentmethoden den Zugriff auf ein persistentes Dictionary. Das Dictionary dient dazu, dokumentinterne Informationen über die Existenz der Ausführungskontexte unterschiedlicher Operationsausführungen hinaus zu speichern. Die Sichtbarkeit des Dictionary ist auf das Dokument beschränkt, dessen Methoden die

set_static-, get_static- und remove_static-Funktionen aufrufen. Die Existenzspanne des Dictionary ist auf die Existenzspanne des jeweiligen Dokuments auf dem Ausführungs-Server beschränkt.

- enter, leave — Diese Funktionen ermöglichen den Zugriff auf Semaphore zur Synchronisation konkurrierender Zugriffe auf die Daten eines Dokuments. Sie erwarten den Namen eines Semaphors als Argument. Die Funktion enter dient der Erzeugung des Semaphors, falls es nicht existiert und dem Betreten des, durch das Semaphor geschützten kritischen Abschnitts, die Funktion leave dient dem Verlassen des kritischen Abschnitts.

#### 3.3.5.3. Traversierungs-Funktionen

Diese Gruppe von Funktionen ermöglicht mobilen Dokumenten (siehe Kapitel 5) die Traversierung des Server-Graphen (siehe Abschnitt 3.4). Sie enthält u. a. Funktionen zur Ermittlung der Menge der benachbarten Server sowie zur Verwaltung von Markierungen. Die Funktionen, die in dieser Gruppe bereitgestellt werden, sind:

- get_neighbors — Diese Funktion liefert die Menge der Ausführungs-Server zurück, die zu dem Ausführungs-Server, auf dem die Funktion aufgerufen wurde benachbart sind (siehe Abschnitt 3.4).

- is_trusted — Diese Funktion erhält die Adresse eines Ausführungs-Servers als Argument und liefert Informationen darüber zurück, ob der Ausführungs-Server auf dem die Funktion aufgerufen wurde, dem Ausführungs-Server, dessen Adresse als Argument übergeben wurde, vertraut. Vertrauen bedeutet hier, daß der genannte Ausführungs-Server keine Authentisierungen fälscht, Dokumentmethoden korrekt ausführt und Metadokumente nicht eigenmächtig modifiziert. In der gegenwärtigen Realisierung ist die Menge der Ausführungs-Server, denen ein Ausführungs-Server traut gleich der Menge seiner Nachbarn.

- set_mark, remove_mark — Diese Funktionen ermöglichen mobilen Dokumenten das Hinterlassen und Löschen von Markierungen auf Ausführungs-Servern zum Zweck der Traversierung des Server-Graphen (siehe Abschnitt 5.2.3).

### 3.3.6. Dokumentmethoden und Methodenspeicher

Dokumentmethoden implementieren orthogonale und private Operationen. Sie sind spezifisch für ein Dokument und eine Laufzeitumgebung. Dokumentmethoden werden durch URIs identifiziert, was ihre dezentrale Speicherung in existierenden Repositorien, z. B. ftp- oder HTTP-Servern, erlaubt. Die Integrität der Dokumentmethoden kann durch die Verwendung von Prüfsummen (siehe Abschnitt 3.3.3.3) gewährleistet werden.

## 3.4. Topologie der Digitalen Bibliothek

Eine auf Indigo basierende Digitale Bibliothek besteht aus einer Menge interoperierender Ausführungs-Server, die auf allen, an der Digitalen Bibliothek beteiligten Rechnerknoten installiert sind. Dadurch wird sowohl auf der Seite des Anwenders als auch auf der Seite des Betreibers die transparente Bereitstellung und sichere Ausführung von Dokumentmethoden gewährleistet. Im vorliegenden Abschnitt werden die Eigenschaften anwenderseitiger und speicherseitiger Ausführungs-Server erläutert und auf die Konstituierung des Speichers der Digitalen Bibliothek durch speicherseitige Ausführungs-Server, die in einer Nachbarschaftsbeziehung zueinander stehen, eingegangen.

### 3.4.1. Speicherseitige Ausführungs-Server

Speicherseitige Ausführungs-Server sind Ausführungs-Server, die auf speicherseitigen Rechnerknoten (siehe Abschnitt 2.4.1) installiert sind. Die auf ihnen instantiierten Dokumentmethoden besitzen lokalen Zugriff auf die in den speicherseitigen Rechnerknoten archivierten digitalen Dokumente. Die Hauptaufgabe speicherseitiger Ausführungs-Server liegt, neben der Ausführung von Dokumentmethoden, in der Speicherung der Dokumentdaten. Die speicherseitigen Ausführungs-Server bilden gemeinsam den Speicher einer auf Indigo basierenden Digitalen Bibliothek. Speicherseitige Ausführungs-Server sind ständig verfügbar.

### 3.4.2. Anwenderseitige Ausführungs-Server

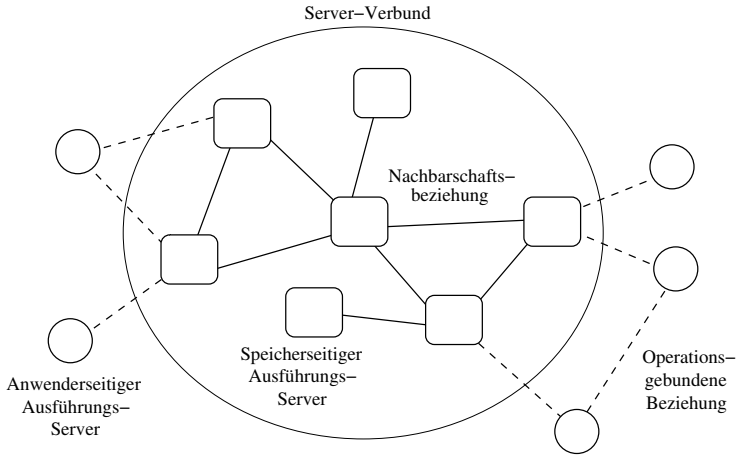
Anwenderseitige Ausführungs-Server sind auf anwenderseitigen Rechnerknoten (siehe Abschnitt 2.4.1) installiert. Die auf ihnen instantiierten Dokumentmethoden besitzen lokalen Zugriff auf die Anwenderkomponenten des jeweiligen Rechnerknotens. Anwenderseitige Rechnerknoten können ebenfalls Dokumente speichern. Aufgrund der Anonymität der anwenderseitigen Ausführungs-Server im Vergleich zu den speicherseitigen Ausführungs-Servern (siehe Abschnitt 3.4.3) ist ihre Existenz innerhalb der Infrastruktur nicht allgemein bekannt. Die Verfügbarkeit anwenderseitiger Ausführungs-Server unterliegt daher auch keinen Einschränkungen.

### 3.4.3. Verbund der Ausführungs-Server

Die Nutzung der Dokumente der Digitalen Bibliothek und die Erbringung von Diensten in der Digitalen Bibliothek erfordert die Kenntnis der eindeutigen Namen aller in der Digitalen Bibliothek gespeicherten Dokumente sowie die Möglichkeit des Zugriffs auf jedes einzelne Dokument. Speicherseitige Ausführungs-Server speichern jeweils eine Teilmenge aller Dokumente. Diese Teilmengen werden durch die Organisation der speicherseitigen Ausführungs-Server in einem Verbund zu einer Menge integriert. Dazu werden die speicherseitigen Ausführungs-Server in eine Nachbarschaftsbeziehung zueinander gesetzt, die transitiv alle Ausführungs-Server umfaßt. Die Menge der Ausführungs-Server und ihrer Nachbarschaftsbeziehungen bilden einen ungerichteten,

zusammenhängenden Graphen, wie in Abbildung 3.1 dargestellt. Dieser Graph wird im folgenden als *Server-Graph* oder *Server-Verbund* bezeichnet.

**Abbildung 3.1.** Struktur des Server-Verbunds




Die Kantenmenge ist verteilt in den speicherseitigen Ausführungs-Servern hinterlegt, indem jeder speicherseitige Ausführungs-Server die Namen der ihm bekannten Nachbarn speichert. Eine unidirektionale Verbindung von einem Ausführungs-Server A zu einem Ausführungs-Server B besteht genau dann, wenn der Ausführungs-Server B in der Nachbarschaftsmenge als Nachbar des Ausführungs-Servers A eingetragen ist.

Die bilaterale Übereinkunft über die Nachbarschaftsmengen macht die dezentrale Integration neuer speicherseitiger Ausführungs-Server in eine bestehende Digitale Bibliothek möglich. Um einen neuen speicherseitigen Ausführungs-Server in die Digitale Bibliothek zu integrieren, ist lediglich eine Übereinkunft mit den Betreibern der benachbarten Ausführungs-Server notwendig, da die Nachbarschaftsmenge eines Ausführungs-Servers nur unidirektionale Kanten enthält, die den Ausführungs-Server als Quelle besitzen. Um die Erreichbarkeit eines neu eingefügten Ausführungs-Servers zu gewährleisten, müssen die benachbarten Ausführungs-Server den neu eingefügten Ausführungs-Server ebenfalls in ihre Nachbarschaftsmenge aufnehmen. Daraus resultiert die Ungerichtetheit des Server-Graphen. Um den Zusammenhang des Server-Graphen zu gewährleisten, muß jeder neu eingefügte Server eine Kante zu mindestens einem existierenden Server besitzen. Das Eingliederungsprotokoll erfordert nicht nur die Modifikation der Nachbarmenge eines neu eingefügten Ausführungs-Servers sondern auch die Modifikation der Nachbarmenge der benachbarten Ausführungs-Server. In der vorliegenden Infrastruktur wird ein soziales Protokoll zur Durchführung dieser Modifikationen verwendet.

Mit diesen Randbedingungen ist die Topologie festgelegt, es handelt sich bei dem Server-Graphen um einen zusammenhängenden ungerichteten Graphen, dessen Knoten die speicherseitigen Ausführungs-Server der Digitalen Bibliothek sind und dessen Kanten die Nachbarschaftsbeziehungen zwischen diesen Knoten sind.

Es ist nicht gefordert, daß der Server-Graph ein Baum ist, was die Realisierung von Diensten vereinfachen würde. Es gibt zwei Gründe dafür, diese Einschränkung nicht zu treffen. Zum einen ist das Eingliederungsprotokoll einfacher, wenn die Baumeigenschaft nicht überwacht werden muß. Zum anderen machen die möglichen redundanten Kanten die Infrastruktur robuster gegenüber Ausfällen von Servern. Der Server-Graph bleibt zusammenhängend, solange nicht alle Server aus der Nachbarschaftsmenge eines Servers ausfallen. Der Preis für das einfache Eingliederungsprotokoll ist die Notwendigkeit zur Implementierung von Traversierungsstrategien zur Einführung neuer Dienste (siehe Kapitel 5).

Im Gegensatz zu speicherseitigen Ausführungs-Servern stehen anwenderseitige Ausführungs-Server in keiner permanenten Beziehung zu anderen Ausführungs-Servern. Allerdings gehen sie in Abhängigkeit von Dokument und Operation temporäre Beziehungen zu anderen Ausführungs-Servern ein. Diese werden aber nicht durch Nachbarschaftsmengen sondern durch den Austausch von Daten zwischen Dokumentmethoden etabliert. Sie sind somit außerhalb des Kontextes einer Operationsausführung nicht bekannt.

## 3.5. Zusammenfassung

Der Ausgangspunkt für den Entwurf von Indigo war die Behebung der bei der Analyse existierender Systeme festgestellten Defizite. Die grundlegenden Ideen von Indigo bestehen darin, die bei der Einführung neuer Dokumentformate unumgängliche manuelle Bearbeitung der Zuordnungsfunktion durch den Dokumentautor vornehmen zu lassen, diese Informationen in der Infrastruktur zu verteilen und mobile Programme zu nutzen, um Dokumentmethoden transparent auf anwenderseitigen und speicherseitigen Rechnerknoten bereitzustellen und so eine bandbreiteneffiziente Ausführung von Operationen zu ermöglichen.

Die Verteilung der Zuordnungsfunktion geschieht in der Form von Metadokumenten, die neben dem Inhalt eines Dokuments den dokumentspezifischen Teil der Zuordnungsfunktion speichern. Zusammen mit einer eindeutigen Benennung der Elemente der Zuordnungsfunktion, dies sind Dokumente, Operationen, Laufzeitumgebungen und Dokumentmethoden, können Metadokumente von Ausführungs-Servern genutzt werden, um transparent Dokumentmethoden bereitzustellen.

Aufgrund der dezentralen Architektur von Indigo, die Digitale Bibliothek besteht aus einer Menge lose gekoppelter Ausführungs-Server, ist die Einführung neuer anwenderseitiger oder speicherseitiger Ausführungs-Server und damit die Erhöhung der Zahl der unterstützten Anwender und Dokumente jederzeit möglich.

Die Erweiterung um neue Dokumenttypen erfolgt allein durch den Autor, indem dieser das entsprechende Metadokument mittels des Deliver-Befehls an einen speicherseitigen Ausführungs-Server übergibt.

Die Integration neuer orthogonaler Operationen geschieht bei neuen Dokumenten durch Berücksichtigung der Operationen bei der Erstellung des Metadokuments. Bei existierenden Dokumenten wird die Erweiterung um neuen orthogonale Operationen mittels des AddMethod-Befehls durchgeführt.

Im folgenden Kapitel wird beschrieben, wie die Mechanismen von Indigo genutzt werden können, um Operationen bandbreiteneffizient durchzuführen und um unterschiedlichste Dokumenttypen zu realisieren. Die transparente Integration neuer Dienste in Indigo ist in Kapitel 5 beschrieben.

# 4. Dokumente in der Digitalen Bibliothek

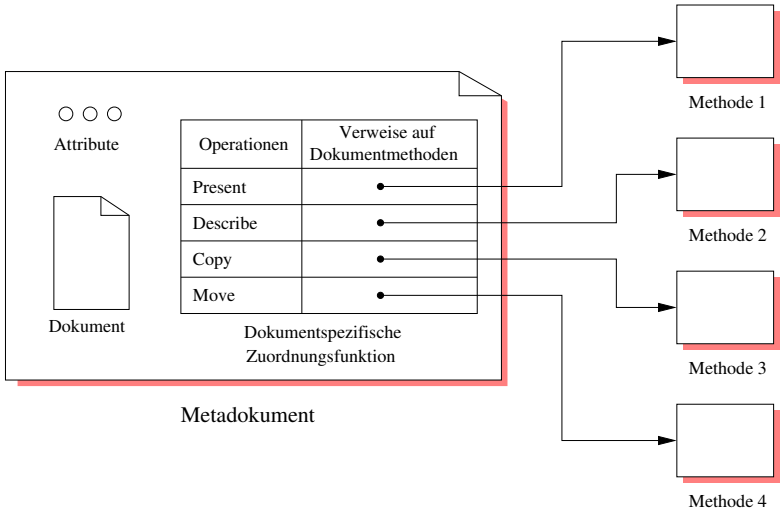
In Kapitel 3 wurde eine Infrastruktur für digitale Bibliotheken entworfen, die eine transparente Bereitstellung von Dokumentmethoden auf anwenderseitigen und auf speicherseitigen Rechnerknoten erlaubt. Um die Vorteile der Infrastruktur nutzen zu können, müssen Dokumente konstruiert werden, die an die entsprechenden Mechanismen der Infrastruktur angepaßt sind. Dabei kann bei der Erstellung neuer Dokumente auf bereits existierende Dokumente zurückgegriffen werden, denn Indigo ermöglicht die Wiederverwendung existierender Dokumente und Dokumentmethoden. Im vorliegenden Kapitel wird zunächst beschrieben, wie existierende Dokumente zur Konstruktion neuer Dokumente verwendet werden können und so als Muster für eine ganze Klasse von Dokumenten fungieren können. Daran schließt sich der Entwurf von Dokumentmustern für unterschiedliche Anforderungen an. Neben Dokumentmustern zur Präsentation wird ein Dokumentmuster zur Realisierung verteilungstransparenter Dokumente zur verteilten Datenhaltung sowie Dokumentmuster für Dokumente mit geschütztem Dokumentinhalt und kontrolliertem Dokumentzugriff entworfen. In Kapitel 5 werden dann Dokumentmuster vorgestellt, die die Integration von Diensten in eine Digitalen Bibliothek ermöglichen.

## 4.1. Dokumentmuster

Die vorgestellte verteilte Infrastruktur für erweiterbare orthogonale Digitale Bibliotheken basiert auf der Verwendung von Metadokumenten zur Beschreibung von Dokumentinhalten und dokumentspezifischen Zuordnungsfunktionen. Für jedes Dokument in Indigo muß daher ein entsprechendes Metadokument existieren, in dem jeder auf dem Dokument definierten Operation eine Dokumentmethode zugeordnet wird. Diese Dokumentmethode liegt in Form eines mobilen Programms auf einem allgemein zugänglichen Methodenspeicher vor (siehe Abschnitt 3.2.2). Abbildung 4.1 illustriert die Beziehung zwischen Dokumenten, Metadokumenten, Operationen und Methoden.

Betrachtet man Abbildung 4.1 fällt auf, daß die Struktur der Metadokumente Ähnlichkeit mit der Struktur der Klasse *Class* der objektorientierten Programmiersprache Smalltalk [40] besitzt. Diese Klasse dient in Smalltalk der Beschreibung der Eigenschaften von Klassen, sie verfügt u. a. über ein Dictionary, das Methodennamen der Implementierung einer Methode zuordnet. Jedes Objekt in Smalltalk verfügt über ein Attribut, das die Instanz von Class, die die Klasse des Objekts beschreibt, referen-

**Abbildung 4.1.** Dokument, Metadokumente, Operationen und Methoden




ziert.[1] Ein Dokument in Indigo läßt sich also, aufgrund der Trennung von Schnittstelle und Implementierung, mit einem Objekt in Smalltalk vergleichen. Die Korrespondenz zwischen den Elementen eines Dokuments der Infrastruktur und einem Objekt in Smalltalk ist in Tabelle 4.1 dargestellt.

**Tabelle 4.1.** Korrespondenz zwischen Elementen von Indigo und Smalltalk

| Dokument in Indigo | Objekt in Smalltalk |
|---|---|
| Dokumentinhalt | Attribute |
| Operationen | Nachrichten |
| Dokumentmethoden | Methoden |
| Zuordnungsfunktion | Instanz der Class-Klasse |

In der Tabelle wird deutlich, daß die Verwendung von Operationsnamen und die dokumentspezifische Abbildung der Operationsnamen auf Dokumentmethoden dem Nachrichtenkonzept in Smalltalk und damit der Grundlage für den Polymorphiemechanismus in Smalltalk entspricht.

Es gibt allerdings auch Unterschiede zwischen Dokumenten in Indigo und Objekten in Smalltalk. Der augenfälligste besteht darin, daß in Indigo die Abbildung von Operationen auf Dokumentmethoden und der Dokumentinhalt in dem gleichen Element, dem

[1] Die Struktur in Smalltalk ist in späteren Versionen der Sprache geändert worden. Seitdem besitzt jede Klasse X eine eigene XClass-Klasse. Die Änderung diente vor allem der Einführung klassenspezifischer Konstruktoren und wird daher hier nicht berücksichtigt.

Metadokument gekapselt sind. In Smalltalk wird diese Abbildung in einer Instanz von Class gespeichert, die von den entsprechenden Objekten lediglich referenziert wird.[2]

Im Gegensatz zu objektorientierten Sprachen stellt Indigo keine Mechanismen zur Unterstützung von Vererbung bei der Definition von Dokumenten bereit. Dennoch lassen sich sowohl Vererbung als auch Polymorphie durch Kopieren und modifizieren existierender Metadokumente erreichen. Vererbung läßt sich durch Kopieren eines Metadokuments und Erweiterung um neuen private Operationen erreichen. Polymorphie kann durch Kopieren eines Metadokuments und Modifikation der Zuordnungsfunktion erreicht werden.[3] Die eindeutige Benennung der Dokumentmethoden und ihre transparente Bereitstellung ermöglichen die Nutzung der existierenden Dokumentmethoden im neuen Dokument.

Das Dokument, aus dem die Zuordnungsfunktion übernommen wurde, dient als eine Art Instantiierungsvorlage. Das Erben von Objekten anstatt von Klassen ist zwar ungewöhnlich wird aber auch an anderer Stelle verwendet, z. B. in der objektorientierten Programmiersprache Self [95], die keine Klassen kennt.

Die Initiierung einer Operation an einem Dokument in Indigo entspricht dem Senden einer Nachricht an ein Objekt in einer objektorientierten Sprache. Der Transport einer Dokumentmethode auf einen Ausführungs-Server und ihre anschließende Ausführung ist mit der Ausführung einer Methode eines Objekts vergleichbar.

Aufgrund der oben geschilderten Möglichkeiten zu Vererbung und Polymorphie ist es sinnvoll, Dokumentmuster anzugeben, die als Vorlage für die Erstellung neuer Dokumente dienen. Je nach Struktur der Dokumentmuster lassen sich nur die Schnittstellen oder die Schnittstellen und die Dokumentmethoden wiederverwenden. Dokumentmuster werden im folgenden durch die Operationen, die sie verwenden und eine Beschreibung der Semantik dieser Dokumentmethoden definiert. Wo generische Dokumentmethoden wiederverwendet werden können wird dies erwähnt.

## 4.2. Dokumentmuster für die Präsentation

Präsentation ist für nahezu alle Dokumente eine wesentliche Operation. Das Ziel der Präsentation ist die Übertragung der Daten vom Speichermedium zum Präsentationsmedium (vgl. [88]). Wie in Kapitel 2 beschrieben, können je nach Dokumenttyp unterschiedliche Verteilungen der an der Präsentation beteiligten Programmfunktionen sinnvoll sein. Wie die Verteilung im einzelnen vorgenommen wird, hängt von vielen Faktoren ab, u. a. von der zeitbezogenen Charakteristik der Daten (kon-

[2] Auf den Vergleich mit C++ [90] wurde hier bewußt verzichtet, da die Auflösung von Namen zur Übersetzungszeit und ihre Abbildung auf anonyme Zeiger nur Raum für den Begriff des „Methodenaufrufs“ läßt. Dieser schließt, in Ermangelung eines Namens für den Vorgang der Zeigerdereferenzierung, den Vorgang der Zeigerdereferenzierung mit ein. Nachricht und Methode sind daher, anders als z. B. in Smalltalk, Self [95] und Objective C [48] begrifflich nicht getrennt. Letztere Programmiersprache führt zur Auffindung einer Methode tatsächlich Vergleiche zwischen Nachrichten, die in diesem Fall durch Zeichenketten dargestellt werden, durch.

[3] Dies entspricht dem Vorgang, den Objektive C [48] oder C++-Compiler [90] bei der Übersetzung von Programmen durchführen. Die Kopie der Zuordnungsfunktion entspricht der Einbindung der Attribute der Basisklasse, die Anpassung der Zuordnungsfunktion entspricht der Erweiterung der Klassenhierarchie in Objektive C bzw. der Erstellung einer entsprechenden Virtual Function Table in C++.

tinuierlich oder synchron), dem Repräsentationsmedium, dem Präsentationsmedium und dem verfügbaren Übertragungsmedium. Obwohl diese Merkmale stark variieren können, herrschen zwei Arten der Präsentation vor, die Offline-Präsentation und die Online-Präsentation. Daneben gewinnt die kooperative Nutzung eines Dokuments durch mehrere Nutzer zunehmend an Bedeutung.

Unter Offline-Präsentation wird die Präsentation von Daten verstanden, die lokal auf dem anwenderseitigen Ausführungs-Server, unabhängig von weiteren Rechnerknoten durchgeführt wird (vgl. [24, S. 19]). Eine Voraussetzung für die Offline-Präsentation ist die vorhergehende Übertragung der Daten auf den anwenderseitigen Ausführungs-Server.

Mit Online-Präsentation wird die Präsentation bezeichnet, bei der die Daten zum Präsentationszeitpunkt von Dokumentmethoden zwischen dem speicherseitigen und dem anwenderseitigen Ausführungs-Server transportiert werden, also ein dokumentspezifisches Protokoll verwendet wird (vgl. [24, S. 19]).

Die dritte Art der Präsentation ist die kooperative Nutzung eines Dokuments durch eine Menge von Benutzern, die sich mit der zunehmenden Verfügbarkeit von Rechnernetzen entwickelt hat, und vor allem im Gebiet des Computer Supported Cooperative Work (CSCW) Einsatz findet.

In den folgenden Abschnitten werden Dokumentmuster für die drei Präsentationsarten vorgestellt und erläutert, wie diese durch spezifische Dokumentmethoden zur Ausführung der Present-Operation realisiert werden können.

### 4.2.1. Offline-Präsentation

Die Offline-Präsentation läuft, nach einer Etablierungsphase, vollständig in dem anwenderseitigen Ausführungs-Server ab. Die Offline-Präsentation erfordert den Transport der zu präsentierenden Daten auf den anwenderseitigen Ausführungs-Server und die anschließende Initiierung einer entsprechenden Operation zur lokalen Durchführung der Präsentation. Das Dokumentmuster für die Offline-Präsentation definiert die folgenden Operationen zur Präsentation eines Dokuments auf dem anwenderseitigen Ausführungs-Server:

**Present** Die orthogonale Present-Operation dient dem Transport des Dokuments auf den anwenderseitigen Ausführungs-Server und dem anschließenden Start der private.localPresent-Operation auf dem anwenderseitigen Ausführungs-Server. Die Dokumentmethode zur Implementierung der Present-Operation ist generisch.

**private.localPresent** Die private.localPresent-Operation wird im anwenderseitigen Ausführungs-Server ausgeführt. Sie liest den Dokumentinhalt vom lokalen Speicher des Ausführungs-Servers und präsentiert ihn dem Anwender. Die Dokumentmethode ist spezifisch für das Dokumentformat bzw. den Dokumentinhalt.

Im einzelnen verläuft die Offline-Präsentation in folgenden Schritten:

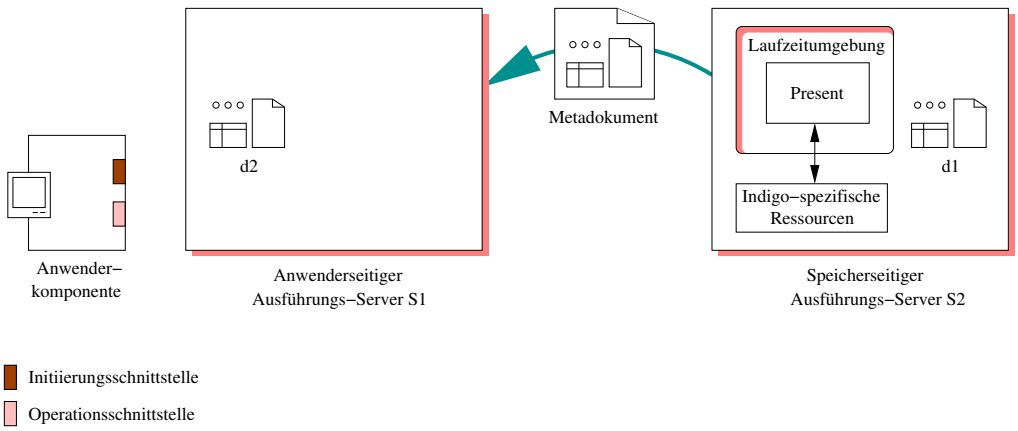
1. Der Benutzer initiiert mittels des Invoke-Befehls die Ausführung der Present-Operation an einem Dokument d1, das sich auf dem Ausführungs-Server S2 befindet. Als Argument übergibt er den Zugangspunkt des anwenderseitigen Ausführungs-Servers, der im folgenden mit S1 bezeichnet wird (siehe Abbildung 4.2).

**Abbildung 4.2.** Initiierung der Present-Operation


2. Die Dokumentmethode, die die Present-Operation des Dokuments d1 implementiert, überträgt das Dokument d1 mittels des Server-basierten Transports an den durch den Zugangspunkt identifizierten Ausführungs-Server, also an S1. Dazu verwendet sie die Schnittstelle zu den Indigo-spezifischen Ressourcen.

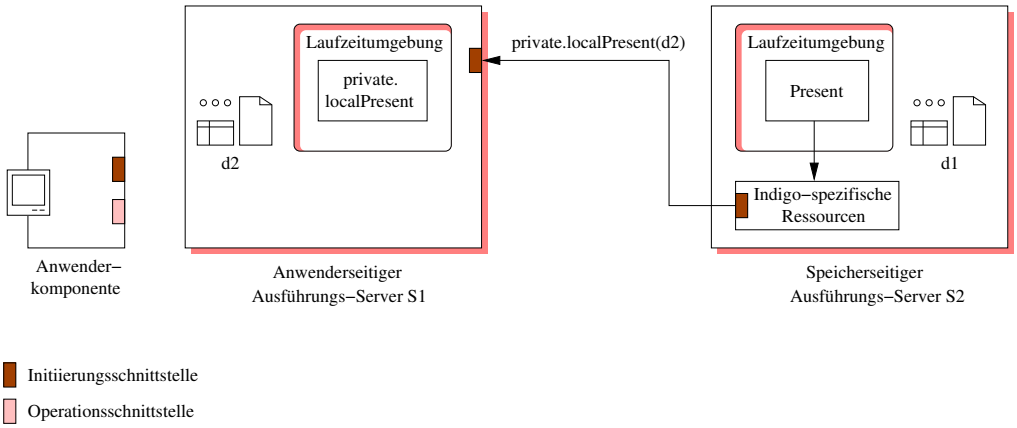
   Als Resultat der Übertragung existiert das Dokument d2, das eine Kopie des Dokuments d1 ist, im Ausführungs-Server S1 (siehe Abbildung 4.3).

**Abbildung 4.3.** Server-basierter Transport zum anwenderseitigen Ausführungs-Server



3. Nach der erfolgreichen Übertragung initiiert die Dokumentmethode, die die Present-Operation implementiert die Ausführung der private.localPresent-Operation an dem Dokument d2 (siehe Abbildung 4.4).

**Abbildung 4.4.** Initiierung der private.localPresent-Operation



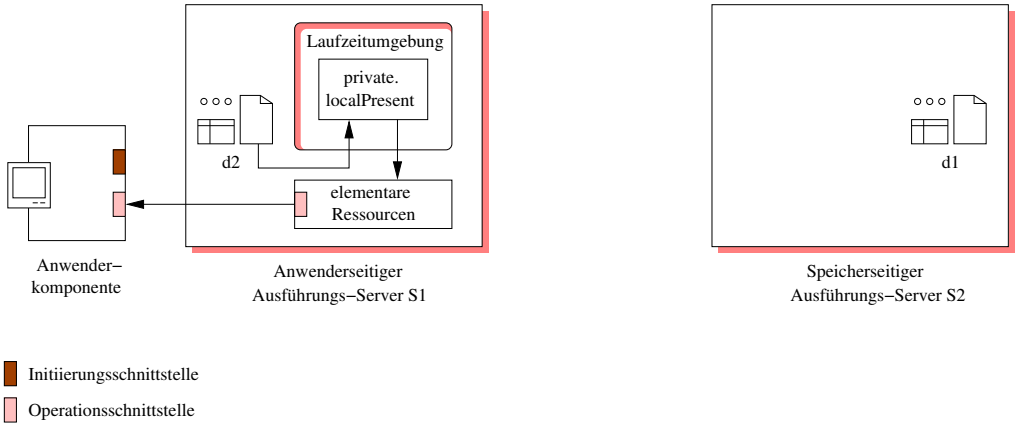

War die Initiierung der private.localPresent-Operation erfolgreich, liefert die Methode eine positive Quittung an die initiierende Anwenderkomponente und beendet sich. In dem Ausführungs-Server S2 existiert nun kein Ausführungskontext mehr, der mit der Präsentation in Zusammenhang steht.

4. Die private.localPresent zugeordnete Dokumentmethode verwendet die vom Ausführungs-Server bereitgestellten Schnittstellen zur Präsentation des Dokumentinhalts von Dokument d2. Sie greift dabei über die Schnittstelle zu den elementaren Ressourcen auf die lokalen Speicher- und Präsentationsmedien des Rechnerknotens zu, auf dem sich der Server S1 befindet, und kommuniziert mit der Anwenderkomponente (siehe Abbildung 4.5).

5. Nachdem die Präsentation beendet ist, entfernt die der private.localPresent-Operation zugeordnete Methode das Dokument d2 aus dem Ausführungs-Server S1 und beendet sich. Damit existiert in S1 kein Kontext mehr, der mit der Präsentation in Zusammenhang steht.

Die Offline-Präsentation ist für Präsentationen geeignet, die einen wahlfreien Zugriff auf den Dokumentinhalt benötigen, da dieser lokal vorliegt und ein Zugriff keinen Netzwerkverkehr erfordert. Typische Offline-Dokumente sind Texte, Bilder und andere zeitdiskrete Medien, aber auch interaktive Dokumente, Spiele etc. Beispiele für die Realisierung von Offline-Dokumenten finden sich in Abschnitt 6.3.1. Die Offline-Präsentation ist nur bedingt für zeitkontinuierliche Medien geeignet, da die Dokumente hierzu einen zu großen Datenumfang aufweisen.

**Abbildung 4.5.** Offline-Präsentation des Dokuments




### 4.2.2. Online-Präsentation

Wie bereits oben erwähnt eignet sich die Offline-Präsentation nur begrenzt für den Umgang mit zeitkontinuierlichen Medien. In diesem Abschnitt wird daher ein Dokumentmuster zur Online-Präsentation vorgestellt, das die Realisierung eines dokumentspezifischen Protokolls zur Übertragung zeitkontinuierlicher Daten zwischen anwenderseitigem und speicherseitigem Ausführungs-Server erlaubt. Dazu werden Dokumentmethoden auf dem anwenderseitigen und auf dem speicherseitigen Ausführungs-Server ausgeführt. Die folgenden Operationen sind in die Online-Präsentation involviert:

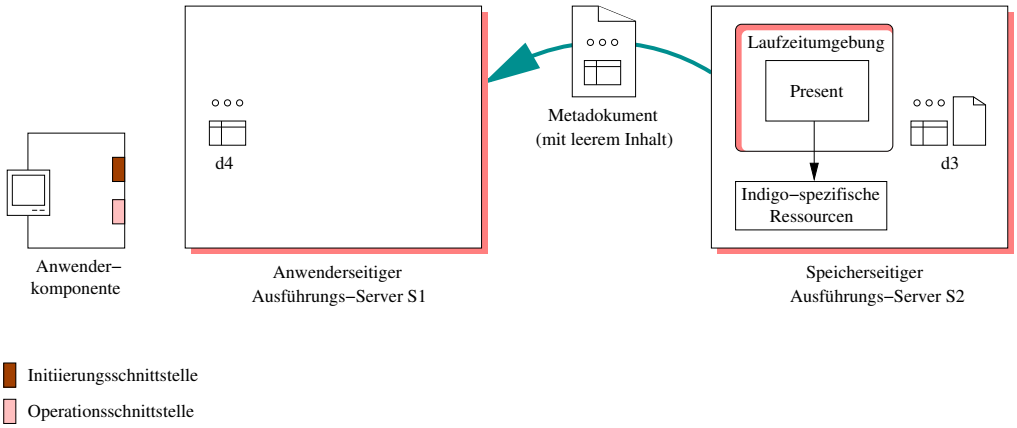
**Present** Die orthogonale Present-Operation dient der Initiierung eines methodenbasierten Transports des Dokuments auf den anwenderseitigen Ausführungs-Server sowie dem anschließenden Transport des Dokumentinhalts auf den anwenderseitigen Ausführungs-Server. Die Dokumentmethode ist spezifisch für das Format des Dokumentinhalts sowie für das Protokoll, das zum Transport der Daten verwendet wird.

**private.localPresent** Die der private.localPresent-Operation zugeordnete Methode wird im anwenderseitigen Ausführungs-Server ausgeführt. Sie kommuniziert mit der Methode zur Realisierung der Present-Operation, um den Dokumentinhalt zu lesen und interagiert über die Benutzerschnittstelle mit dem Anwender. Die Dokumentmethode ist spezifisch für das verwendete Protokoll.

Im einzelnen verläuft die Online-Präsentation in folgenden Schritten:

1. Der Anwender initiiert die Ausführung der Present-Operation des Dokuments d3 auf dem Ausführungs-Server S2. Als Argument übergibt er den Zugangspunkt des von ihm verwendeten Ausführungs-Servers S1. Dieser Schritt ist aufgrund der Orthogonalität der Operationen analog zum ersten Schritt der Offline-Präsentation.

**Abbildung 4.6.** Methodenbasierter Transport zum anwenderseitigen Ausführungs-Server




2. Die der Present-Operation zugeordnete Methode des Dokuments d3 überträgt das Dokument auf den anwenderseitigen Ausführungs-Server S1. Zur Übertragung wird im Gegensatz zur Offline-Präsentation der methodenbasierte Transport verwendet, der ein Metadokument ohne Inhalt überträgt (siehe Abbildung 4.6). Als Resultat der Übertragung steht das Dokument d4 mit leerem Inhalt auf dem Ausführungs-Server S1 zur Verfügung.

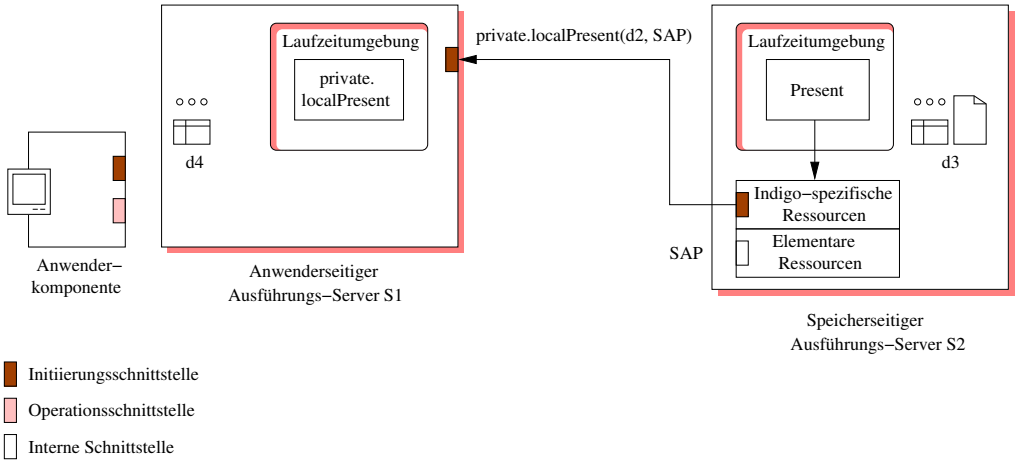
3. Nach der erfolgreichen Übertragung des Dokuments auf den Server S1 erzeugt die der Present-Operation zugeordnete Methode des Dokuments d3 einen Kommunikationsendpunkt (Service Access Point, SAP) und initiiert die Ausführung der private.localPresent-Operation des Dokuments d4, wobei sie den Namen des soeben erzeugten SAPs als Parameter übergibt (siehe Abbildung 4.7).

   Die Signatur der private.localPresent-Operation des Dokuments d2 unterscheidet sich also von der Signatur der private.localPresent-Operation des Dokuments d4. Im Gegensatz zu der Present-Operation des in Abschnitt 4.2.1 beschriebenen Dokuments d1 beendet sich die Present-Operation des Dokuments d3 nicht, sondern wartet an dem Kommunikationsendpunkt auf eine Verbindung.

4. Die der private.localPresent-Operation zugeordnete Methode des Dokuments d4 verwendet die Schnittstelle zu den elementaren Ressourcen des Ausführungs-Servers S1, um die der Present-Operation zugeordnete Methode des Dokuments d3 über den erzeugten SAP zu kontaktieren. Über diesen Kommunikationskanal empfängt bzw. sendet sie die zur Präsentation des Dokuments notwendigen Daten. Gleichzeitig greift die der private.localPresent-Operation zugeordnete Methode des Dokuments d4 auf die Präsentationsmedien des anwenderseitigen Ausführungs-Servers zu und präsentiert den Inhalt des Dokuments (siehe Abbildung 4.8).

5. Nachdem die Präsentation beendet ist, entfernt die der private.localPresent-

**Abbildung 4.7.** Initiierung der private.localPresent-Present am anwenderseitigen Ausführungs-Server




Operation des Dokuments d4 zugeordnete Methode das Dokument d4 von dem Ausführungs-Server S1 und beendet sich. In S1 existiert damit kein mit der Präsentation in Zusammenhang stehender Ausführungskontext mehr.

6. Nachdem sich die der private.localPresent-Operation zugeordnete Methode beendet hat, beendet sich die der Present-Operation des Dokuments d3 zugeordnete Methode ebenfalls. Damit existiert auch in dem Server S1 kein Ausführungskontext mehr, der mit der Präsentation in Verbindung steht.

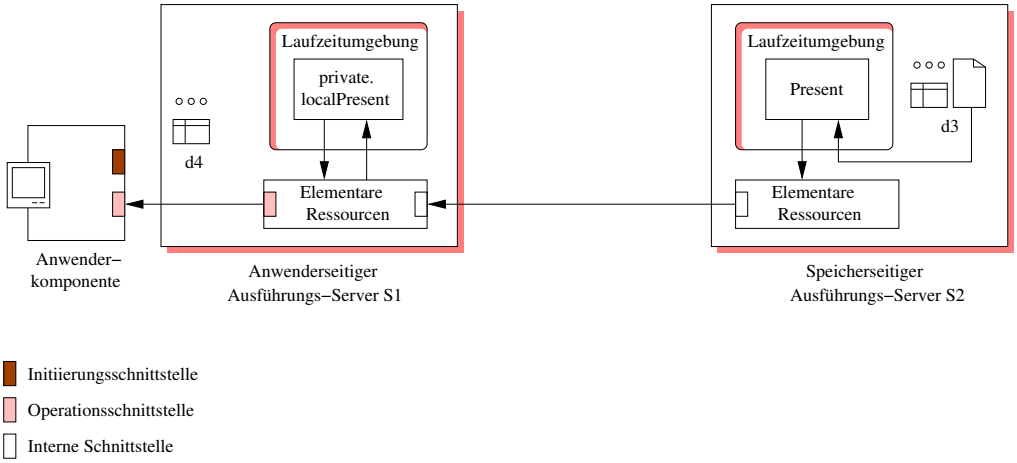
Diese Art der Präsentation eignet sich für Dokumente mit umfangreichen Daten, also vor allem für zeitkontinuierliche Daten, wie Audio- und Video-Dokumente. Die Online-Präsentation erlaubt aufgrund der Interaktion des Benutzers mit den Präsentationsmethoden auch die selektive Übertragung von Daten, z. B. die Übertragung von Videosequenzen und von Bildausschnitten. Ein Beispiel für die Realisierung von Online-Dokumenten findet sich in Abschnitt 6.3.2.

### 4.2.3. Kooperativ genutzte Dokumente

Die in Abschnitt 4.2.2 vorgestellten Dokumente unterstützten, aufgrund ihrer Struktur jeweils nur einen Benutzer und eignen sich damit nicht für eine Kooperation zwischen mehreren Benutzern, wie sie z. B. im Bereich des CSCW in der Form von Whiteboards oder bei Chats vorkommt.

In diesem Abschnitt wird ein Dokumentmuster vorgestellt, das diese simultane Nutzung eines Dokuments durch mehrere Anwender unterstützt, wobei der Dokumentinhalt zentral auf dem speicherseitigen Ausführungs-Server gehalten wird, auf dem sich das Dokument befindet. Dazu werden die oben vorgestellten Dokumentmuster um die Operation private.startServer erweitert. Das Dokumentmuster für kooperative Nutzung enthält somit folgende Operationen:

**Abbildung 4.8.** Online-Präsentation des Dokuments




**Present** Die orthogonale Present-Operation dient dem Transports der Zuordnungsfunktion des Dokuments auf den anwenderseitigen Ausführungs-Server sowie gegebenenfalls der Initiierung einer Operation zur zentralen Verwaltung des Dokumentinhalts. Die der Present-Operation zugeordnete Methode ist generisch.

**private.startServer** Die private.startServer-Operation dient dem Starten einer Methode zur Verwaltung des Dokumentinhalts und zur Kommunikation mit den anwenderseitigen Methoden.

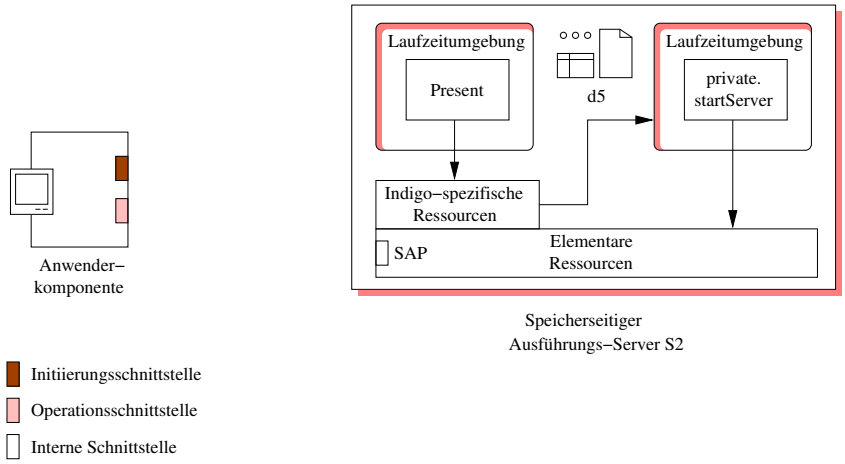
**private.localPresent** Die der private.localPresent-Operation zugeordneten Methoden werden in den anwenderseitigen Ausführungs-Servern ausgeführt. Sie kommunizieren mit der speicherseitigen Methode zur Verwaltung des Dokumentinhalts und interagieren über die Benutzerschnittstelle mit dem Anwender.

Folgende Schritte zur Präsentation werden durchgeführt:

1. Ein erster Benutzer initiiert die Ausführung der der Present-Operation zugeordneten Methode des Dokuments d5 auf dem Ausführungs-Server S2. Als Argument übergibt er den Zugangspunkt des von ihm verwendeten Ausführungs-Servers S1. Dieser Schritt ist, aufgrund der Orthogonalität, analog zum ersten Schritt der Offline-Präsentation sowie zum ersten Schritt der Online-Präsentation.

2. Falls noch kein lokaler Server für das kooperative Dokument existiert, startet die der Present-Operation zugeordnete Methode des Dokuments d5 den lokalen Server durch Initiierung der private.startServer-Operation. Diese installiert einen Server für das Dokument d5, der dann unabhängig von der Existenz einer der Present-Operation zugeordneten Methode auf dem Server S2 ausgeführt wird (siehe Abbildung 4.9).

**Abbildung 4.9.** Initiierung von private.startServer auf dem speicherseitigen Ausführungs-Server



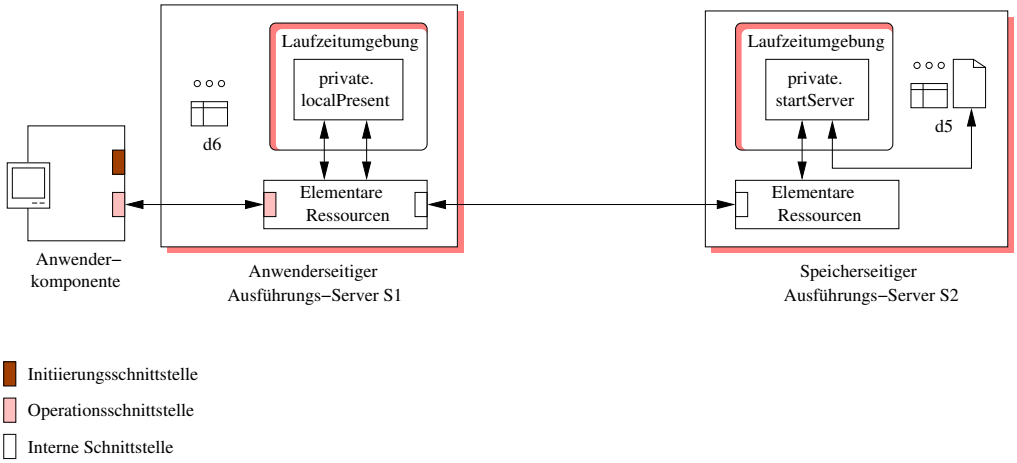

3. Wenn der Server für das Dokument d5 existiert, transportiert die der Present-Operation des Dokuments d5 zugeordnete Methode das Dokument d5 mittels methodenbasiertem Transport auf den Ausführungs-Server des Anwenders. Als Resultat der Übertragung wird das Dokument d6 mit leerem Inhalt auf dem Ausführungs-Server S1 erzeugt. Dieses Vorgehen ist analog zu dem Transport eines leeren Dokuments auf den Ausführungs-Server des Anwenders bei der Präsentation eines Online-Dokuments.

4. Nach der erfolgreichen Übertragung des Dokuments auf den Server S1 startet die der Present-Operation des Dokuments d5 zugeordnete Methode die private.localPresent-Operation des Dokuments d6, wobei sie den SAP des Servers als Parameter übergibt. Danach beendet sich die der Present-Operation zugeordnete Methode. Die der private.localPresent-Operation zugeordnete Methode des Dokuments d6 kontaktiert den SAP des Servers auf S2 und führt die Präsentation durch (siehe Abbildung 4.10).

5. Zu jedem Zeitpunkt können weitere Anwender die Ausführung der Present-Operation des Dokuments d5 initiieren. Dies führt zur Übertragung weiterer leerer Kopien des Dokuments d5 auf die entsprechenden Ausführungs-Server. In Abbildung 4.10 ist das Ergebnis der Initiierung dieser Present-Operation des Dokuments d5 dargestellt, wobei die Adresse des Ausführungs-Servers S3 als Argument übergeben wurde. Als Resultat dieses Aufrufs existiert nun das Dokument d7 auf dem Server S3. Die private.localPresent-Methode von d7 kommuniziert mit dem Server auf S2, um die Präsentation durchführen zu können (siehe Abbildung 4.11).

6. Beendet ein Anwender die der private.localPresent-Operation zugeordnete Methode, meldet sich diese vom Server auf S2 ab und entfernt ihr Dokument von anwenderseitigen Ausführungs-Server. Das Beenden der der private.localPresent-Operation des Dokuments d6 zugeordneten Methoden

**Abbildung 4.10.** Nutzung eines kooperativen Dokuments durch einen Anwender




beispielsweise führt zur Entfernung des Dokuments d6 vom Ausführungs-Server S1.

7. Nach dem Beenden der letzten Methode, die eine Verbindung zu ihm hat, kann der Server entscheiden, ob er sich selbst beendet oder existent bleibt. Diese Entscheidung ist natürlich dokumentspezifisch. Sollte sich der Server beenden, muß er bei der ersten erneuten Initiierung der Present-Operation von d5 wieder gestartet werden.

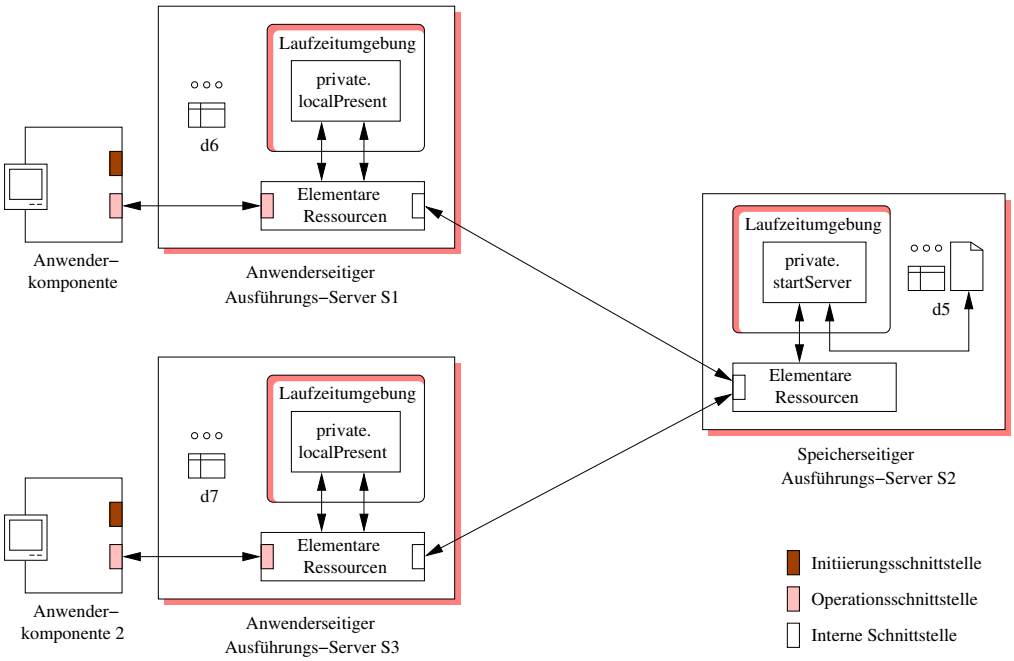
Das dynamische Starten und Stoppen der Server erlaubt die Realisierung von Persistenztransparenz und ermöglicht so, falls das mit dem Zweck des Servers vereinbar ist, die Beschränkung der Ressourcennutzung auf das Zeitintervall, in dem tatsächlich Anwender das Dokument nutzen. Ein Beispiel für die Realisierung eines kooperativen Dokuments, das einen Chat-Raum realisiert, findet sich in Abschnitt 6.3.3.

Mit den hier vorgestellten Präsentationsmustern sind die Möglichkeiten zur Präsentation natürlich nicht erschöpft. Die vorgestellten Muster verstehen sich als Beispiele für einige häufig vorkommende Dokumenttypen. Aufgrund der Flexibilität von Indigo können andere Mechanismen jederzeit implementiert werden. So sind z. B. kooperative Dokumentmuster möglich, in denen die anwenderseitigen Methoden der beteiligten Parteien direkt miteinander kommunizieren.

## 4.3. Dokumentmuster für verteilte Datenhaltung

Dokumente in Indigo speichern nicht nur Inhalte, die für Benutzer gedacht sind, sondern können auch die Funktion der Infrastruktur selbst unterstützen. In einem verteilten System, wie der vorliegenden Infrastruktur, werden z. B. Namensdienste benötigt, in einer Digitalen Bibliothek wird ein Katalogdienst benötigt. Diese Dienste setzen aufgrund der umfangreichen Daten, die sie verarbeiten, Mechanismen zur dezentralen

**Abbildung 4.11.** Nutzung des kooperativen Dokuments durch einen weiteren Anwender




Speicherung von Daten voraus. Eine Möglichkeit zur Bereitstellung dieser Mechanismen besteht in ihrer Implementierung in den Ausführungs-Servern. Der Vorteil dieses Ansatzes ist die Effizienz und allgemeine Verfügbarkeit der Dienste. Sein Nachteil ist, daß durch die Implementierung der Konsistenzstrategie in den Servern eine a priori Festlegung des Verteilungs- und Konsistenzmodells getroffen wird, die nicht allen Anforderungen gerecht werden kann (vgl. [86, 87]).

Indigo bietet eine Alternative zur Integration von Verteilungs- und Konsistenzstrategien in die Ausführungs-Server: die Realisierung dieser Strategien mittels Dokumentmethoden. Dadurch lassen sich die verwendeten Strategien den tatsächlichen Anforderungen spezifischer Dokumente anpassen. In diesem Abschnitt wird ein Dokumentmuster vorgestellt, das von dieser Möglichkeit Gebrauch macht, um eine für den Anwender transparente Verteilung von Daten zu realisieren. Das hier vorgestellte Dokumentmuster versteht sich als Beispiel für die Möglichkeiten, die die Infrastruktur bietet. Es belegt, daß die Infrastruktur nicht nur die Realisierung präsentationsorientierter Dokumente unterstützt, sondern auch Dokumente ermöglicht, die der Verwaltung der Infrastruktur dienen. Damit erweitert sich das Spektrum der Anwendungen, in denen individuell abgestimmte Methoden verwendet werden können.

### 4.3.1. Gegenseitiger Ausschluß

Die verteilte Speicherung von Daten in mehreren Dokumenten kann zu konkurrierenden Zugriffen auf replizierte Daten führen. Um die Konsistenz verteilt gespeicherter

Daten zu gewährleisten, ist eine Synchronisation der schreibenden Zugriffe auf die Daten notwendig. Eine Möglichkeit zur Synchronisation von Prozessen bildet der gegenseitige Ausschluß, durch den sichergestellt wird, daß zu einem Zeitpunkt nur eine Instanz das Privileg besitzt, Daten zu ändern. Dieses Konzept wird hier näher betrachtet. Alternativ bietet sich eine optimistische Strategie mit nachträglicher Behebung von Inkonsistenzen an, was jedoch zur zeitweiligen Existenz inkonsistenter Zustände führen kann.

Für die Implementierung des gegenseitigen Ausschlusses, der einen Teil der Konsistenzstrategien bildet, gilt was in [86, 87] für Konsistenzstrategien im allgemeinen festgestellt wird: eine a priori Festlegung eines Mechanismus kann niemals in der Lage sein, das Spektrum der möglichen Anforderungen befriedigend abzudecken. Aus diesem Grund wird hier ein Dokumentmuster vorgestellt, das die Realisierung des gegenseitigen Ausschlusses an die Dokumentmethoden der Dokumente delegiert. Das Dokumentmuster zum gegenseitigen Ausschluß verfügt über folgende privaten Operationen:

**private.getLock** Die private.getLock-Operation dient dem Betreten eines kritischen Abschnitts. Der Aufrufer übergibt den Namen des Abschnitts, den er betreten möchte, als Argument. Die Operation kehrt zurück, wenn sie den kritischen Abschnitt betreten hat. Das Resultat der Operationsausführung ist eine Referenz, die innerhalb des Dokuments eindeutig ist und den Aufrufer sowie den kritischen Abschnitt identifiziert.

**private.releaseLock** Die private.releaseLock-Operation dient dem Verlassen eines kritischen Abschnitts. Der Aufrufer gibt die von der private.getLock-Operation zurückgelieferte Referenz zur Identifikation des kritischen Abschnitts an, den er verlassen möchte.

Dokumentmethoden, die einen kritischen Abschnitt betreten wollen, rufen die private.getLock-Operation mit dem Namen des kritischen Abschnitts auf. Sie erhalten als Ergebnis eine Referenz, die sowohl den kritischen Abschnitt als auch den Inhaber des Locks identifiziert. Nach dem Verlassen des kritischen Abschnitts rufen sie die private.releaseLock-Operation mit der beim Betreten des kritischen Abschnitts erhaltenen Referenz auf, um ihre Abwesenheit aus dem kritischen Abschnitt mitzuteilen.

Im Rahmen des vorgegebenen Dokumentmusters sind unterschiedliche dokumentspezifische Implementierung des gegenseitigen Ausschlusses möglich, so daß z. B. zentrale Ausschlußmechanismen, verteilte Ausschlußmechanismen, Token-basierte Ausschlußmechanismen oder mehrheitsbasierte Ausschlußmechanismen zum Einsatz kommen können. Diese erfordern aber i. a. die Definition zusätzlicher Operationen z. B. zur Festlegung der Menge der beteiligten Instanzen.

In den folgenden Abschnitten werden Dokumentmethoden für die genannten Operationen vorgestellt, die einen gegenseitigen Ausschluß mit Hilfe eines zentralen Ausschlußmechanismus realisieren.

#### 4.3.1.1. private.getLock

Die der private.getLock-Operation zugeordnete Methode prüft zunächst, ob ein entsprechender kritischer Abschnitt bereits existiert. Sollte das nicht der Fall sein, wird eine Variable sowie ein Semaphor für diesen kritischen Abschnitt erzeugt. Die Methode verwendet dazu die enter- und leave-Funktion aus der Gruppe der Dokument-Funktionen (siehe Abschnitt 3.3.5.2). Existiert der kritische Abschnitt bereits, sind keine weiteren vorbereitenden Maßnahmen notwendig. Die Methode verwendet ein lokales Semaphor zum Schutz der private.getLock-Operation gegen konkurrierende Ausführungen.

Zum Betreten des kritischen Abschnitts ruft die Methode die P-Funktion[4] des Semaphors auf, das dem kritischen Abschnitt zugeordnet ist. Nach der Rückkehr der P-Funktion erzeugt die Methode eine Referenz, die den Kontext des Aufrufs repräsentiert und liefert diese als Ergebnis der Operationsausführung zurück.

#### 4.3.1.2. private.releaseLock

Die der private.releaseLock-Operation zugeordnete Methode prüft zunächst, ob das Semaphor belegt ist und ob die als Argument übergebene Referenz der beim Betreten des Semaphors erzeugten Referenz entspricht. Ist dies der Fall ruft die Methode die V-Funktion des Semaphors auf, das dem kritischen Abschnitt zugeordnet ist. Ist dies nicht der Fall, wird die Ausführung der Methode beendet ohne den kritischen Abschnitt freizugeben. Diese Überprüfung dient der Identifizierung des Initiators des Operationsaufrufs und hilft, eine versehentliche oder mißbräuchliche Ausführung der private.getLock-Operation zu verhindern.

### 4.3.2. Verteilte Speicherung von Baumstrukturen

Die in Kapitel 3.3.3 beschriebenen systemweit eindeutigen Dokumentnamen sind ortsabhängig, da sie die Adresse des Ausführungs-Server enthalten, auf dem das Dokument, bzw. das entsprechende Metadokument, gespeichert ist. Dokumentnamen verlieren daher bei einem Umzug des Dokuments auf einen anderen Ausführungs-Server ihre Gültigkeit. Alle Systeme, die die genannten Dokumentnamen verwenden, müssen entsprechend nachgeführt werden. Dieser Aufwand läßt sich durch die Verwendung logischer Dokumentnamen, die auf die entsprechenden systemweit eindeutigen ortsabhängigen Dokumentnamen abgebildet werden, vermeiden. Dazu ist jedoch die Einführung eines Namensdienstes notwendig, der die Abbildung zwischen logischen und ortsabhängigen Dokumentnamen vornimmt. In Indigo wird zur Realisierung eines solchen Namensdienstes ein Dokument verwendet, das diese Informationen verteilt speichert und auf Anfrage einen logischen Namen auf einen ortsabhängigen Dokumentnamen abbildet.

Die von diesem Dokument unterstützten logischen Namen sind hierarchisch aufgebaut. Sie bestehen aus Teilnamen, die durch spezielle Begrenzungszeichen voneinan-

[4] Die P-Funktion wird i. a. als P-Operation bezeichnet, die Bezeichnungen P-Funktion bzw. V-Funktion wurden hier gewählt, um eine Verwechslung mit den orthogonalen Operationen zu vermeiden.

der getrennt sind, analog zu den Namen im Domain Name System (DNS) (vgl. [59]) oder in einer Management Information Base (MIB) (vgl. [10]). Die Menge dieser Namen bildet zusammen mit einer virtuellen Wurzel einen Baum, wobei die Knoten mit den Teilnamen bezeichnet sind. Jeder Knoten kann einen Verweis auf einen systemweit eindeutigen ortsabhängigen Dokumentnamen besitzen.

Im folgenden wird ein Dokument vorgestellt, das den Baum der Namen verteilt speichert. Das Dokument besteht aus einer Menge von *Teildokumenten*, die sich gegenseitig bekannt sind. Jedes Teildokument besitzt zu diesem Zweck eine Liste aller anderen Teildokumente. Die Teildokumente speichern gemeinsam den Baum der existierenden Namen, indem die Menge der Blätter des Baumes partitioniert wird und die so erhaltenen Teilmengen in jeweils einem Teildokument gespeichert werden. Außerdem speichern die Teildokumente den vollständigen Pfad der in ihnen gespeicherten Blätter bis zur Wurzel des Baumes. Ein Teil des Baumes wird also repliziert in verschiedenen Dokumenten gespeichert. Die Liste der Teildokumente enthält dabei Informationen über die Teilbäume, die ausschließlich in einem Dokument gespeichert sind. Die Speicherung des gesamten Pfades bis zu den Blättern macht die Auflösung von Namen auch dann noch möglich, wenn Teildokumente ausfallen.

Um die Konsistenz beim Einfügen neuer Daten zu gewährleisten, verwendet das Dokument zur Speicherung von Bäumen ein Dokument zum gegenseitigen Ausschluß aus kritischen Abschnitten, wie es in Abschnitt 4.3.1 beschrieben wurde. Jedes Teildokument speichert den ortsabhängigen Namen des Dokuments zum gegenseitigen Ausschluß.

Das Dokumentmuster der Teildokumente verfügt über folgende private Operationen:

**private.resolve** Diese Operation dient der Abbildung eines logischen Namens auf einen systemweit eindeutigen ortsabhängigen Dokumentnamen.

**private.addNode** Diese Operation dient dem Einfügen eines neuen logischen Namens beziehungsweise der Änderung des ortsabhängigen Namens, auf den ein existierenden logischer Name verweist.

**private.lockedAddNode** Diese Operation dient ebenfalls dem Einfügen oder Ändern eines logischen Namens. Im Gegensatz zur private.addNode-Operation ist die Teildokumentliste während der Ausführung dieser Operation gegen Änderungen durch andere Operationen geschützt.

**private.addDocument** Diese Operation dient der Erweiterung der Teildokumentliste um neue Teildokumente oder der Bekanntgabe einer Änderung des Zuständigkeitsbereichs existierender Teildokumente. Sie akzeptiert eine Liste von Teildokumenten sowie der Bäume, die von diesen Teildokumenten verwaltet werden, als Argumente.

Eine ausführlichere Beschreibung der Methoden findet sich in den folgenden Abschnitten.

#### 4.3.2.1. private.resolve

Die Auflösung eines Namens wird durch den Aufruf der private.resolve-Operation an einem der Teildokumente initiiert. Wenn dieses Teildokument den entsprechenden Knoten selbst speichert, gibt es den ortsabhängigen Dokumentnamen als Ergebnis zurück. Speichert es den Namen nicht selbst, kann es aufgrund der Liste aller Teildokumente dasjenige Teildokument ermitteln, das den Namen speichert, und nimmt seinerseits die Auflösung über den Aufruf der private.resolve-Operation an diesem Dokument vor. Dieses Ergebnis liefert es dann an den Aufrufer zurück. Aufgrund der Kenntnis aller Teildokumente benötigt die Durchführung der Namensauflösung nur zwei Schritte.

#### 4.3.2.2. private.addNode

Um einen neuen Knoten einzufügen, wird eines der Teildokumente kontaktiert. Dieses Dokument sperrt mittels des Dokuments zum gegenseitigen Ausschluß die Teildokumentlisten gegen Veränderung. Dann ermittelt es anhand seiner Teildokumentliste das Dokument, das für den neuen Knoten zuständig ist und ruft die Methode die private.lockedAddNode-Operation dieses Dokuments auf. Findet sich kein für den Knoten zuständiges Teildokument, ruft es seine eigene private.lockedAddNode-Operation auf. Anschließend hebt es die Schreibsperre für die Teildokumentlisten wieder auf und beendet die Operationsausführung.

#### 4.3.2.3. private.lockedAddNode

Die der private.lockedAddNode-Operation zugeordnete Methode fügt einen Knoten in den Dokumentinhalt des Dokuments ein, an dem sie aufgerufen wurde. Da in Indigo ein Dokument vollständig in einem Ausführungs-Server gespeichert wird und die Ressourcen der Ausführungs-Server endlich sind, kann dabei der Fall eintreten, daß die Menge der in dem Teildokument gespeicherten Knoten zu groß für die Speicherung auf dem Ausführungs-Server wird. In diesem Fall muß zur Speicherung des Knotens ein neues Teildokument auf einem anderen Ausführungs-Server erstellt werden. Dies geschieht, indem die Methode folgende Schritte ausführt:

1. Sie erzeugt auf einem anderen Ausführungs-Server ein neues Teildokument und kopiert einen Teil der im lokalen Dokument gespeicherten Knoten in das neu erzeugte Dokument.

2. Sie fügt das neu erzeugte Teildokument in die Teildokumentliste ihres Dokuments ein.

3. Sie ruft an allen anderen Teildokumenten die private.addDocument-Operation auf, mit der das neue Teildokument und die Änderung des Zuständigkeitsbereichs ihres Dokuments bekannt gemacht werden.

4. Sie löscht alle Daten, die in das Teildokument kopiert wurden, aus dem Inhalt ihres Dokuments.

Nach der Ausführung dieser Schritte beendet sich die Dokumentmethode. Ein Aufrufer gibt schließlich die Teildokumente wieder zum Schreiben frei.

#### 4.3.2.4. private.addDocument

Die der private.addDocument-Operation zugeordnete Methode fügt den systemweit eindeutigen ortsabhängigen Namen eines neuen Teildokuments in die Liste der Teildokumente ihres Dokuments ein. Hierbei sind keine weiteren Maßnahmen zum Schutz gegen konkurrierende Zugriffe notwendig, da diese Operation nur während der Ausführung einer private.lockedAddNode-Operation, die gegen konkurrierenden Zugriff geschützt ist, initiiert wird.

#### 4.3.2.5. Verwandte Arbeiten

Das hier vorgestellte Dokumentmuster zur verteilten Datenhaltung folgt dem Konzept der fragmentierten Objekte. Fragmentierte Objekte sind verteilte Objekte, z. B. replizierte oder partitionierte Objekte, deren Verteilung für den Benutzer des Objekts transparent ist. Der Anwender greift über ein Zugriffsobjekt auf den gesamten Datenbestand zu. Der Begriff des fragmentierten Objekts im hier genannten Sinn ist in [85, 52] definiert. Fragmentierte Objekte werden z. B. in MIS/0 [45] und in Globe [86, 87], einer weitverteilten, skalierbaren Infrastruktur für Dokumente, benutzt.

In [52] wird ein ähnliches, auf dem Konzept der fragmentierten Objekte basierendes Dokument zur verteilten Speicherung eines Baumes beschrieben, das ebenfalls der Realisierung eines Namensdienstes dient. Der dort verwendete Multicast-Kanal wird hier durch die Liste der Teildokumente und den expliziten Aufruf von Operationen an allen Dokumenten dieser Liste realisiert.

Der Ansatz, Konsistenzbedingungen durch dokumentspezifische Methoden zu implementieren findet sich ebenfalls in Globe. Im Gegensatz zu Indigo ist die in Globe vorgenommene Strukturierung der Objekte in die Teilobjekte Kontrolle, Semantik, Replikation und Kommunikation statisch. In Indigo werden durch die Dokumentmuster zwar bestimmte Funktionsblöcke vorgeschlagen, diese sind aber nicht verbindlich. Neue Anforderungen können durch eine entsprechende, individuelle Definition der Operationen realisiert werden.

Auch das in Abschnitt 4.3.2 vorgestellte Dokument dient als Beispiel für die Möglichkeiten von Indigo, weitere Partitionierungsmuster und unterschiedliche Optimierungen sind denkbar. Hier soll vor allem gezeigt werden, daß eine verteilungstransparente verteilte Datenhaltung vollständig durch Dokumente in Indigo realisiert werden kann. Von dieser Möglichkeit wird in Abschnitt 5.4.1 bei der Erstellung eines Kataloges Gebrauch gemacht. Mit dem Entwurf des Dokuments zur verteilten Speicherung von Baumstrukturen ist nicht die Ersetzung oder Neuschaffung existierender Namensdienste angestrebt, vielmehr werden die Möglichkeiten zur Erfüllung spezifischer Anforderungen und die dafür notwendigen Mechanismen dokumentiert. Als Nebeneffekt steht ein logischer Namensdienst in der Infrastruktur zur Verfügung.

## 4.4. Weitere Dokumentmuster

In diesem Abschnitt werden Dokumentmuster vorgestellt, die weitere Anforderungen Digitaler Bibliotheken erfüllen bzw. häufig benötigte Mechanismen implementieren.

### 4.4.1. Aggregation und Assoziation von Dokumenten

In Indigo lassen sich Dokumente durch ihren Namen systemweit eindeutig referenzieren. Diese Referenzen können als Inhalt eines Dokuments auftreten und so Dokumente in Relation zueinander setzten. Obwohl Relationen zwischen Dokumenten vielfältige Bedeutungen besitzen können herrschen zwei Arten von Relationen vor: die Assoziation von Dokumenten und die Aggregation von Dokumenten. Zwischen Aggregation und Assoziation wird hier gemäß [82] unterschieden. Assoziation stellt eine, hier nicht näher beschriebene, Beziehung zwischen den Dokumenten dar, während ein aggregiertes Dokument aus seinen Teilen besteht. Der Inhalt eines aggregierten Dokuments setzt sich also aus dem Inhalt der referenzierten Dokumente, der Teildokumente, zusammen, und ist unvollständig wenn ein Teildokument unvollständig ist oder fehlt. Beispiele für Assoziation von Dokumenten sind annotierte Dokumente, Kollektionen von Dokumenten und Dokumente mit einer Hyperlink-Struktur. Beispiele für eine Aggregation sind HTML-Seiten mit integrierten Abbildungen. Mit Hilfe der Aggregation bzw. Assoziation läßt sich eine Vielzahl unterschiedlicher Anwendungen realisieren, z. B. die Erstellung von Sammlungen, die Erstellung von Annotationen oder die Erstellung eines Dokuments mit Hyperlink-Struktur.

Der Unterschied zwischen Assoziation und Aggregation manifestiert sich im Umgang mit den referenzierten Dokumenten, also in den Dokumentmethoden des referenzierenden Dokuments. Diese interpretieren Referenzen als Aggregation oder Assoziation und führen die entsprechenden Aktionen durch. Die Darstellung der Referenzen ist frei wählbar. Sie kann z. B. aus einer einfachen Liste der Dokumentnamen bestehen, in diesem Fall müssen die Dokumentmethoden Informationen über die Art der Referenz, z. B. Aggregation oder Assoziation, besitzen. Ein Alternative wäre die Verwendung qualifizierter Referenzen, in denen der Charakter der Referenz bereits kodiert ist, z. B. XLink-Verweise [99].

Die Present- und Describe-Operationen aggregierter Dokumente lassen sich durch die Delegation der Operationsausführung an die Dokumentmethoden der Teildokumente realisieren. Dadurch kann die Assoziation bzw. Aggregation von Dokumenten ohne eine Modifikation der Teildokumente vorgenommen werden.

**Present** Die Present-Operation eines aggregierten Dokuments muß alle Bestandteile des Dokuments präsentieren. Dies kann z. B. durch die parallele Initiierung der Present-Operationen der Teildokumente geschehen.

**Describe** Die Describe-Operation des aggregierten Dokuments muß eine Beschreibung liefern, die aus den Beschreibungen der Teildokumente zusammengesetzt ist. Dies kann durch eine Initiierung der einzelnen Describe-Operationen und eine anschließende Integration der Ergebnisse erreicht werden.

Die Dokumentmethoden zur Durchführung Present- und Describe-Operationen aggregierter Dokumente sind also nicht von den Typen der enthaltenen Dokumente abhängig sondern lediglich von der Syntax der Referenzen in dem referenzierenden Dokument.

Andere Verknüpfungssemantiken lassen sich ebenfalls realisieren. Dazu muß das Verhalten der Dokumentmethoden des referenzierenden Dokuments modifiziert werden. So könnte z. B. ein Dokument erstellt werden, das eine Sammlung der referenzierten Dokumente repräsentiert. Die Present- und Describe-Operationen dieses Dokuments hätten dann folgende Semantik:

**Present** Die Present-Operation eines Dokuments zur Sammlung existierender Dokumente präsentiert die Referenzen in Form einer Liste. Bei Auswahl eines Listenelements wird die Ausführung der Present-Operation des ausgewählten Dokuments initiiert.

**Describe** Die Describe-Operation eines Dokuments zur Sammlung liefert eine Beschreibung, die den Sammlungscharakter widerspiegelt, indem sie die Describe-Operationen der referenzierten Dokumente initiiert und deren Resultate zu einer Menge von Beschreibungen zusammenfaßt.

Die Dokumentmethoden des Dokuments zur Sammlung sind ebenfalls generisch, d. h. sie sind nicht vom Typ der in der Sammlung enthaltenen Dokumente abhängig.

Die Orthogonalität der Operationen in Indigo unterstützt also nicht nur die rechnergesteuerte Ausführung von Operationen und die Bearbeitung neuer Dokumenttypen durch existierende Dienste, sondern ermöglicht auch die Erstellung generischer Dokumente zur Aggregation und Assoziation von Dokumenten. Ein Beispiel für die Realisierung eines Dokuments zur Aggregation findet sich in Abschnitt 6.3.4.

### 4.4.2. Zugriffsgeschütze Dokumente

Dokumente können wertvolle Inhalte transportieren, die nur autorisierten Benutzern zugänglich sein sollen und nicht-autorisierten Benutzern, wenn überhaupt, nur in wertreduzierter Form zur Verfügung stehen sollen. So gibt es eine Reihe von Digitalisierungsprojekten, die wertvolle Vorlagen unter großem Ressourceneinsatz in hoher Qualität digitalisieren. Diese hochwertigen digitalen Dokumente stehen zwar nur wenigen autorisierten Nutzern zur Verfügung, eine wertverminderte bzw. kopiergeschützte Version ist dagegen öffentlich zugänglich. Beispiele dafür sind das Digitalisierungsprojekt der Vatikan-Bibliothek[5] oder das Digitalisierungsprojekt der Lutherhalle Wittenberg[6]. In beiden Fällen werden die Dokumente durch niedrige Auflösung oder digitale Wasserzeichen (vgl. [56]) soweit wertvermindert, daß sie nur für den privaten Gebrauch sinnvoll nutzbar sind.

In diesem Abschnitt werden zwei Dokumentmuster beschrieben, die die Autorisierung von Zugriffen und die wertverminderte Präsentation in Indigo realisieren. Darüber

[5] Beispiele markierter Dokumente unter: `<url:http://www-4.ibm.com/software/is/dig-lib/vatican/manuscript.html>`

[6] Beispiele markierter Dokumente unter: `<url:http://www-4.ibm.com/software/is/dig-lib/info/lutherpic.html>`

hinaus wird ein Dokument vorgestellt, das einen sicheren Transport über unsichere anwenderseitige Ausführungs-Server hinweg, durchführen kann.

#### 4.4.2.1. Autorisierung der Operationsausführung

Wie in Kapitel 3 beschrieben, kann ein Ausführungs-Server eine Authentifizierung des Initiators einer Operation durchführen. Als Resultat der Authentifizierung steht entweder das Zertifikat des Initiators oder die Information, daß der Initiator anonym ist, zur Verfügung. Mit Hilfe dieser Informationen kann ein Dokument eine dokumentspezifische Autorisierung des Initiators einer Operation durchführen. Die Authentifizierung erfolgt anhand des vom Ausführungs-Server bereitgestellten Zertifikats des Initiators und anhand einer im Dokument gespeicherten Liste vertrauenswürdiger Zertifizierungsstellen.

Das Dokumentmuster zur Autorisierung von Operationsausführungen verfügt über folgende Methoden:

**private.authorize** Diese Operation erwartet einen Operationsnamen als Argument. Sie nutzt die get_initiator-Funktion der Funktionsgruppe Server (siehe Abschnitt 3.3.5.1) zur Ermittlung der Identität des Initiators und liefert Informationen darüber zurück, ob der Initiator dazu autorisiert ist, die genannte Operation auszuführen. Zu diesem Zweck verwendet sie im Dokument gespeicherte Informationen über berechtigte Subjekte und über vertrauenswürdige Zertifizierungsinstanzen.

**private.addSubject** Die private.addSubject-Operation erwartet einen Operationsnamen und einen Initiator als Argumente und autorisiert den genannten Initiator, in Zukunft die Operation, deren Name als Argument übergebenen wurde, auszuführen. Die Ausführung der private.addSubject-Operation beinhaltet die Ausführung der private.authorize-Operation zur Autorisierung der Operationsausführung.

**private.removeSubject** Die private.removeSubject-Operation erwartet ebenfalls einen Operationsnamen und einen Initiator als Argument und widerruft die Autorisierung des genannten Initiators zur zukünftigen Ausführung der Operation, deren Name als Argument übergeben wurde. Die Ausführung der private.removeSubject-Operation beinhaltet die Ausführung der private.authorize-Operation zur Autorisierung der Operationsausführung.

**private.addCertInstance** Die private.addCertInstance-Operation erwartet den Namen einer Zertifizierungsstelle als Argument und fügt diese in die Menge der Zertifizierungsstellen ein, deren Zertifikate akzeptiert werden. Die Ausführung der private.addCertInstance-Operation beinhaltet die Ausführung der private.authorize-Operation zur Autorisierung der Operationsausführung.

**private.removeCertInstance** Die private.addCertInstance-Operation erwartet den Namen einer Zertifizierungsstelle als Argument und entfernt diese aus der Menge der akzeptierten Zertifizierungsstellen. Auch die Ausführung

der private.removeCertInstance-Operation wird durch die Ausführung der private.authorize-Operation autorisiert.

Die Informationen, auf die die genannten Methoden zugreifen, werden als Teil des Dokumentinhalts gespeichert. Bei der Konzeption weiterer Methoden des Dokuments, ist darauf entsprechend Rücksicht zu nehmen. Dies kann z. B. durch eine Reimplementierung der Dokument-Funktionen open, close, read und write (siehe Abschnitt 3.3.5.2) in Form entsprechender privater Operationen zum Zugriff auf den Dokumentinhalt geschehen. Diese Operationen können dann von weiteren Operationen des Dokuments genutzt werden, um transparent auf den Inhalt des Dokuments zuzugreifen.

Die Autorisierung wird korrekt ausgeführt, wenn das Dokument nur über vertrauenswürdige Server transportiert wurde und sich auf einem vertrauenswürdigen Server befindet. Dadurch, daß die Methoden vom Dokumentautor mit einer Prüfsumme versehen wurden, kann davon ausgegangen werden, daß auf einem sicheren Server die korrekten Methoden ausgeführt werden.

Die Verlagerung der Autorisierung in die Dokumentmethoden ermöglicht eine sehr flexible Autorisierung, die nicht auf die Verwendung eines Mechanismus, z. B. auf Zugriffskontrollisten beschränkt ist. Vielmehr können zeitbasierte Verfahren oder Mengenkontingente realisiert werden.

In der FEDORA-Architektur (vgl. [70]) werden ebenfalls dokumentspezifische Programmfunktionen, die Disseminatoren, zur Realisierung individueller Sicherheitsstrategien verwendet (vgl. [71]).

#### 4.4.2.2. Transport wertverminderter Inhalte

Bei wertvollen Dokumenten möchte man häufig Anwendern, die nicht zum Zugriff auf den vollständigen Inhalt berechtigt sind, die Präsentation einer wertverminderten Version des Dokuments erlauben. Um den nicht-autorisierten Zugriff auf den Dokumentinhalt zu verhindern, darf das Dokument dabei nicht vollständig auf den unsicheren Ausführungs-Server eines Anwenders transportiert werden. Eine Dokumentmethode, die auf einem unsicheren bzw. manipulierten Ausführungs-Server ausgeführt wird, kann weder mit Sicherheit feststellen, daß sie sich in dieser Situation befindet noch zuverlässig Prüfungen, wie z. B. die Verifikation einer digitalen Signatur, vornehmen (vgl. [12, 28]).

Aus diesem Grund muß vermieden werden, daß wertvolle Dokumentinhalte auf unsichere Server transportiert werden. Da sich ein Dokument nach seiner Integration in Indigo auf einem vertrauenswürdigen speicherseitigen Ausführungs-Server befindet, kann es vor einem Transport davon ausgehen, daß sämtliche Operationen korrekt ausgeführt werden. Dies erlaubt es den Dokumentmethoden, vor einem Transport, der immer von einer Dokumentmethode initiiert wird, mittels der is_trusted-Funktion (siehe Abschnitt 3.3.5.3) die Vertrauenswürdigkeit des Ausführungs-Servers zu prüfen, auf den das Dokument transportiert werden soll. Das Dokumentmuster zur Präsentation wertverminderter Versionen enthält folgende Operationen:

**private.trustedPresent** Die private.trustedPresent-Operation erhält die Adresse ei-

nes vertrauenswürdigen Ausführungs-Servers als Eingabe und führt eine uneingeschränkte Präsentation des Dokuments durch.

**private.untrustedPresent** Die private.untrustedPresent-Operation erhält die Adresse eines nicht-vertrauenswürdigen Ausführungs-Servers als Eingabe und führt eine wertverminderte Präsentation des Dokuments durch.

**private.trustedCopy** Die private.trustedCopy-Operation erhält die Adresse eines vertrauenswürdigen Ausführungs-Servers als Eingabe und transportiert eine vollständige Kopie des Dokuments auf diesen Ausführungs-Server.

**private.untrustedCopy** Die private.untrustedCopy-Operation erhält die Adresse eines nicht-vertrauenswürdigen Ausführungs-Servers als Eingabe und transportiert eine wertverminderte Version des Dokuments auf diesen Ausführungs-Server.

**Present** Die Present-Operation prüft zunächst, ob der als Argument übergebene Ausführungs-Server vertrauenswürdig ist. Abhängig von dem Ergebnis dieser Prüfung initiiert sie dann die Operation private.trustedPresent-Operation oder die private.untrustedPresent-Operation.

**Copy** Die Copy-Operation prüft ebenfalls die Vertrauenswürdigkeit des als Argument übergebenen Ausführungs-Servers und initiiert in Abhängigkeit vom Ergebnis die Ausführung der private.trustedCopy-Operation oder der private.untrustedCopy-Operation auf.

**Move** Die Move-Operation verhält sich analog zur Copy-Operation, entfernt aber, nachdem die Ausführung der private.trustedCopy-Operation bzw. der private.untrustedCopy-Operation beendet ist, das Dokument, an dem die Move-Operation initiiert wurde, vom Ausführungs-Server.

Da die Durchführung eines Dokumenttransports immer unter der Kontrolle einer Dokumentmethode erfolgt, kann ein Dokument bei Bedarf sicherstellen, nie vollständig zu einem unsicheren Server transportiert zu werden. Eine wertverminderte Präsentation oder Kopie eines Dokuments kann analog zur Online-Präsentation (siehe Abschnitt 4.2.2) realisiert werden, indem über eine Verbindung zwischen einer anwenderseitigen und einer speicherseitigen Dokumentmethode der entsprechend modifizierte Dokumentinhalt, z. B. mit reduzierter Auflösung, reduzierter Bitrate oder einem Wasserzeichen versehen, gesendet wird.

#### 4.4.2.3. Ende-zu-Ende-Verschlüsselung

Es gibt eine Möglichkeit, wertvolle Dokumente auch über unsichere Server hinweg vertraulich zwischen zwei Parteien auszutauschen. Dies wird dadurch erreicht, daß das Dokument beim Verlassen eines vertrauenswürdigen Ausführungs-Servers den Dokumentinhalt verschlüsselt, z. B. mit dem öffentlichen Schlüssel des Adressaten oder einem vereinbarten symmetrischen Schlüssel. Gleichzeitig wird vermerkt, daß der Dokumentinhalt verschlüsselt ist, so daß sich die Dokumentmethoden entsprechend

verhalten können. Damit ist der vertrauliche Transport des Dokumentinhalts auf den Ausführungs-Server des Adressaten möglich.

Zur Nutzung des Dokuments gibt der Adressat den notwendigen Schlüssel an, woraufhin die entsprechende Dokumentmethode, z. B. die Methode zur Präsentation, den Dokumentinhalt entschlüsseln und präsentieren kann. Die Durchführung der Entschlüsselung innerhalb der Dokumentmethode setzt voraus, daß der Adressat dem lokalen Ausführungs-Server vertraut. Alternativ zur Entschlüsselung innerhalb des Ausführungs-Servers bietet sich die Erstellung einer privaten Operation, z. B. private.EmitBase64, an, die den Inhalt des Dokuments in einem vereinbarten Format ausgibt. Die Entschlüsselung kann dann durch den Empfänger in einer Laufzeitumgebung geschehen, der er vollständig vertraut.

## 4.5. Fazit

In diesem Kapitel wurde gezeigt, wie sich Dokumentmuster definieren lassen, die als Basis für die Realisierung komplexerer Dokumente verwendet werden können. Ferner wurden Dokumentmuster definiert, die sowohl elementare Funktionen auf Dokumenten bereitstellen, als auch weitergehende Mechanismen implementieren.

Eine wichtige Operation auf Dokumenten ist die Präsentation. Anhand der Beispiele wurde gezeigt, daß die Infrastruktur die Präsentation einer Vielzahl von Dokumenttypen unterstützt. Sie bietet mit ihrer Flexibilität die Möglichkeit zur optimalen Verteilung der Dokumentmethoden und damit zu einer effizienten Bandbreitennutzung.

Komplexere Dokumentmuster lassen sich ebenfalls transparent für den Anwender realisieren. So läßt sich das Konzept der fragmentierten Objekte, das die transparente Verteilung von Daten erlaubt, in der Infrastruktur umsetzten. Die entsprechenden Dokumente können durch einfache Kombination der beschriebenen Operationen definiert werden.

# 5. Dienste in der Digitalen Bibliothek

In diesem Kapitel wird die dezentrale Implementierung von Diensten beschrieben, deren Durchführung, im Gegensatz zu Operationen auf Dokumenten, die Möglichkeit zum Zugriff auf alle Dokumente voraussetzt. Zu diesen Diensten gehören u. a. die Suche in Dokumenten sowie die Erstellung von Katalogen und Indizes.

## 5.1. Dienste

### 5.1.1. Ziele und Randbedingungen

Bei den bisherigen Betrachtungen wurden jeweils Operationen behandelt, die sich auf ein Dokument beziehen. Neben der Möglichkeit zur Ausführung von Operationen auf einzelnen Dokumenten zeichnen sich Digitale Bibliotheken durch die Bereitstellung von Diensten aus, die Berechnungen durchführen, die alle Dokumente der Bibliothek einbeziehen. Ein Beispiel für einen solchen Dienst ist die Bereitstellung eines Katalogs, der dem Existenz- oder Standortnachweis digitaler Dokumente dienen kann. Weitere Beispiele für Dienste sind die Suche nach Dokumenten sowie die Erstellung eines Index aller Dokumente. Neben diesen auch in traditionellen Bibliotheken vorhandenen Diensten ermöglicht die Digitale Bibliothek die Realisierung neuer Dienste, wie z. B. benutzerspezifischer Suchdienste oder Profildienste.

Allen Diensten ist die Notwendigkeit gemeinsam, auf alle Dokumente der Infrastruktur zugreifen zu können. Dies wird durch zwei Faktoren erschwert: durch die dezentrale Einführung neuer Dokumente und durch die dezentrale Einführung neuer Ausführungs-Server. Diese Dynamik macht die Möglichkeit zur einmaligen, statischen Festlegung eines Katalogs aller Server und aller Dokumente zunichte. Vielmehr müssen die verfügbaren Server und Dokumente dynamisch ermittelt werden. Dabei gilt aber wiederum, daß keine zentrale Instanz verwendet werden darf, die z. B. einen Katalog aller Server oder aller Dokumente umfaßt. Neben dem Problem der Skalierbarkeit[1] ergibt sich bei einer solchen zentralen Instanz vor allem das Problem der Verfügbarkeit, da sie von allen Diensten verwendet wird und bei ihrem Ausfall alle Dienste betroffen wären.

Daher muß ein Weg gefunden werden, mittels eines dezentralen Verfahrens alle Server und damit alle Dokumente lokalisieren zu können. Dieses Verfahren soll möglichst

[1] Natürlich stellt sich die Frage, ob dieses Problem bei einem reinen Server-Katalog auftauchen würde. Selbst wenn das System die heutige Zahl an HTTP-Servern erreichen sollte, ließe sich ein solcher Katalog verwalten. Für die Dokumentliste gilt dies nicht mehr, selbst große Indexierungs-Server wie AltaVista indexieren nur einen Teil der Dokumente der einzelnen Sites bzw. des WWW. Neben der reinen Speicherkapazität spielt hier auch die verfügbare Rechenleistung eine Rolle.

robust gegenüber Änderungen der Server-Topologie und der Zahl der Dokumente sein. Gleichzeitig wird aus Effizienzgründen gefordert, daß beim Zugriff der einzelnen Dienste auf die Dokumentdaten möglichst viele Berechnungen lokal erfolgen.[2]

### 5.1.2. Lösungsstrategie

Dienste werden in Indigo gemäß der in Abschnitt 2.3.1.2 vorgestellten Architektur, die zwischen dokumentspezifischen und generischen Berechnungen zur Erbringung des Dienstes unterscheidet, realisiert. Die dokumentspezifischen Berechnungen werden durch die einzelnen Dokumente in Form orthogonaler Operationen bereitgestellt. Die generischen Berechnungen werden durch Komponenten realisiert, die auf die orthogonalen Operationen zugreifen und das Ergebnis ihrer Berechnungen an der Dienstschnittstelle zur Verfügung stellen. Diese Komponenten werden im folgenden als *diensterbringende Komponenten* bezeichnet. Aufgrund der genannten Anforderungen sollen diensterbringende Komponenten dezentral realisiert werden und Operationen an Dokumenten möglichst lokal durchführen.

Der für Indigo entwickelte Ansatz besteht in der Verwendung von Dokumenten zur Implementierung diensterbringender Komponenten. Betrachtet man die Dokumente in Indigo genauer, fällt auf, daß sie sich, ähnlich wie mobile Agenten, zur Verteilung von Berechnungen nutzen lassen. Die Zustandsinformation kann innerhalb des Dokumentinhalts gespeichert werden, der Programmtext wird in Form von Operationen bzw. in Form der den Operationen zugeordneten Dokumentmethoden gespeichert. Ein solches Dokument gleicht einem mobilen Agenten insofern, als es einen internen Zustand und Programmcode besitzt, der zwischen Rechnern übertragen und ausgeführt werden kann. Der Unterschied zum Agenten besteht darin, daß ein Dokument sich nicht wirklich fortbewegt, sondern nur Kopien seiner selbst auf einem entfernten Rechner erzeugt, während es in seiner ursprünglichen Umgebung verweilt. Im Gegensatz dazu verlassen Agenten ihre ursprüngliche Umgebung, wenn sie sich auf einen neuen Rechner verlagern. [3]

Mit der Verwendung von mobilen Dokumenten zur Implementierung von Diensten lassen sich zwei Anforderungen erfüllen. Zum einen können Berechnungen dadurch lokal ausgeführt werden, indem ein diensterbringendes Dokument zu den jeweiligen speicherseitigen Ausführungs-Servern transportiert wird. Zum anderen ist damit eine Möglichkeit zur flexiblen Erweiterung um neue Dienste gegeben. Neue Dienste werden durch die Erstellung mobiler Dokumente zur Diensterbringung realisiert. Diese Flexibilität wäre durch eine statische Integration der generischen Berechnungen zur Diensterbringung in die Server nicht zu erreichen.

Zu lösen bleibt das Problem, die diensterbringenden Dokumente auf alle Ausführungs-Server zu verteilen, ohne auf ein zentrales Verzeichnis angewiesen zu sein, und die lokalen Berechnungen, die für die Ausführung des Dienstes notwendig sind, auf diesen

[2] Diese Forderung führte z. B. zu der verteilten Architektur des WWW-Indexierungsdienstes Harvest (vgl. [9]), der Teile der Indexberechnung lokal in den HTTP-Server durchführt.

[3] Die Fortbewegung des Dokuments besteht also in einer Art entferntem fork-Aufruf. Sollte das Dokument sich selbst von dem Ursprungsrechner löschen, wäre die Semantik der Mobilität, bis auf die Identität der Dokumente, die gleiche wie bei Agenten. Diese Möglichkeit wurde auch in AspectIX erkannt (vgl. [37]).

Ausführungs-Servern genau einmal zu initiieren. Eine mehrfache Ausführung der lokalen Berechnungen ist unbedingt zu vermeiden. Sie wäre nicht nur ineffizient sondern würde auch bei nicht-idempotenten Berechnungen zu Fehlern führen.

Eine mögliche Lösung des Problems besteht in der Verwendung eines dezentralen Algorithmus zur Berechnung des Spannbaums des Server-Graphen in den Ausführungs-Servern und der Flutung des Spannbaums mit diensterbringenden Dokumenten. Weitere Möglichkeiten zur Lösung des Problems bestehen in der Verteilung der diensterbringenden Dokumente auf alle Ausführungs-Server und der Implementierung von Mechanismen zur Erkennung bereits besuchter Ausführungs-Server. Diese Alternativen zur Verteilung diensterbringender Dokumente in Indigo werden im folgenden Abschnitt diskutiert. Die Integration von dienstspezifischen Berechnungen und Berechnungen zur Verteilung der Dokumente wird in Abschnitt 5.3 behandelt.

## 5.2. Verteilung diensterbringender Dokumente

Innerhalb der Infrastruktur gibt es mehrere Möglichkeiten zur Berechnung des Spannbaums des Server-Graphen. Eine Möglichkeit besteht in der Berechnung des Spannbaums in den Ausführungs-Servern und der verteilten Speicherung des Resultats in den Ausführungs-Servern, Alternativ dazu kann die Spannbaumberechnung innerhalb eines mobilen Dokuments zum Zeitpunkt seiner Verteilung, also online, durchgeführt werden. Als dritte Möglichkeit bietet sich eine kooperative Berechnung des Spannbaums durch Dokumentmethoden und Ausführungs-Server an. Jede dieser Varianten besitzt spezifische Vor- und Nachteile. In den folgenden Abschnitten werden die drei Verfahren erläutert und die Auswahl eines Ansatzes motiviert.

### 5.2.1. Spannbaumberechnung in den Servern

Für die Berechnung des Spannbaums in den Ausführungs-Servern wird der Algorithmus von Chang [11] verwendet. Die Berechnung des Spannbaums erfolgt dezentral in den Ausführungs-Servern. Das Ergebnis der Berechnung wird dezentral in den Ausführungs-Servern gespeichert, indem jeder Server seinen Vorgänger- und seine Nachfolgerknoten speichert.[4] Das Speichern der Nachfolgerknoten erfordert eine Erweiterung des ursprünglichen Algorithmus um eine Rückmeldung der Nachfolger an die Vorgänger (vgl. [51, S. 498 f]). Der verwendete Algorithmus wird im folgenden beschrieben.

#### 5.2.1.1. Algorithmus von Chang

Bei dem Algorithmus von Chang handelt es sich um einen wellenbasierten Algorithmus (vgl. [93, S. 177 f]). Die Ausführung des Algorithmus wird durch einen ausgezeichneten Server angestoßen, indem dieser Wellennachrichten an alle Server in seiner Nachbarschaftsmenge sendet. Jeder Server arbeitet nach dem folgenden Prinzip:

[4] Der genannte Algorithmus führt keine Tiefensuche auf dem Server-Graphen durch, dafür läßt sich der Algorithmus von Awerbuch [2] oder die verbesserte Form von Cidon [14] einsetzen.

- Wenn eine Wellennachricht von einem Nachbarn eingeht, wird zunächst geprüft, ob bereits eine Wellennachricht eingetroffen ist.
- Ist dies der Fall, wird eine Nicht-Nachfolgernachricht an den Nachbarn gesendet, von dem die Wellennachricht empfangen wurde.
- Falls dies nicht zutrifft, wird der Nachbar, von dem die Wellennachricht empfangen wurde, als Vorgängerknoten des Spannbaums gespeichert und eine Nachfolgernachricht an ihn zurückgesendet. Ferner wird eine Wellennachricht an jeden Nachbarn, außer an den Vorgänger gesendet. Jeder Nachbar, von dem eine positive Nachfolgernachricht empfangen wird, wird dann in die Nachfolgerliste eingetragen.
- Die Berechnung endet, wenn der auslösende Server von allen Nachbarn Nachfolgernachrichten bzw. Nicht-Nachfolgernachrichten erhalten hat.

Der Algorithmus des Initiators ist in Programm 5.1 dargestellt, der Algorithmus der Nicht-Initiatoren ist in Programm 5.2 dargestellt.[5]

**Programm 5.1** start-Prozedur der Server-basierten Spannbaumberechnung

```
// Wird vom Initiator der Spannbaumberechnung ausgeführt.
proc start() :
  // Die Menge der Nachfolger im Baum initialisieren.
  successors := ∅
  // An alle Nachbarn durch Initiierung der continue-Prozedur
  // eine Wellennachricht senden. Als Argument wird die Adresse
  // des Initiators übergeben.
  handle_set := ∅
  for n ∈ neighbors do
      handle_set := handle_set ∪ {n.continue(serverAddress)}
  od
  // Ausführung fortsetzten, bis alle Nachbarn
  // eine Antwort zurückgeliefert haben.
  while handle_set do
        // Ergebnis des Aufrufs ermitteln.
        handle := wait_for_result(handle_set)
        handle_set := handle_set \ {handle}
        // Alle Nachbarn, die eine positive Quittung liefern
        // als Nachfolger im Spannbaum eintragen.
        if get_result(handle) then
          successors := successors ∪ {source(handle)}
        fi
  od
end
```

[5] Die Algorithmen sind in Pseudo-Code notiert. Für die Ausführung von Prozeduren auf entfernten Rechnern wird eine RPC-ähnliche Notation verwendet. Kommentare werden mit // eingeleitet.

Der von dem vorgestellten Algorithmus berechnete Spannbaum entspricht nicht zwingend dem Baum, den man durch Breitensuche vom auslösenden Server aus erhält. Vielmehr reflektiert er den Zustand der Kanäle zwischen den Servern, der zum Zeitpunkt der Ausführung besteht. Kanäle mit geringer Verzögerung werden gegenüber Kanälen mit hoher Verzögerung bevorzugt. Gestörte Kanäle werden ignoriert. Dies kann dazu führen, daß ein Nachbar im Graph kein direkter Nachfolger bzw. Vorgänger im Spannbaum ist.

**Programm 5.2** continue-Prozedur der Server-basierten Spannbaumberechnung

```
// Wird von Nicht-Initiatoren der Spannbaumberechnung ausgeführt.
// Als Argument wird die Adresses des Initiators der Berechnung
// übergeben.
proc continue(caller) :
  // Prüfen ob die Berechnung diesen Knoten bereits erreicht hat.
  if already_visited then
    // Falls ja, ist dieser Knoten kein Nachfolger des Aufrufers.
    return(false)
  fi
  already_visited := true
  // Die Menge der Nachfolger im Baum mit dem Aufrufer
  // initialisieren (Vorgänger und Nachfolger werden
  // in der gleichen Menge gespeichert).
  successors := {caller}
  // Die Wellennachricht an alle Nachbarn, außer dem Aufrufer,
  // weitergeben. Als Argument wird die Adresse dieses Servers
  // übergeben.
  handle_set := ∅
  for n ∈ neighbors \ {caller} do
      handle_set := handle_set ∪ {n.continue(serverAddress)}
  od
  // Ausführung fortsetzten, bis alle Aufrufe zurückkehren.
  while handle_set do
        handle := wait_for_result(handle_set)
        handle_set := handle_set \ {handle}
        // Alle Nachbarn, die eine positive Quittung liefern
        // als Nachfolger im Spannbaum eintragen.
        if get_result(handle) then
          successors := successors + source(handle)
        fi
  od
  // Dieser Knoten ist ein Nachfolger des Aufrufers.
  return(true)
end
```

#### 5.2.1.2. Nutzung des Server-berechneten Spannbaum

In diesem Abschnitt wird ein diensterbringendes Dokument vorgestellt, das den in den Ausführungs-Servern berechneten Spannbaum nutzt, um Operationen lokal auf allen Ausführungs-Servern durchzuführen. Die Ausführungs-Server stellen Dokumentmethoden dazu eine Liste aller Nachbarn im Spannbaum zur Verfügung. Die Dokumentmethoden können über entsprechende Schnittstellen der Laufzeitumgebung auf die Liste der Nachbarn im Baum zugreifen. Das diensterbringende Dokument verfügt über zwei private Operationen, private.Start und private.Continue.

**private.Start** Die private.Start-Operation wird aufgerufen, um die Verteilung des Dokuments zu initiieren. In ihr werden Kopien des eigenen Dokuments an alle Nachbarn im Spannbaum gesendet und an diesen die Ausführung der private.Continue-Operation initiiert. Wenn das Ergebnis von allen Dokumenten eingetroffen ist, beendet sich die private.Start-Operation und liefert das Gesamtergebnis zurück.

**private.Continue** Die private.Continue-Operation erhält als erstes Argument die Adresse des Ausführungs-Servers, von dem aus sie initiiert wurde. Die Operation versendet ihrerseits Kopien des eigenen Dokuments auf alle benachbarten Ausführungs-Server, außer an den initiierenden Ausführungs-Server, im Spannbaum und initiiert dort die Ausführung der private.Continue-Operation. Als Argument übergibt sie die Adresse des Ausführungs-Servers, auf dem sie sich befindet. Sie liefert ein Ergebnis zurück, wenn sämtliche dieser Aufrufe terminiert sind.

#### 5.2.1.3. Diskussion

Ist die Topologie des Server-Graphen stabil oder sind die Zustände der Kanäle zwischen den Ausführungs-Servern konstant, ist die Server-basierte Spannbaumberechnung ausreichend. Falls sich aber die Topologie ändert oder die Kanalzustände variieren, muß die Spannbaumberechnung wiederholt werden, um die Integration aller Ausführungs-Server in den Spannbaum sicherzustellen. Zudem erhält man durch Wiederholung der Spannbaumberechnung einen möglichst kleinen Spannbaum, gemessen an der aktuellen Auslastung der Kanäle.

Der vorgestellte Algorithmus kann ohne Modifikation aber nicht mehrfach auf demselben Graphen ausgeführt werden. Da sich die Nachrichten einzelner Aufrufe nicht unterscheiden lassen, werden in ungünstigen Fällen in den einzelnen Knoten die Nachrichten mehrerer Aufrufe gemischt bearbeitet, was zu inkorrekten Ergebnissen führen kann. Eine erneute oder parallele Ausführung des Algorithmus kann nur erfolgen, wenn die Ergebnisse einer vorhergehenden Berechnung gelöscht wurden, oder wenn sich die in den unterschiedlichen Berechnungen erhobenen Daten unterscheiden lassen. Eine mögliche Lösung verwendet Zeitstempel und einen Prioritätsmechanismus zur eindeutigen Kennzeichnung unterschiedlicher Berechnungen. Zusätzlich zu dieser Erweiterung muß eine Synchronisation der diensterbringenden Dokumente mit den Ausführungs-Servern erfolgen, die sicherstellt, daß die Berechnung des verwendeten Spannbaums abgeschlossen ist und daß ein Spannbaum während der Existenz diens-

terbringender Dokumente, die diesen Spannbaum nutzen, nicht aus den Ausführungs-Server entfernt werden.

Diese Synchronisationsaufgaben sind komplex, da sie jeweils die Menge aller Ausführungs-Server einschließen. In den folgenden Abschnitten werden daher Alternativen für eine Spannbaumberechnung in den diensterbringenden Dokumenten und eine Spannbaumberechnung, in der Ausführungs-Server und diensterbringende Dokumente kooperieren, vorgestellt. Diese Alternativen erfordern keine Synchronisation konkurrierender Spannbaumberechnungen.

### 5.2.2. Dokumentbasierte Verteilung

Die in Abschnitt 5.2.1 vorgestellte Lösung zur Berechnung des Spannbaums in den Servern hat den Nachteil, daß erneute Spannbaumberechnungen aufwendig sind. Dennoch sind erneute Spannbaumberechnungen notwendig, um auf Änderungen der Topologie des Server-Graphen und des Auslastungszustands der Kanäle reagieren zu können. Ein alternativer Weg zur Navigation der diensterbringenden Dokumente im Server-Graphen besteht darin, während der Verteilung der Dokumente in den Dokumenten, die bereits besuchten Server zu protokollieren. Die von den Dokumenten verwendeten Daten zur Navigation sind vollständig unabhängig von Spannbaumdaten in den Ausführungs-Servern, was die parallele Verteilung mehrerer Dokumente ermöglicht.

#### 5.2.2.1. Algorithmus zur dokumentbasierten Verteilung

Die Navigation der diensterbringenden Dokumente kann auf einfache Weise durch die Implementierung einer Tiefensuche in den Dokumenten erfolgen. Ein solches Dokument besitzt die folgenden privaten Operationen: private.Start und private.Continue.

Die Verteilung des Dokuments wird mit der Ausführung der Operation private.Start an dem diensterbringenden Dokument initiiert. Das Dokument kopiert sich dann auf einen Ausführungs-Server aus der Nachbarmenge des Servers, auf dem es sich befindet, und ruft an der Kopie die Operation private.Continue auf. Als Argument wird eine Liste aller bisher besuchten Ausführungs-Server, die zu diesem Zeitpunkt nur den Start-Server enthält, übergeben. Die Operation private.Continue liefert eine Liste aller infolge ihrer Ausführung besuchten Server zurück. Die Operation private.Start fährt fort, sich auf die benachbarten Server zu kopieren, die nicht in der Liste der besuchten Server enthalten sind. Das diensterbringende Dokument wurde auf jedem Ausführungs-Server einmal instantiiert, wenn alle Nachfolger des Startknotens in der Liste der besuchten Server enthalten sind. Die private.Start-Operation ist in Programm 5.3 dargestellt.

Die private.Continue-Operation der dokumentbasierten Verteilung nimmt die Liste der besuchten Server als Argument entgegen. Sie kopiert das diensterbringende Dokument auf einen Nachbar-Server, der sich nicht in der Liste befindet und ruft an dem neu erzeugten Dokument die private.Continue-Operation auf. Als Argument übergibt sie die Liste der besuchten Server, die um die eigene Server-Kennung erweitert ist. Das Ergebnis dieses Aufrufs wird in die Liste der besuchten Server integriert. Wenn keine

**Programm 5.3** private.Start-Operation der dokumentbasierten Verteilung

```
// Realisierung der private.Start-Operation.
proc private.Start() :
  // Die Menge der besuchten Knoten mit der Adresse
  // dieses Rechnerknotens initialisieren.
  visited := {serverAddress}
  // Das Dokument auf alle Nachbarknoten kopieren, die
  // noch nicht besucht wurden und an den Kopien die
  // Ausführung der private.Continue-Operationen initiieren.
  for n ∈ get_neighbors() do
    if n ∉ visited then
      // Server-basierter Transport des Dokuments.
      // (self ist eine Referenz auf das Dokument, an
      // dem die aktuelle Operation initiiert wurde.)
      d := s_deliver(self, n)
      // An der Kopie die private.Continue-Operation
      // initiieren und die vom Nachfolger besuchten Knoten der
      // Menge der besuchten Konten hinzufügen.
      visited_by_n := invoke(n, d, private.Continue, visited)
      visited := visited ∪ visited_by_n
    fi
  od
end
```

unbesuchten Nachbarn mehr existieren, löscht die private.Continue-Operation das Dokument und gibt diese integrierte Liste an den Aufrufer zurück. Programm 5.4 zeigt die private.Continue-Operation.

Dieser Algorithmus findet sich unter verschiedenen Namen wieder, z. B. bei Tel als Tiefensuche mit Kenntnis der Nachbarn [93, S. 213]. Er besucht die Knoten des Server-Graphen in einer Depth-First-Ordnung. Der Durchlauf erfolgt sukzessive, d. h. alle Knoten werden nacheinander besucht. Zu einem Zeitpunkt warten alle diensterbringenden Dokumente, bis auf eines, auf die Rückkehr der private.Continue-Operation.

Das sukzessive Durchlaufen der Knoten ist notwendig, da Dokumente anhand des Zustands eines Servers nicht erkennen können, ob ein anderes Dokument ihres Suchlaufs bereits den Server besucht hat. Das wäre bei einer parallelen Bearbeitung verschiedener Nachfolger notwendig, um mehrfache Berechnungen auf einem Ausführungs-Server zu vermeiden. Das sukzessive Durchlaufen sorgt dafür, daß alle Dokumente auf einen gemeinsamen Speicher zugreifen können, der die während des Durchlaufens besuchten Ausführungs-Server enthält. Dieser Speicher besteht in der Liste der besuchten Ausführungs-Server, die bei den Operationsaufrufen zwischen den Dokumenten ausgetauscht wird. Die Speicherräume unterschiedlicher Suchläufe sind voneinander getrennt, da jeder Suchlauf mit einer eigenen Liste beginnt.

**Programm 5.4** private.Continue-Operation der dokumentbasierten Verteilung

```
// Realisierung der private.Continue-Operation.
// Als Argument wird eine Liste der bereits besuchten
// Ausführungs-Server übergeben.
proc private.Continue(visited) :
  // Diesen Ausführungs-Server der Menge der besuchten
  // Ausführungs-Server hinzufügen.
  visited := visited ∪ {serverAddress}
  // Das Dokument auf alle Nachbarknoten kopieren, die
  // noch nicht besucht wurden und an den Kopien die
  // Ausführung der private.Continue-Operationen initiieren.
  for n ∈ get_neighbors() do
    if n ∉ visited then
      // An jeder Dokumentkopie die Ausführung der
      // private.Continue-Operation initiieren.
      d := s_deliver(self, n)
      visited_by_n := invoke(n, d, private.Continue, visited)
      // Die von den Kopien besuchten Knoten der
      // Menge der besuchten Konten hinzufügen.
      visited := visited ∪ visited_by_n
    fi
  od
  // Das Dokument von diesem Ausführungs-Server entfernen
  // und die Menge der besuchten Knoten an den Aufrufer
  // übergeben.
  delete()
  return(visited)
end
```

#### 5.2.2.2. Diskussion

Der Vorteil der dokumentbasierten Spannbaumberechnung ist, daß durch die getrennten Speicherräume zur Speicherung der besuchten Knoten beliebig viele parallele Spannbaumberechnungen möglich sind. Jede dieser Berechnungen basiert auf der aktuellen Topologie. Die Nachteile der dokumentbasierten Spannbaumberechnung sind der Umfang der zwischen den Operationen versendeten Daten sowie die Laufzeit des Algorithmus.

Wenn $N$ die Anzahl der Server und $B$ die Zahl der Bits für die Kodierung der Server-Namen ist, und wenn man annimmt, daß keine zusätzlichen Informationen zur Kodierung der Liste benötigt werden, liegt die Zahl der in den Listen insgesamt ausgetauschten Bits in der Größenordnung $O(N^2B)$. Ausgetauschte Nachrichten enthalten maximal alle Knoten, also $NB$ Bits. Zu jedem unbekannten Knoten wird eine Liste der besuchten Server gesendet und später von ihm empfangen. Der Startknoten ist zu keinem Zeitpunkt unbekannt. Daher wird insgesamt $2(N-1)$-mal eine Liste der be-

suchten Server versendet. Insgesamt führt dies zu maximal $2(N^2B-NB) \in O(N^2B)$ ausgetauschten Bits.

Das sukzessive Besuchen der Ausführungs-Server führt darüber hinaus zu einer Laufzeit, die in der Größenordnung der Zahl der Server liegt. Eine Parallelisierung der Berechnungen auf unterschiedlichen Ausführungs-Servern wird durch diesen Algorithmus nicht ermöglicht.

Aufgrund der genannten Nachteile erscheint dieser Ansatz zur Verteilung diensterbringender Dokumente nicht praktikabel. Allerdings lassen sich die genannten Vorteile erhalten und die Nachteile weitgehend vermeiden, indem die Spannbaumberechnung in den Ausführungs-Servern und die dokumentbasierte Verteilung kombiniert werden, wie in Abschnitt 5.2.3 beschrieben wird.

### 5.2.3. Kooperative Verteilung

Eine Alternative, die sowohl die verteilte Speicherung als auch die Möglichkeit zur Ausnutzung von Parallelität bietet, liegt in einer Verteilung der Aufgaben zwischen den Ausführungs-Servern und den diensterbringenden Dokumenten. In diesem Verfahren stellen die Server den Dokumenten Speicher zur Verfügung, den sie nutzen können, um ihren Weg durch den Server-Graphen zu markieren. Damit lassen sich mehrfache Besuche eines Servers erkennen. Dieses Verfahren wird im folgenden als *kooperative Verteilung* bezeichnet.

#### 5.2.3.1. Algorithmus zur kooperativen Verteilung

Die kooperative Verteilung verwendet den in Abschnitt 5.2.1 beschriebenen Algorithmus. Dieser Algorithmus wird diesmal aber nicht in den Servern, sondern in den Dokumenten zur kooperativen Verteilung ausgeführt. Anders als die in Abschnitt 5.2.2 beschriebenen Dokumente tauschen die Dokumente zur kooperativen Verteilung allerdings keine Listen besuchter Ausführungs-Server aus, sondern hinterlassen Markierungen auf den Ausführungs-Servern, die sie bereits besucht haben. Zu diesem Zweck stellen die Ausführungs-Server an der Schnittstelle der Laufzeitumgebungen in der Funktionsgruppe Traversierung (siehe Abschnitt 3.3.5.3) die Funktionen set_mark und delete_mark bereit, mit denen solche Kennzeichen erzeugt bzw. gelöscht werden können.

Die Funktion set_mark ist atomar und prüft, bevor sie die angegebene Markierung setzt, ob die Markierung bereits vorhanden ist. Ist dies der Fall, liefert sie ein negatives Ergebnis, ansonsten speichert der Server die Markierung und liefert ein positives Ergebnis. Die Funktion remove_mark ermöglicht das Löschen einer Markierung. Sie ist ebenfalls atomar und liefert ein positives Ergebnis, falls eine zu löschende Markierung existierte, ansonsten ein negatives Ergebnis.

Ein Dokument zur kooperativen Verteilung verfügt über eine private.Start-Operation und eine private.Continue-Operation. Die kooperative Verteilung wird mit der Ausführung der Operation private.Start an dem diensterbringenden Dokument initiiert. Die der private.Start-Operation zugeordnete Dokumentmethode erzeugt zunächst einen eindeutigen Bezeichner für den Verteilungsvorgang und markiert den aktuellen

Ausführungs-Server mit dieser Bezeichnung. Anschließend kopiert es sich auf alle benachbarten Ausführungs-Server und initiiert die Ausführung der private.Continue-Operation an den Kopien. Als Argument wird die eindeutige Bezeichnung des Verteilungsvorgangs sowie die Adresse des eigenen Ausführungs-Servers übergeben, die private.Continue-Operation liefert kein Ergebnis zurück. Wenn alle private.Continue-Operationen terminiert sind, ist die Verteilung vollständig durchgeführt worden. Programm 5.5 zeigt die private.Start-Operation des Dokuments zur kooperativen Verteilung der diensterbringenden Dokumente.

**Programm 5.5** private.Start-Operation der kooperativen Verteilung

```
// Realisierung der private.Start-Operation der kooperativen Verteilung.
proc private.Start() :
  // Eindeutigen Bezeichner für diesen Verteilungsvorgang
  // erzeugen und diesen Ausführungs-Server markieren.
  id := serverAddress + getTime()
  set_mark(id)
  handle_set := ∅
  for n ∈ get_neighbors() do
      // Das Dokument auf jeden benachbarten Ausführungs-Server
      // kopieren und dort die Ausführung der
      // private.Continue-Operation initiieren. Als Argumente
      // werden der Bezeichner des Verteilungsvorgangs
      // und die Adresse dieses Ausführungs-Servers übergeben.
      d := s_deliver(self, n)
      handle_set := handle_set ∪ {invoke_async(n, d, private.Continue, id,
          serverAddress)}
  od
  // Rückkehr der Aufrufe abwarten und Methode beenden.
  while handle_set do
        handle := wait_for_result(handle_set)
        handle_set := handle_set \ {handle}
  od
end
```

Die private.Continue-Operation der kooperativen Verteilung erhält als Argument die eindeutige Bezeichnung des Verteilungsvorgangs, in die sie involviert ist. Sie versucht nach ihrer Initiierung zunächst, den Server mit dieser Bezeichnung zu markieren. Schlägt dies fehl, weil die Markierung schon auf dem Server existiert, beendet sich die private.Continue-Operation. Andernfalls verhält sie sich analog zu der private.Start-Operation des Dokuments zur kooperativen Verteilung. Sie kopiert das diensterbringende Dokument auf alle benachbarten Ausführungs-Server und initiiert die Ausführung der private.Continue-Operation an den Kopien, wobei sie die Bezeichnung des Verteilungsvorgangs übergibt. Die Operation entfernt das Dokument von dem Server, auf dem sie sich befindet und terminiert, wenn alle von ihr initiierten private.Continue-Operationen terminiert sind. Die private.Continue-Operation des Dokuments zur kooperativen Verteilung ist in Programm 5.6 abgebildet.

**Programm 5.6** private.Continue-Operation der kooperativen Verteilung

```
// Realisierung der private.Continue-Operation der kooperativen Verteilung.
// Als Argumente werden der eindeutige Bezeichner der Verteilung
// und die Adresse des Ausführungs-Servers übergeben, von dem
// aus die Ausführung initiiert wurde.
proc private.Continue(id, caller) :
  // Falls dieser Ausführungs-Server bereits markiert ist,
  // wird keine weitere Aktion auf diesem Server ausgeführt.
  if ¬set_mark(id) then
    return
  fi
  handle_set := ∅
  for n ∈ get_neighbors() \ {caller} do
      // Das Dokument auf jeden benachbarten Ausführungs-Server
      // außer dem aufrufenden kopieren und dort die Ausführung der
      // private.Continue-Operation initiieren. Als Argumente
      // werden der Bezeichner der Spannbaumberechnung
      // und die Adresse dieses Ausführungs-Servers übergeben.
      d := s_deliver(self, n)
      handle_set := handle_set ∪ {invoke_async(n, d, private.Continue, id,
          serverAddress)}
  od
  // Rückkehr der Aufrufe abwarten.
  while handle_set do
        handle := wait_for_result(handle_set)
        handle_set.delete(handle)
  od
  // Das Dokument von diesem Ausführungs-Server entfernen
  // und die Methode beenden.
  delete()
end
```

Das beschriebene Verhalten der Operationen private.Start und private.Continue des Dokuments zur kooperativen Verteilung führt zur parallelen Ausführung der private.Continue-Operationen.

#### 5.2.3.2. Löschen von Markierungen

Nimmt man an, daß die Server eine endliche Menge an Speicher für die Markierungen bereitstellen, müssen Markierungen wieder vom Server entfernt werden. Hier bieten sich zwei Möglichkeiten an: ein zweiter Durchlauf durch den Server-Graphen, bei dem die Markierungen entfernt werden, oder die zeitbasierte Entfernung der Markierungen.

Die aktive Entfernung der Markierungen kann mit Hilfe eines speziellen Dokuments zum Löschen der Markierungen erfolgen. Dieses Dokument kann allerdings nicht die

kooperative Verteilung verwenden, da es keine neuen Markierungen erzeugen darf. Eine Alternative bestünde in der Anwendung der dokumentbasierten Verteilung. Es gibt aber noch eine zweite Möglichkeit, die den Umstand ausnutzt, daß das Fehlen einer Markierung auf einem Ausführungs-Server die Information beinhaltet, daß das Dokument zum Löschen der Markierung diesen Ausführungs-Server schon erreicht hat. Dies ist der Fall, da jeder Bezeichner für einen Verteilungsvorgang nur einmal verwendet wird.

Ein Dokument zu Löschen, das das Fehlen von Markierungen ausnutzt verfügt ebenfalls eine private.Start- und eine private.Continue-Operation, die der private.Start- und private.Continue-Operation des Dokuments zur kooperativen Verteilung ähnlich sind. Der wesentliche Unterschied besteht darin, daß die Operationen versuchen, die Markierung von den Ausführungs-Server zu entfernen, und sich beenden, wenn die Markierung bereits entfernt worden war. Programm 5.7 zeigt die private.Start-Operation des Löschdokuments. Die private.Continue-Operation des Löschdokuments ist in Programm 5.8 abgebildet.

**Programm 5.7** private.Start-Operation des Löschdokuments

```
// Realisierung der private.Start-Operation des Löschdokuments.
// Als Argument wird der zu löschende Bezeichner übergeben.
proc private.Start(id) :
  // Die Markierung von diesem Ausführungs-Server entfernen.
  remove_mark(id)
  for n ∈ get_neighbors() do
    d := s_deliver(self, n)
    // Das Dokument auf alle benachbarten Ausführungs-Server
    // kopieren und an den Kopien die Ausführung der
    // private.Continue-Operation initiieren.
    // Als Argumente werden der zu löschende Bezeichner
    // und die Adresse dieses Servers übergeben.
    invoke(n, d, private.Continue, id, serverAddress)
  od
end
```

Alternativ zur Löschung durch einen separaten Durchlauf bietet sich die zeitbasierte bzw. die bedarfsgesteuerte Löschung der ältesten Markierung an. In diesem Fall muß die Zahl der Dienstdokumente pro Zeiteinheit sowie die Verweildauer der Dienstdokumente im Server-Verbund bei der Festlegung des maximalen Alters einer Markierung bzw. bei der Dimensionierung des Speichers für Markierungen berücksichtigt werden.

#### 5.2.3.3. Diskussion

Die kooperative Verteilung erlaubt die parallele Bearbeitung von Operationen auf unterschiedlichen Ausführungs-Servern und ist daher in ihrer Laufzeit mit der Laufzeit der Server-basierten Spannbaumberechnung vergleichbar. Die systemweit eindeutige Bezeichnung der Verteilung, die in der private.Start-Operation generiert wird, er-

**Programm 5.8** private.Continue-Operation des Löschdokuments

```
// Realisierung der private.Start-Operation des Löschdokuments.
// Als Argument werden der zu löschende Bezeichner und
// die Adresse des Ausführungs-Server, von dem aus die
// Ausführung dieser Methode initiiert wurde übergeben.
proc private.Continue(id, caller) :
  // Die Markierung von diesem Ausführungs-Server entfernen.
  // Falls die Markierung auf diesem Ausführungs-Server nicht
  // existierte, wird dieser Zweig der Berechnung beendet.
  if ¬remove_mark(id) then
    return
  fi
  for n ∈ get_neighbors() \ {caller} do
      // Das Dokument auf alle benachbarten Ausführungs-Server,
      // außer demjenigen, von dem aus diese Methode initiiert
      // wurde kopieren. An den Kopien die Ausführung der
      // private.Continue-Operation mit dem zu löschenden Bezeichner
      // und der Adresse dieses Servers als Argumente initiieren.
      d := s_deliver(self, n)
      invoke(n, d, private.Continue, id, serverAddress)
  od
  // Das Dokument von diesem Ausführungs-Server entfernen
  // und die Methode beenden.
  delete()
end
```

laubt die parallele Speicherung der Informationen mehrerer Verteilungsläufe in den Ausführungs-Servern. Damit bietet diese Variante den Vorteil der Server-basierten Spannbaumberechnung, die verteilte Speicherung des Spannbaums und den Vorteil der dokumentbasierten Verteilung, die dynamische Anpassung an Topologieänderungen.

Ein Nachteil dieses Verfahrens ist, daß der Markierungszustand der Ausführungs-Server den private.Continue-Operationen erst nach ihrem Transport auf den Ausführungs-Server zugänglich ist. Dadurch erhöht sich die Zahl der Nachrichten, die gesendet werden müssen, denn ein Dokument muß erst auf einen Server übertragen werden bevor es feststellen kann, daß die Berechnung dort bereits ausgeführt wurde. Das bedeutet im ungünstigsten Fall, daß ein Dokument jeweils von beiden Seiten über ein Kante gesendet wird. Wenn $E$ die Menge der Kanten im Server-Graphen ist, werden $2|E|$ Nachrichten gesendet. Nehmen wir an, das die Zahl $B$ der pro Nachricht ausgetauschten Bits konstant ist, ergibt sich ein Bitaufwand von $2B|E| \in O(|E|)$ (vgl. [51, S. 498]). Im ungünstigsten Fall, bei einer total vermaschten Topologie, bedeutet dies den Austausch von $2B(N^2 - N)$ Bits.

Es gibt Möglichkeiten zur Verbesserung dieser Situation. In den oben beschriebenen Operationen erhält die private.Continue-Operation den eindeutigen Namen des Ausführungs-Servers, von dem aus sie initiiert wurde als Argument. Dieser Server

muß dann nicht mehr besucht werden, denn er ist notwendigerweise markiert. Mit der eingeführten Optimierung reduziert sich die Zahl der Nachrichten um $N - 1$, da alle Knoten außer dem Start-Knoten eine Nachricht einsparen.

### 5.2.4. Auswahl

Aufgrund der oben genannten Vorteile wird in Indigo die kooperative Verteilung der Dienstlogik mit einer zeitbasierten Entfernung der Markierungen verwendet. In den Ausführungs-Servern ist dazu die Unterstützung der Verwaltung von Markierungen notwendig. Das zeitbasierte Löschen der Markierungen wurde gewählt, um die Entfernung von Markierungen nicht vollständig vom Verhalten der Dienstdokumente bzw. der Löschdokumente abhängig zu machen. Zusätzlich zu der zeitbasierten Entfernung können Dokumente zum Löschen von Markierungen eingesetzt werden.

Während die kooperative Verteilung durch entsprechende Mechanismen der Ausführungs-Server unterstützt werden muß, ist die dokumentbasierte Verteilung auf keinerlei weitergehende Unterstützung durch die Ausführungs-Server angewiesen. Der Einsatz der dokumentbasierten Verteilung der Dienstlogik ist daher in Indigo ebenfalls möglich, falls die Server-Menge klein genug ist, um zentral, d. h. auf einem Ausführungs-Server, gespeichert zu werden.

## 5.3. Integration von Diensterbringung und Verteilung

Im den vorhergehenden Abschnitten wurden Dokumente vorgestellt, die sich selbst in dem Verbund der Ausführungs-Server verteilen. Bei der Verteilung werden Kopien dieser Dokumente auf jeden der Ausführungs-Server transportiert. Ziel der Verteilung der Dokumente ist die Ausführung dienstspezifischer Operationen auf jedem Ausführungs-Server.

Um dieses Ziel zu erreichen muß ein Weg zur Integration der zur Diensterbringung notwendigen Berechnungen in die zur Verteilung notwendigen Berechnungen gefunden werden. Dazu muß zunächst die Frage beantwortet werden, welche Ereignisse beim Durchlaufen des Server-Verbundes relevant sind. Dies sind genau die Ereignisse, die beim Durchlaufen eines Baumes oder Graphen allgemein von Interesse sind:

**Enter** Der Zeitpunkt nach der Ankunft eines Dokuments auf einem noch nicht besuchten Knoten. Zu diesem Zeitpunkt können dienstspezifische Argumente des Operationsaufrufs bearbeitet werden.

**Leave** Der Zeitpunkt vor dem Beenden der private.Start- bzw. der private.Continue-Operation, also nachdem alle Nachbarknoten bearbeitet wurden. Zu diesem Zeitpunkt kann ein dienstspezifisches Ergebnis errechnet werden, das dann an den Initiator der Operation zurückgeliefert wird.

**PreSubnode** Der Zeitpunkt vor der Initiierung der private.Continue-Operation auf einem Nachfolgerknoten. Zu diesem Zeitpunkt können dienstspezifische Argumente für den Aufruf der private.Continue-Operation auf den Nachbarknoten erzeugt werden.

**PostSubnode** Der Zeitpunkt nach dem Erhalt eines Ergebnisses von einem Nachfolgerknoten. Zu diesem Zeitpunkt können die vom Nachfolgeknoten erhaltenen dienstspezifischen Resultate verarbeitet werden.

Beim Eintritt eines dieser Ereignisse muß die Möglichkeit zur Ausführung dienstspezifischer Berechnungen bestehen. Im folgenden werden zwei Möglichkeiten vorgestellt, dienstspezifische Berechnungen beim Eintritt dieser Ereignisse in der Verteilung auszuführen: die operationsbasierte Integration und die programmierte Integration.

### 5.3.1. Operationsbasierte Integration

Diese Variante der Integration von dienstspezifischen Berechnungen in die Verteilung verwendet private Operationen zur Implementierung der dienstspezifischen Berechnungen. Das erlaubt die Definition eines Dokumentmusters zur verteilten Realisierung von Diensten. Dieses Dokumentmuster verfügt über eine private.Start- und private.Continue-Operation sowie über die vier privaten Operationen private.Enter, private.Leave, private.PreSubnode und private.PostSubnode. Die vier letztgenannten Operationen korrespondieren mit den Ereignissen der Durchlaufens.

**private.Start** Die private.Start-Operation entspricht im wesentlichen der private.Start-Operation des Dokuments zur kooperativen Verteilung (siehe Abschnitt 5.2.3.1). Sie erhält als zusätzliches Argument dienstspezifische Argumente und initiiert die private.Enter-, private.Leave-, private.PreSubnode- und private.PostSubnode-Operationen des eigenen Dokuments, wenn die entsprechenden Ereignisse eintreten.

**private.Continue** Die private.Continue-Operation entspricht im wesentlichen der private.Continue-Operation zur kooperativen Verteilung, sie erhält ebenfalls ein zusätzliches Argument zur Übergabe dienstspezifischer Argumente. Analog zur private.Start-Operation des Dokumentmusters initiiert sie beim Eintritt der entsprechenden Ereignisse die Ausführung der private.Enter-, private.Leave-, private.PreSubnode- und private.PostSubnode-Operationen des eigenen Dokuments.

**private.Enter** Die private.Enter-Operation erhält als Argumente die dienstspezifischen Argumente, die bei der Initiierung der private.Start-Operation oder als Ergebnis der Ausführung der private.PreSubnode-Operation bereitgestellt wurden.

**private.Leave** Die private.Leave-Operation erhält keine weiteren Argumente, ihre Ausgabe wird als dienstspezifisches Ergebnis der Berechnung auf dem Ausführungs-Server an die Instanz übergeben, die die Operationsausführung initiierte. Dies war entweder die private.Continue- oder die private.Start-Operation.

**private.PreSubnode** Die private.PreSubnode-Operation erhält als Argument die Adresse des Ausführungs-Servers, auf den die Kopie des Dokuments transportiert wird. Ihr Resultat wird als dienstspezifisches Argument bei der Initiierung

der private.Continue-Operation und damit bei der Initiierung der private.Enter-Operation an der Kopie des Dokuments auf dem entsprechenden benachbarten Ausführungs-Server verwendet.

**private.PostSubnode** Die private.PostSubnode-Operation erhält als Argument das dienstspezifische Ergebnis, das von der private.Continue-Operation und damit indirekt von der private.Leave-Operation des benachbarten Ausführungs-Servers zurückgeliefert wurde.

Wie die Verarbeitung der Ereignisse in der private.Start-Operation durchgeführt wird ist in Programm 5.9 dargestellt, die Integration der Ereignisverarbeitung in die private.Continue-Operationen findet sich in Programm 5.10.

**Programm 5.9** private.Start-Operation zur operationsbasierten Integration

```
// Dienstspezifische Argumente werden in args übergeben.
proc private.Start(args) :
  id := serverAddress + getTime()
  set_mark(id)
  // Enter-Ereigniss bearbeiten.
  private.Enter(args)
  handle_set := ∅
  for n ∈ get_neighbors() do
      d := s_deliver(self, n)
      // PreSubnode-Ereigniss bearbeiten.
      n_args := private.PreSubnode(n)
      // Dienstspezifische Argumente an private.Continue
      // werden in n_args übergeben.
      handle_set := handle_set ∪ {invoke_async(n, d, private.Continue, id,
          serverAddress, n_args)}
  od
  while handle_set do
        handle := wait_for_result(handle_set)
        // PostSubnode-Ereigniss bearbeiten.
        private.PostsubNode(get_result(handle))
        handle_set := handle_set \ {handle}
  od
  // Leave-Ereigniss bearbeiten.
  result := private.Leave()
  return(result)
end
```

Der Vorteil der operationsbasierten Integration liegt in der Trennung der Berechnungen zur Verteilung und der Ereignisverarbeitung. Damit wird eine Beeinträchtigung der Verteilung durch die dienstspezifischen Operationen weitgehend ausgeschlossen. Die Trennung erlaubt darüber hinaus die Verwendung eines Dokumentmusters zur operationsbasierten Integration.

**Programm 5.10** private.Continue-Operation zur operationsbasierten Integration

```
// Dienstspezifische Argumente werden in args übergeben.
proc private.Continue(id, caller, args) :
  if ¬set_mark(id) then
    return
  fi
  // Enter-Ereigniss bearbeiten.
  private.Enter(args)
  handle_set := ∅
  for n ∈ get_neighbors() \ {caller} do
      d := s_deliver(self, n)
      // PreSubnode-Ereigniss bearbeiten.
      n_args := private.PreSubnode(n)
      // Dienstspezifische Argumente an private.Continue
      // werden in n_args übergeben.
      handle_set := handle_set ∪ {invoke_async(n, d, private.Continue, id, caller,
          n_args)}
  od
  while handle_set do
        handle := wait_for_result(handle_set)
        // PostSubnode-Ereigniss bearbeiten.
        private.PostsubNode(get_result(handle))
        handle_set := handle_set \ {handle}
  od
  // Leave-Ereigniss bearbeiten.
  result := private.Leave()
  delete()
  return(result)
end
```

Die Trennung der Kontexte hat aber auch den Nachteil, daß Argumente und Ergebnisse zwischen Kontexten verschiedener Operationen transportiert werden müssen, was i. a. aufwendig ist. Darüber hinaus erfordert z. B. der Zugriff der Dokumente auf gemeinsame Zustandsvariablen das Verfügbarmachen der Variablen in unterschiedlichen Kontexten, z. B. durch die Speicherung der Variablen im lokal verfügbaren temporären Speicher oder im Dokumentinhalt, der in diesem Fall auch von temporärer Natur ist, denn das Dokument wird nach der vollständigen Bearbeitung des Knotens gelöscht. Im folgenden Abschnitt wird kurz auf eine alternative Möglichkeit zur Integration von Verteilung und dienstspezifischer Berechnung eingegangen, die den Transport dienstspezifischer Daten zwischen getrennten Kontexten vermeidet.

### 5.3.2. Programmierte Integration

Eine Alternative zur operationsbasierten Integration der dienstspezifischen Berechnungen in die Verteilung ist die direkte Integration der Dienstlogik an den entsprechenden Stellen in der private.Start- und die private.Continue-Operation (siehe Programm 5.9 und 5.10).[6] Der Vorteil besteht in der Vermeidung des Transports von Daten über die Kontexte unterschiedlicher Operationsausführungen hinaus. Nachteilig an der programmierten Integration ist die Tatsache, daß Fehler bei der Programmierung der ereignisverarbeitenden Prozeduren zu Fehlern bei der Verteilung und infolgedessen zu einer fehlerhaften Diensterbringung und zu einem unbegrenzten Verweilen der diensterbringenden Dokumente im Server-Verbund führen können.

Die Verfahren zur operationsbasierten oder programmierten Integration schließen sich nicht gegenseitig aus. Beide können in Indigo verwendet werden. Die Auswahl hängt von den jeweiligen Umständen ab, insbesondere vom Umfang der zwischen den Operationen auszutauschenden Daten sowie vom Arbeitsaufwand für die programmierte Integration. Die Verfahren können ebenso ein Dokument zur dokumentbasierten Verteilung verwenden, mit den in Abschnitt 5.2.2 genannten Vor- und Nachteilen.

## 5.4. Beispiele für Dienste der Digitalen Bibliothek

In diesem Abschnitt werden Beispiele für Dienstdokumente vorgestellt. Im einzelnen wird beschrieben, wie ein Katalog aller Dokumente bzw. ein Index über alle Dokumente erstellt und wie eine Suche durchgeführt werden kann.

### 5.4.1. Katalog- und Indexerstellung

Ziel der Katalogerstellung und der Indexerstellung ist die Ausführung einer Operation, die für jedes Dokument der Infrastruktur Informationen erzeugt, die dann zum Aufbau eines Katalogs oder eines Index genutzt werden. Der Katalog sammelt Informationen über alle Dokumente in der Digitalen Bibliothek. Der Umfang der Informationen, die zu den einzelnen Dokumenten gespeichert werden, ist je nach Katalogart unterschiedlich. Bei einem reinen Existenznachweis genügt die Speicherung des eindeutigen Dokumentnamens, bei einem Standortnachweis benötigt man zusätzlich den Ort an dem das Dokument gespeichert wird und bei einem Sachkatalog zusätzlich Informationen über den Inhalt des Dokuments.

Der vollständige Katalog einer Digitalen Bibliothek kann ein beträchtliches Datenvolumen besitzen. In dem vorliegenden Beispiel wird daher ein verteiltes Dokument zur Speicherung des Katalogs verwendet. Ein solches Dokument kann auf ähnliche Weise wie ein Dokument zur verteilten Speicherung von Baumstrukturen (siehe Abschnitt 4.3.2) konstruiert werden. Im folgenden wird angenommen, daß ein Dokument zur Speicherung eines Kataloges existiert, das u. a. über eine private.AddEntries-

[6] Der Programmtext muß nicht vollständig an den genannten Position stehen, die Implementierungen (siehe Kapitel 6) erlauben sowohl die Nutzung von Prozeduren bzw. Funktionen als auch die Modularisierung des Programmtextes.

Operation verfügt, die das Hinzufügen einer Menge von Einträgen in den Katalog erlaubt.

Ein Dokument zur Katalogerstellung kann dann aus dem Dokumentmuster zur verteilten Realisierung von Diensten durch eine entsprechende Implementierung der private.Enter-Operation erstellt werden. Die private.Leave-, private.PreSubnode- und private.PostSubnode-Operationen werden von dem Dokument zur Katalogerstellung nicht verwendet. Zusätzlich zu den durch das Dokumentmuster vorhandenen Operationen verfügt das Dokument zu Katalogerstellung über eine private.Create-Operation, besitzt also u. a. folgende Operationen:

**private.Create** Die private.Create-Operation dient der Initiierung der Katalogerstellung. Sie erhält als Argument den eindeutigen Dokumentnamen eines Dokuments zur Speicherung des Katalogs. Diesen übergibt sie an die private.Start-Operation, wenn sie die Verteilung initiiert.

**private.Enter** Die private.Enter-Operation erhält als Eingabe den Namen des Dokuments zur Speicherung des Katalogs (siehe Abschnitt 5.3.1). Sie sammelt Daten über alle auf dem lokalen Ausführungs-Server vorhandenen Dokumente, indem sie die entsprechenden Operationen der Dokumente ausführt. Das Ergebnis dieser Berechnung übermittelt sie an das Dokument zur Speicherung des Katalogs, indem sie dessen private.addEntries-Operation aufruft.

Die private.Create-Operation des Dokuments zur Katalogerstellung ist in Programm 5.11 dargestellt.

**Programm 5.11** Die private.Create-Operation des Dokuments zur Katalogerstellung

```
// Realisierung der private.Create-Operation des Dokuments zur Katalogerstellung.
// Als Argument wir der Name des Dokuments zur Speicherung des
// Katalogs übergeben.
proc private.Create(catalog) :
   // Verteilung mit dem Namen des Katalogdokuments als
   // dienstspezifisches Argument starten.
   invoke(serverAddress, self, private.Start, catalog)
end
```

Programm 5.12 skizziert die Realisierung der private.Enter-Operation des Dokuments zur Katalogerstellung.

Die Erstellung eines Indexes kann analog zu der Erstellung des Katalogs durchgeführt werden. Statt eines Dokuments zur Speicherung eines Katalogs wird bei der Initiierung der private.Create-Operation lediglich ein Dokument zur Speicherung eines Indexes als Argument übergeben. In der private.Enter-Operation des Dokuments zur Indexerstellung wird dann ein Teilindex erzeugt, der über den Aufruf der private.addIndex-Operation des Dokuments zur Speicherung des Indexes in den bestehenden Index integriert wird. Programm 5.13 stellt die private.Enter-Operation des Dokuments zur Indexerzeugung dar. Auf die Darstellung der private.Create-Operation

**Programm 5.12** Die private.Enter-Operation des Dokuments zur Katalogerstellung

```
// Realisierung der private.Enter-Operation des Dokuments zur Katalogerstellung.
// Als Argument wir der Name des Dokuments zur Speicherung des
// Katalogs übergeben.
proc private.Enter(catalog) :
   // Erzeuge Teilkatalog aller lokalen Dokumenten.
   part := makeCatalog()
   // Füge den Teilkatalog in das Dokument zur Speicherung
   // des Katalogs ein.
   invoke(server, catalog, private.AddEntries, part)
end
```

zur Initiierung der Indexerzeugung wurde hier verzichtet, da sie der private.create-Operation der Katalogerstellung entspricht.

**Programm 5.13** Die private.Enter-Operation des Dokuments zur Indexerstellung

```
// Name des Dokuments zur Speicherung des Indexes
// wird als dienstspezifisches Argument übergeben.
proc private.Enter(index) :
   // Erzeuge Teilindex aller lokalen Dokumenten.
   part := makeIndex()
   // Füge Teilindex in das Dokument zur Speicherung
   // des Index ein.
   invoke(server, index, private.AddIndex, part)
end
```

### 5.4.2. Suche

Die Suche über Dokumente kann mittels des erzeugten Index geschehen, wenn zwei Voraussetzungen erfüllt sind: die Suche wird durch den Index unterstützt, und der Index reflektiert den aktuellen Zustand der Digitalen Bibliothek. Treffen diese Voraussetzungen nicht zu, bietet sich die Möglichkeit, eine Suche durch ein Dienst-Dokument zu implementieren. In diesem Dokument kann dann eine benutzerspezifische Suche realisiert werden, die z. B. die von der orthogonalen Describe-Operation gelieferte Beschreibung lokal verarbeitet und das Ergebnis an den Initiator der Suche zurücksendet. Die benutzerspezifische Suche könnte z. B. Konzeptgraphen zur Suche in Bilddokumenten (vgl. [66, 22, 38]) und Video-Dokumenten (vgl. [29]) verwenden.

Im vorliegenden Abschnitt wird ein Dienstdokument vorgestellt, das die Describe-Operation zur Ermittlung von Informationen über die lokalen Dokumente nutzt, um diese zu einer Anfrage in Beziehung zu setzen. Es führt eine Rangfolgenanordnung der lokalen Dokumente durch und liefert die 100 am höchsten bewerteten Dokumente

als Ergebnis zurück.[7] Der Initiator der Operationsausführung integriert die Ergebnisse des Ausführungs-Servers, auf dem er sich befindet, sowie die Ergebnisse aller benachbarten Ausführungs-Server und liefert seinerseits die 100 höchstbewerteten Dokumente als Gesamtergebnis der Suche zurück. Die Suche bietet ein Beispiel für die Übertragung von Resultaten als Rückgabewert des Aufrufs.

Das Dienstdokument zur Suche verwendet eine programmierte Integration der Dienstlogik in die Verteilung. Programm 5.14 skizziert die private.Continue-Operation das Dokuments zur Suche.

**Programm 5.14** Die private.Continue-Operation des Dokuments zur Suche

```
// Die Anfrage wird in query übergeben.
proc private.Continue(id, caller, query) :
  if ¬set_mark(id) then
    return
  fi
  // Die 100 relevantesten lokalen Dokumente berechnen.
  result := searchTop100(query)
  handle_set := ∅
  for n ∈ get_neighbors() \ {caller} do
      d := s_deliver(self, n)
      // Die Anfrage an die benachbarten Ausführungs-Server übergeben.
      handle_set := handle_set ∪ {invoke_async(n, d, private.Continue, id,
          serverAddress, query)}
  od
  while handle_set do
        handle := wait_for_result(handle_set)
        // Das Resultat in die bestehende Rangfolge integrieren.
        result := integrateTop100(result, get_result(handle))
        handle_set := handle_set \ {handle}
  od
  delete()
  // Die 100 relevantesten Dokumente des Teilbaums,
  // dessen Wurzel dieser Server ist, zurückliefern.
  return(result)
end
```

## 5.5. Zusammenfassung

Die transparente Integration neuer Dienste in Digitale Bibliotheken ist von ebenso großer Bedeutung, wie die transparente Integration neuer Dokumenttypen. Dabei soll eine skalierbare Realisierung neuer Dienste gewährleistet sein. In dem vorliegenden Kapitel wurde eine Möglichkeit aufgezeigt, solche Dienste dezentral mit Hilfe von

[7] Die Zahl 100 wurde hier willkürlich gewählt.

mobilen Dokumenten zu realisieren, und es wurden verschiedene Dokumente zur Erbringung von Diensten vorgestellt. Die Möglichkeiten zur verteilten Realisierung eines Dienstes sind keineswegs auf die beschriebenen Mechanismen beschränkt, die Einbeziehung der Dokumente in die Verteilung ermöglicht die Realisierung weiterer Verteilungsmuster.

Der Vorteil der Verwendung mobiler Dokumente liegt in der dynamischen Erweiterbarkeit des Dienstspektrums und in der Möglichkeit, diese Erweiterung individuell durchführen zu können. Dadurch können neue Anforderungen schnell umgesetzt werden, ohne eine Modifikation der Basiselemente der Digitalen Bibliothek, wie z. B. der Ausführungs-Server vornehmen zu müssen.

Der hier vorgestellte Ansatz hat den Nachteil, daß die Flexibilität mit Performance-Einbußen und, aufgrund des notwendigen Transports der Dokumente, mit einem erhöhten Bandbreitenbedarf gegenüber statischen Dienstimplementierungen erkauft wird. Diese Effekte lassen sich aber durch die Verwendung von Caching-Mechanismen zur Speicherung von Dokumentmethoden minimieren. Werden Dokumentmethoden in Caches gespeichert sind Übertragungen von Dokumentmethoden nur dann notwendig, wenn ein neuer Dienst etabliert wird. Die statische Implementierung von Diensten bietet bei der Einführung eines neuen Dienste dann, im Bezug auf den Transport von Dokumentmethoden, keinerlei Vorteile mehr, da hier ebenfalls der Transport der Dienstlogik auf die Ausführungs-Server notwendig ist und darüber hinaus die Notwendigkeit der manuellen Modifikation der Ausführungs-Server besteht.

Die Idee, mobile Komponenten zur dynamischen Integration neuer Dienste in Digitale Bibliotheken zu nutzen, wurde auch in anderen Projekten aufgegriffen. Die SARA Digital Library unterstützt mobile User Request Agents (URAs) zum Transport dienstspezifischer Berechnungen auf Repositorien (vgl. [107]). Im Zusammenhang mit Uniform Resource Agents, die ebenfalls mit dem Akronym URA bezeichnet werden, wurde ebenfalls die Möglichkeit zum Transport von Programmfunktionen auf Datenspeicher, um datenintensive Operationen lokal ausführen zu können, diskutiert (vgl. [18]).

# 6. Realisierung der Infrastruktur

Die Realisierung der in den vorangegangenen Kapiteln entworfenen Infrastruktur für Digitale Bibliotheken beinhaltet die Realisierung der Ausführungs-Server, die Definition eines Protokolls zur Initiierung von Befehlen an den Ausführungs-Servern sowie die Implementierung von Dokumenten unterschiedlicher Dokumenttypen.

## 6.1. Kommunikation der Ausführungs-Server

Für die Kommunikation einer Anwenderkomponente mit einem Ausführungs-Server bzw. zwischen Ausführungs-Servern ist ein Protokoll notwendig, das die Initiierung der Befehle der Ausführungs-Server (siehe Abschnitt 3.3.4) erlaubt und den Transport von Metadokumenten zwischen Ausführungs-Servern unterstützt. In der vorliegenden Implementierung wurde das Hypertext Transfer Protocol (HTTP) [3] zur Kommunikation zwischen Ausführungs-Servern sowie zwischen Ausführungs-Servern und Anwenderkomponenten verwendet. Die Gründe für den Einsatz von HTTP sind (vgl. [50]):

- HTTP ist wohlverstanden und akzeptiert. Die technischen und administrativen Voraussetzungen für seinen Einsatz sind allgemein erfüllt.
- HTTP stellt alle notwendigen Kommandos bereit und erlaubt die Erweiterung um neue Kommandos, falls dies notwendig werden sollte (vgl. [30]).
- Eine HTTP-basierte Infrastruktur kann von den vielfältigen Forschungs-, Standardisierungs- und Entwicklungsaktivitäten im Bereich des WWW Gebrauch machen. Beispiele dafür sind Mechanismen zur vertraulichen Kommunikation und Authentifizierung, z. B. Prüfsummen-basierte Authentifizierung (vgl. [32]), Secure Socket Layer (SSL) [36] oder Secure HTTP [77].

### 6.1.1. HTTP-basiertes Kommunikationsprotokoll

Die Verwendung von HTTP zur Kommunikation zwischen Ausführungs-Servern erfordert die Umsetzung der Befehle der Ausführungs-Server auf entsprechende HTTP-Requests. In der Realisierung von Indigo werden dazu spezifische lokale URLs verwendet, die von den Ausführungs-Servern als Befehle interpretiert werden. Tabelle 6.1 zeigt die in Indigo vorgenommene Abbildung der Befehle der Ausführungs-Server auf HTTP-Requests.

**Tabelle 6.1.** Abbildung der Server-Befehle auf HTTP-Requests

| Befehl | HTTP-Request |
|---|---|
| Documents | `GET /diglib/documents` |
| Runtimes | `GET /diglib/runtimes` |
| Attributes | `GET /diglib/attributes/`*Dokumentname* |
| Methods | `GET /diglib/methods/`*Dokumentname* |
| Invoke | `GET /diglib/invoke/`*Dokumentname*`?`*Methode*`&`*Argumente* |
| Deliver | `POST /diglib/deliver?`*Methode*`&`*Argumente* |
| AddMethod | `GET /diglib/addmethod/`*Dokumentname*`?` *Methode*`&`*URI*`&`*Laufzeitumgebung*`&`*Signatur* |

Um z. B. den Documents-Befehl eines Servers aufzurufen, wird der HTTP-Befehl GET auf der lokalen URL /diglib/documents ausgeführt, wie in der folgenden Zeile dargestellt.

```
GET /diglib/documents HTTP/1.0
```

Der Ausführungs-Server sendet als Antwort auf diesen HTTP-Request einen HTTP-Response an den Aufrufer, der das Ergebnis der Befehlsausführung enthält. Der Rumpf der Response-Meldungen besteht aus einzelnen Feldern, deren Format sich an dem Format für Textnachrichten im Internet (vgl. [17]) orientiert. Die Antwort des Ausführungs-Servers auf den Erhalt des oben genannten Befehls könnte wie folgt aussehen:

```
HTTP/1.0 200 OK
Date: Fri 17 Nov 2000 13:50:23
Server: Indigo/0.1
Content-type: text/plain

Documents:
2420-989830249-8597
2439-989830273-9697
```

In diesem Beispiel liefert der Ausführungs-Server eine Liste namens „Documents“ mit zwei lokal eindeutigen Dokumentbezeichnern zurück.

Im Rahmen der Realisierung der Infrastruktur wurden kommandozeilenorientierte Anwenderkomponenten zur programmgesteuerten Ausführung von Befehlen an Ausführungs-Servern entwickelt. Als Anwenderkomponenten lassen sich aber auch WWW-Browser oder telnet-Clients verwenden.

### 6.1.2. Kodierung der Metadokumente

Metadokumente (siehe Abschnitt 3.3.3) werden bei der Durchführung des Deliver-Befehls im Rumpf des POST-Requests an den Ausführungs-Server übergeben. Für

die notwendige Kodierung wird das in den Multipurpose Internet Mail Extensions definierte MIME-Format [34] verwendet. Metadokumente werden dabei in Form einer MIME-Multipart Nachricht (vgl. [35]) mit dem in Indigo definierten MIME-Typ *application/x-metadoc* kodiert. Die Verwendung von MIME erlaubt, analog zur Verwendung von HTTP, die Nutzung aktueller Entwicklungen wie z. B. Secure/Multipurpose Internet Mail Extensions (S/MIME) [76].

Eine MIME-Multipart Nachricht zum Transport eines Metadokuments besteht aus drei Teilen, die mit den drei Bestandteilen eines Metadokuments korrespondieren: Attribute, dokumentspezifische Zuordnungsfunktion und Dokumentinhalt.

**Attribute** Die Attribute des Dokuments werden in einem MIME-Datenobjekt des in Indigo definierten Typs *application/x-metadoc-attributes* gespeichert. Listing 6.1 zeigt einen Ausschnitt aus einem Metadokument, der die Repräsentation einiger der in Abschnitt 3.3.3.3 definierten Attribute illustriert:

**Listing 6.1** Repräsentation der Attribute im Metadokument

```
Content-Type: application/x-metadoc; boundary=ekidpUnhoJ

--ekidpUnhoJ
Content-Type: application/x-metadoc-attributes

Author:
-----BEGIN CERTIFICATE-----
MIIEKjCCA5OgAwIBAgIBADANBgkqhkiG9w0BAQQFADCBxTELMAkGA1UEBhMCREUx
DzANBgNVBAgTBkhlc3NlbjEaMBgGA1UEBxMRRnJhbmtmdXJ0IGFtIE1haW4xIjAg

      ⋮

xfbcEFOTR61Kt8rFv39KrE3UvByfc+fOs1pqSrKn/q+bd1qQaJT2pGLEI2YWlm+v
YiKT3WXQK9XXzoY62aw=
-----END CERTIFICATE-----

MethodDigests:
Present b3718b2aeb87baabbd95af1f17e67372
Describe 27687765e4680b26a675cebb60e57d9d
private.localPresent 39272ff08de697020947badab0c91752

--ekidpUnhoJ

      ⋮

--ekidpUnhoJ--
```

Das Author-Attribut des hier abgebildeten Metadokuments enthält ein X.509-Zertifikat des Autors, das MethodDigests-Attribut enthält MD5-Prüfsummen über die jeweiligen Dokumentmethoden bzw. die Archivdateien, in denen Dokumentmethoden gespeichert werden (siehe Abschnitt 6.2.3).

**Dokumentspezifische Zuordnungsfunktion** Die dokumentspezifische Zuordnungsfunktion wird in einem MIME-Datenobjekt des in Indigo definierten Typs *application/x-metadoc-methods* gespeichert. Dieser Teil des Metadokuments enthält

für jede Dokumentmethode eine Zeile, in der folgende Informationen stehen: der Name der Operation, der URI[1] der Dokumentmethode bzw. einer Archivdatei, in der die Dokumentmethode gespeichert ist (siehe Abschnitt 6.2.3) und der Name der Laufzeitumgebung, die die Dokumentmethode benötigt. Die Repräsentation der dokumentspezifischen Zuordnungsfunktion ist in Listing 6.2 dargestellt. Gegenwärtig existieren Laufzeitumgebungen zur Ausführung von Python-basierten und Java-basierten Dokumentmethoden. Die Laufzeitumgebungen sind mit den systemweit eindeutigen Namen *application/x-python* respektive *application/x-java* bezeichnet.

**Listing 6.2** Repräsentation der Zuordnungsfunktion im Metadokument

```
Content-Type: application/x-metadoc; boundary=ekidpUnhoJ

      ⋮

--ekidpUnhoJ
Content-Type: application/x-metadoc-methods

Present http://141.2.14.28/image/present.zip  application/x-python
Describe http://141.2.14.28/image/describe.zip application/x-python
private.localPresent http://141.2.14.28/image/lpresent.zip application/x-
python
--ekidpUnhoJ

      ⋮

--ekidpUnhoJ--
```

**Dokumentinhalt** Der Dokumentinhalt wird in einem MIME-Datenobjekt des in Indigo definierten experimentellen Typs *application/x-metadoc-content* gespeichert. Eine Kodierung für die Daten wird nicht vorgeschrieben. Der Dokumentautor kann die Kodierung frei wählen, er muß lediglich die Randbedingungen der MIME-Spezifikation einhalten. Listing 6.3 stellt einen Ausschnitt aus dem Dokumentinhalt des in Abschnitt 6.3.1 beschriebenen Bilddokuments dar. Es handelt sich dabei um ein Base64-kodiertes Bild im GIF-Format.

Es fällt auf, daß im Metadokument keine Angabe über die Kodierung des Dokumentinhalts existiert. Diese Information muß nicht innerhalb des Metadokuments gespeichert werden, da die Dokumentmethoden über die Kodierung des Inhalts informiert sind.

## 6.2. Realisierung der Ausführungs-Server

Die Aufgabe der Ausführungs-Server besteht in der Durchführung der in Abschnitt 3.3.4 vorgestellten Befehle. Die Durchführung dieser Befehle beinhaltet den Austausch und die Speicherung von Metadokumenten, die transparente Bereitstellung von Dokumentmethoden und deren Instantiierung in entsprechenden Laufzeitumgebungen sowie die Gewährleistung des sicheren Zugriffs der Dokumentmethoden auf

[1] Zur Zeit unterstützen die implementierten Ausführungs-Server lediglich die Angabe von URLs.

**Listing 6.3** Repräsentation des Dokumentinhalts im Metadokument

```
Content-Type: application/x-metadoc; boundary=ekidpUnhoJ

      ⋮

--ekidpUnhoJ
Content-Type: application/x-metadoc-content

R0lGODdhgAJAAfcAAAgcIAggKRAkKRAwKRAwQRA0ORA4ShBAWhBIYhggGBgsORg4GBg4ShhA
UhhIWhhMahhQahhVcyAsICBQUiBVaiBZeyBhcyBlgyk8ICk8MSlIKSlpgzEkGDFVKTFVSjFl

      ⋮

JY+2Yl/2ZlESaJ0WZT+2R5OVYgHTaF+2RpFVXZ32aDeCZIW2YxlCZf/yWrGzYusVTI80bXf2
OL+UWp3VVQu2R8W2aWu1M21/dGztVVVttWP38zOaVmv9lm9N9W93VWV7VnDFVm8blke99lYZ
V2179WG1hFdXtSECAgA7
--ekidpUnhoJ--
```

die elementaren und Indigo-spezifischen Ressourcen des Rechnerknotens, auf dem sie sich befinden.

Im Rahmen der vorliegenden Arbeit wurden Ausführungs-Server unter Verwendung der Programmiersprache Python [79] und unter Verwendung der Programmiersprache Java [41] realisiert. Im folgenden wird der Python-basierte Ausführungs-Server näher beschrieben und am Schluß dieses Abschnitts kurz auf die Java-basierte Implementierung eingegangen.

### 6.2.1. Struktur der Ausführungs-Server

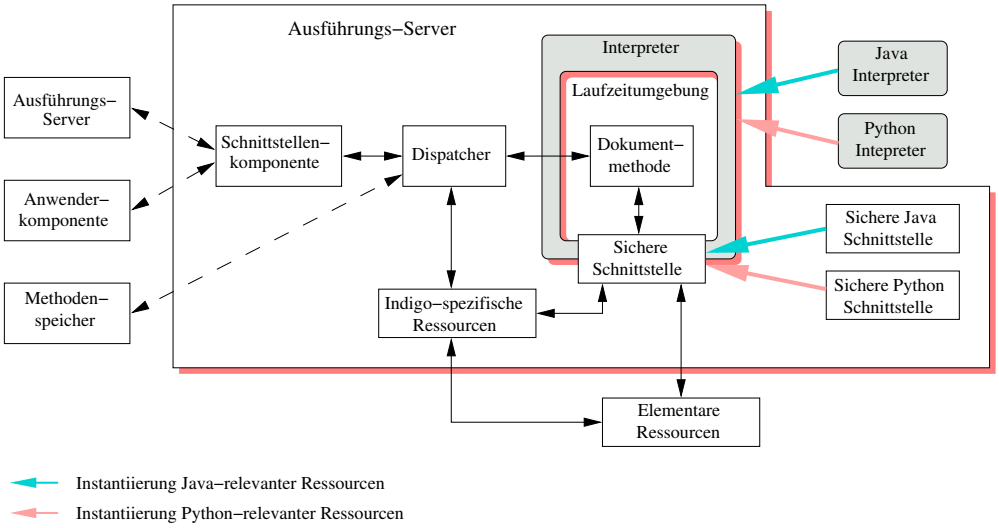
Der Python-basierte Ausführungs-Server setzt sich aus folgenden Komponenten zusammen:

- Einer oder mehrerer Schnittstellenkomponenten, die die Kommunikation mit einer Anwenderkomponente oder einem weiteren Ausführungs-Server übernehmen. Schnittstellenkomponenten führen eine Transformation zwischen dem verwendeten Kommunikationsprotokoll und der intern vom Ausführungs-Server verwendeten Darstellung von Befehlen, Parametern und Metadokumenten durch.
- Einer Dispatcherkomponente, die Befehle in der internen Darstellung entgegennimmt und die entsprechenden Aktionen veranlaßt.
- Einer Menge von Programmbibliotheken, die für jede Laufzeitumgebung eine sichere Schnittstelle zu Indigo-spezifischen und elementaren Ressourcen bereitstellen.
- Indigo-spezifischen Ressourcen.

Daneben benötigt ein Ausführungs-Server einen oder mehrere Interpreter, zur Bereitstellung der Laufzeitumgebungen für die Ausführung von Dokumentmethoden.

Die Interpreter sind, zusammen mit den genannten Bestandteilen eines Ausführungs-Servers, in Abbildung 6.1 dargestellt.

**Abbildung 6.1.** Komponenten des Ausführungs-Servers




### 6.2.2. Die Schnittstellenkomponente

Die Schnittstellenkomponente des Ausführungs-Server dient der Trennung der protokollorientierten und der befehlsorientierten Aspekte des Ausführungs-Servers. Durch den Austausch der Schnittstellenkomponente oder eine Erweiterung um neue Schnittstellenkomponenten läßt sich das vom Ausführungs-Server verwendete Protokoll austauschen bzw. ein neues Protokoll integrieren. Die vorliegende Implementierung verfügt über eine Schnittstellenkomponente, die das in Abschnitt 6.1 vorgestellte HTTP-basierte Protokoll und die Kodierung von Metadokumenten als MIME-Multipart Nachrichten unterstützt.

### 6.2.3. Der Dispatcher

Die Aufgaben des Dispatchers bestehen in der Durchführung der Server-Befehle. In Abbildung 6.1 ist die Ausführung einer Operation durch Instantiierung der entsprechenden Dokumentmethode dargestellt. Dieser Vorgang beinhaltet den Transport der Dokumentmethode von einem Methodenspeicher in den lokalen Speicher des Ausführungs-Servers, der Bestandteil der Indigo-spezifischen Ressourcen ist. Nachdem die Dokumentmethode lokal vorhanden ist, wird eine eventuell vorhandene Prüfsumme kontrolliert und bei Übereinstimmung eine entsprechende Laufzeitumgebung erzeugt. Dies kann im vorliegenden Fall eine Umgebung für Java-basierte oder ein Umgebung für Python-basierte Dokumentmethoden sein. Die Erzeugung der Laufzeitumgebung beinhaltet die Instantiierung des entsprechenden Interpreters und

die Montage der dazugehörigen Schnittstelle zur Sicherung der Ressourcen gegen unbefugte Zugriffe (siehe Abschnitt 6.2.4). Die instantiierte Methode greift über diese Schnittstelle auf die Indigo-spezifischen Ressourcen und die elementaren Ressourcen des Rechnerknotens zu, auf dem sich der Ausführungs-Server befindet. Die Ergebnisse der Ausführung der Dokumentmethode werden vom Dispatcher über die entsprechende Schnittstellenkomponente an den Aufrufer übermittelt.

Die implementierten Ausführungs-Server unterstützen Dokumentmethoden, die aus mehreren Dateien bestehen. Dies ist häufig wünschenswert, denn zum einen tritt bei komplexen Methoden der Wunsch nach Modularisierung auf. Java schreibt die Modularisierung bei der Verwendung mehrerer Klassen sogar vor, selbst bei der Verwendung nur einer Klasse und damit nur einer Quelltextdatei erzeugen Java-Compiler mehrere Klassen-Dateien, wenn in dem Quelltext interne oder anonyme Klassen verwendet werden. Zum anderen ermöglicht erst die Verwendung mehrerer Dateien die Nutzung existierender Module und Klassen. Davon wurde z. B. bei der Realisierung der Online-Dokumente Gebrauch gemacht, indem das Java Media Framework zur Implementierung der Dokumentmethoden genutzt wurde (siehe Abschnitt 6.3.2). Die Bündelung und der Transport mehrerer Dateien wird durch die Verwendung von Archiven, d. h. Dateien, in denen die Inhalte und Strukturen mehrerer Dateien zusammengefaßt sind, ermöglicht. Die in Listing 6.2 dargestellte dokumentspezifische Zuordnungsfunktion referenziert keine ausführbaren Python- oder Java-Programme, sondern Archive. Der Dispatcher speichert die Inhalte der Archive in getrennten Bereichen, so daß alle zu einer Dokumentmethode gehörenden Dateien sich in einem eigenen Namensraum befinden. Dadurch wird eine Kollision zwischen Dateien, die zu unterschiedlichen Dokumentmethoden gehören, vermieden.

Die Ausführung der Dokumentmethode geschieht laufzeitspezifisch durch Zugriff auf die Klasse bzw. das Modul, das einen entsprechenden Namen trägt. So wird eine Java-basierte Dokumentmethode zur Durchführung der Present-Operation durch Aufruf der öffentlichen statischen Methode *main* der Klasse *Present* aufgerufen. Eine Python-basierte Dokumentmethode zur Durchführung der Present-Operation wird durch Ausführung der Befehle im Python-Modul *Present.py* gestartet.

### 6.2.4. Laufzeitumgebungen

Die Bereitstellung der Laufzeitumgebungen für Java- bzw. Python-basierte Dokumentmethoden geschieht durch die Verwendung existierender Interpreter. Dabei sind folgende Anpassungen zur Realisierung sicherer Laufzeitumgebungen notwendig:

- Die Bereitstellung von Schnittstellen zu den Indigo-spezifischen Ressourcen.
- Der Schutz der elementaren Ressourcen und der Indigo-spezifischen Ressourcen gegen unberechtigte Zugriffe von Dokumentmethoden.

Diese beiden Aufgaben sind in Abbildung 6.1 jeweils in einer Komponente zusammengefaßt dargestellt. Die Komponenten sind mit *Sichere Java Schnittstelle* respektive *Sichere Python Schnittstelle* bezeichnet.

**Laufzeitumgebung für Java-basierte Dokumentmethoden** Die Schnittstellen zu den Indigo-spezifischen Ressourcen werden für Java-basierte Dokumentmethoden durch die Klasse *de.uni_frankfurt.indigo.ExecServer* zum Zugriff auf die Server-Funktionen, die Klasse *de.uni_frankfurt.indigo.Document* zum Zugriff auf die Dokument-Funktionen und die Klasse *de.uni_frankfurt.indigo.Traverse* zum Zugriff auf die Traversierungsfunktionen realisiert. Die Methoden dieser Klassen prüfen die Zulässigkeit der Aufrufe vor ihrer Ausführung.

Neben den Methoden dieser drei Klassen müssen zusätzlich die sicherheitskritischen Methoden aller anderen, in der Laufzeitumgebung verfügbaren Klassen, abgesichert werden. Dies wird durch die Implementierung eines Sicherheits-Managers realisiert. Ist ein solcher Sicherheits-Manager vorhanden, stellt der Java-Interpreter vor der Ausführung einer Java-Methode eine Anfrage an den Sicherheits-Manager, ob die Methode mit den jeweiligen Argumenten ausgeführt werden darf. In der Realisierung geschieht dies durch Aufrufe von Methoden an einer Instanz einer von der Klasse *SecurityManager* abgeleiteten Klasse. In Indigo wurde eine Klasse namens *IndigoSecurityManager* erstellt, die alle sicherheitskritischen Methodenaufrufe vor ihrer Ausführung prüft.

**Laufzeitumgebung für Python-basierte Dokumentmethoden** Die Schnittstellen zu den Indigo-spezifischen Ressourcen werden für Python-basierte Dokumentmethoden durch das Modul *execserver* zum Zugriff auf die Server-Funktionen, das Modul *document* zum Zugriff auf die Dokument-Funktionen und das Modul *traverse* zum Zugriff auf die Traversierungsfunktionen realisiert. Die Funktionen dieser Module prüfen ebenfalls die Zulässigkeit der Aufrufe vor ihrer Ausführung.

Die Absicherung der Ressourcen gegen den Aufruf anderer Funktionen geschieht durch den Prozeßkontext, in dem die Python-basierten Dokumentmethoden ausgeführt werden. Eine feingranularere Ausführung ließe sich durch Verwendung der Standard Module *rexec* und *Bastion* erreichen (vgl [78, S. 399 ff]).

### 6.2.5. Eine alternative Realisierung

Zusätzlich zu der Python-basierten Implementierung wurde eine Java-basierte Implementierung eines Ausführungs-Server vorgenommen. Der Grund für die Erstellung dieses alternativen Ausführungs-Servers war, daß dieser prinzipiell als Applet in Java-fähigen WWW-Browsern, z. B. dem Netscape Navigator oder dem Internet Explorer, ausgeführt werden kann. Durch die Ausführung des Ausführungs-Servers im WWW-Browser des Anwenders entfällt der bei der Python-basierten Implementierung notwendige Installationsaufwand, so daß eine ad hoc-Nutzung der Infrastruktur möglich wird.

Der Java-basierte Ausführungs-Server unterscheidet sich von dem oben vorgestellten Python-basierten Ausführungs-Server vor allem in zwei Punkten: in den unterstützten Laufzeitumgebungen und in dem Kontext der Methodenausführung. Aufgrund der Beschränkung auf die vom Browser bereitgestellte Java Umgebung werden nur Java-basierte Dokumentmethoden unterstützt. Im Gegensatz zu dem Python-basierten Ausführungs-Server instantiiert der Java-basierte Ausführungs-Server die

von ihm ausgeführten Methoden nicht in dedizierten Prozessen, sondern erzeugt eine Thread-Gruppe, in der die Dokumentmethode als Thread instantiiert wird. Der Zugriff auf die Ressourcen des Rechnerknotens durch die Mitglieder dieser Thread-Gruppe wird durch einen Sicherheitsmanager begrenzt. Die Verwendung einer Thread-Gruppe ermöglicht die Ausführung und Kontrolle von Dokumentmethoden, die mehrere Threads erzeugen.

Die Ausführung eines Java-basierten Ausführungs-Servers als Applet ist allerdings nicht uneingeschränkt möglich. Sie setzt die Möglichkeit zur feingranularen Festlegung von Zugriffsrechten, wie sie Java 1.2 bzw. Java 2.0 oder der Internet Explorer bietet, voraus. Für eine Ausführung des Ausführungs-Servers im Netscape Navigator ist dagegen die Verwendung spezifischer Sicherheits-Klassen notwendig, die den Anwender interaktiv zur Gewährung von Zugriffsrechten auffordern. Diese Unterschiede in den von den Browsern bereitgestellten Laufzeitumgebungen für Applets führen zu der Notwendigkeit der aufwendigen Erstellung Browser-spezifischer Versionen des Java-basierten Ausführungs-Servers, wenn dieser als Applet ausgeführt werden soll.[2] Aus diesem Grund wurde die Realisierung der Ausführungs-Server als Applet nicht weiter verfolgt.

## 6.3. Beispiele realisierter Dokumente

In diesem Abschnitt wird ein Teil der Dokumente vorgestellt, die implementiert wurden. Die Beispiele illustrieren die Realisierung einiger der in Kapitel 4 beschrieben Dokumentmuster und Dokumente.

### 6.3.1. Offline-Präsentation

Das Offline-Dokument realisiert die in Abschnitt 4.2.1 beschriebene Offline-Präsentation. Nach einer Etablierungsphase, in der das Dokument zum anwenderseitigen Ausführungs-Server übertragen wird, läuft die Präsentation vollständig innerhalb des anwenderseitigen Ausführungs-Servers ab.

Dokumente zur Offline-Präsentation besitzen typischerweise drei Operationen: Present, Describe und private.localPresent. Die Präsentation beginnt mit der Ausführung der orthogonalen Present-Operation an dem zu präsentierenden Dokument. Die der Present-Operation zugeordnete Dokumentmethode wird auf dem Ausführungs-Server, auf dem das Dokument gespeichert ist, ausgeführt. Ein Ausschnitt der entsprechenden Python-basierten Methode, u. a. wurden die Befehle zur Fehlerbehandlung entfernt, ist in Listing 6.4 dargestellt.

Die Dokumentmethode überträgt das Dokument, auf dem sie ausgeführt wird, mittels des Server-basierten Transports auf den als Argument angegebenen Ziel-Server. Nachdem das Dokument zum Ziel-Server übertragen wurde initiiert die Dokumentmethode

[2] Der Java-basierte Ausführungs-Server läßt sich auch als Java Applikation ausführen. In diesem Fall ist eine ad hoc-Nutzung aber nicht mehr möglich, da ebenso wie beim Einsatz des Python-basierten Ausführungs-Server eine vorhergehende Installation notwendig ist.

**Listing 6.4** Die Present-Methode des Offline-Dokuments

```
# Present.py

def main(self, host, port):
    # Zugriff auf Server-Funktionen über das Modul execserver.
    import execserver

    # Das eigene Dokument auf den Ziel-Server kopieren. Resultat
    # ist der lokal eindeutige Name der Kopie.
    new_id = execserver.s_deliver(host, port, self)

    # Ausführung von private.localPresent an der Kopie initiieren.
    execsever.invoke(host, port, new_id, 'private.localPresent')
    return


# Die Ausführung der Methode begint hier, als Argumente werden
# der Name des Dokuments, die IP-Adresse und Portnummer des
# Ziel-Servers übergeben.
if __name__ == '__main__':
    import sys, string

    main(sys.argv[1], sys.argv[2], string.atoi(sys.argv[3]))
    return
```

die Ausführung der private.localPresent-Operation auf dem neuen Dokument.[3] Die Dokumentmethode zur Realisierung der Present-Operation ist generisch, da sie von jedem Dokument, das dem Muster der Offline-Präsentation folgt, verwendet werden kann. Die Verwendung erfordert lediglich die Kopie der entsprechenden Zeile des Metadokuments in das neu zu erstellende Dokument. Die der private.localPresent-Operation zugeordnete Dokumentmethode findet sich, ebenfalls ausschnittsweise, in Listing 6.5.

Die hier vorgestellte Dokumentmethode ist für alle GIF-kodierten Bilder, die Base64 kodiert sind, verwendbar. Auf der graphischen Benutzeroberfläche des Anwenders erscheint als Nebeneffekt der Ausführung der Dokumentmethode eine graphische Darstellung des Bildes wie sie in Abbildung 6.2 dargestellt ist.

[3] Diese Vorgehensweise tritt so häufig auf, daß eine Implementierung erstellt wurde, die einen a priori Transport des Dokuments zum anwenderseitigen Ausführungs-Server durchführt (vgl. [89]).

**Listing 6.5** Die private.localPresent-Methode des Offline-Dokuments

```
# private.localPresent.py

def main(argv):
    # Zugriff auf Dokument-Funktionen über das Modul document.
    import document

    # Zugriff auf elementare Ressourcen.
    import Tkinter, tempfile, base64

    # Bild in das File plain dekodieren.
    coded = document.open()
    plain = tempfile.TemporaryFile()
    base64.decode(coded, plain)
    plain.seek(0)

    # Bild darstellen
    mainWindow = Tkinter.toplevel()
    img = Tkinter.PhotoImage(file=plain)
    canvas = Tkinter.Canvas(mainWindow, width=img.width(), height=img.height())
    canvas.create_image(0, 0, image=img, anchor='nw')
    canvas.configure(scrollregion=(0, 0, img.width(), img.height()))
    canvas.pack()
    mainWindow.mainloop()
    return

# Die Ausführung der Methode begint hier, als Argument wird
# der Name des Dokuments übergeben.
if __name__ == '__main__':
    import sys
    sys.exit(main(sys.argv))
```

**Abbildung 6.2.** Präsentation des Offline-Dokuments


### 6.3.2. Online-Dokument

Bei diesem Beispiel handelt es sich um ein Dokument, das eine Online-Präsentation durchführt, wie sie in Abschnitt 4.2.2 beschrieben ist. Um die Präsentation vorzunehmen, kommunizieren eine anwenderseitige und eine speicherseitige Dokumentmethode miteinander. Dokumente zur Online-Präsentation besitzen ebenfalls typischerweise drei Operationen: Present, Describe und private.localPresent.

Die Präsentation beginnt, wie bei jedem Dokument, mit der Ausführung der orthogonalen Present-Operation an dem zu präsentierenden Dokument. Die der Present-Operation zugeordnete Dokumentmethode wird auf dem Ausführungs-Server, auf dem das Dokument gespeichert ist, ausgeführt. Sie überträgt eine Kopie des Dokuments mittels des methodenbasierten Transports auf den Ausführungs-Server, dessen Adresse ihr als Argument übergeben wurde. Der methodenbasierte Transport führt dazu, daß ein Metadokument mit leerem Inhalt, wie in Listing 6.6 dargestellt, an den Ziel-Server übertragen wird.

**Listing 6.6** Von m_deliver erzeugtes, leeres Metadokument

```
      ⋮

--ekidpUnhoJ
Content-Type: application/x-metadoc-content

--ekidpUnhoJ--
```

Nach Abschluß des Kopiervorgangs erzeugt die der Present-Operation zugeordnete Methode einen Netzzugangspunkt (Service Access Point, SAP), an dem sie den Inhalt des Dokuments bereitstellt. Im vorliegenden Beispiel wird das Java Media Framework 2.0 [91] zur Erzeugung eines Servers verwendet, der synchrone Daten als RTP-Strom [84] bereitstellt. Nach der Erzeugung des SAP wird dieser bei der Initiierung der private.localPresent-Operation an der Kopie als Argument übergeben. Wenn der SAP kontaktiert wird, liefert die Methode die entsprechenden Dokumentdaten. Die Java-basierte Methode zur Implementierung der Present-Operation des Online-Dokuments ist ausschnittsweise in Listing 6.7 wiedergegeben.

Die der private.localPresent-Operation zugeordnete Methode kontaktiert den RTP-Server, dessen SAP ihr als Argument übergeben wurde, und präsentiert die von ihm erhaltenen Daten. Die private.localPresent-Methode ist ausschnittsweise in Listing 6.8 dargestellt.

Bei dem Beispieldokument handelt es sich um ein Video-Dokument. Die Ausführung der Methode zur Implementierung der private.localPresent-Operation des Online-Dokuments auf dem anwenderseitigen Ausführungs-Server führt als Nebeneffekt zur Darstellung der in Abbildung 6.3 dargestellten Benutzeroberfläche zur Präsentation des Dokuments.

### 6.3.3. Ein kooperativ genutztes Dokument

Als Beispiel für ein kooperativ genutztes Dokument wird hier ein Dokument vorgestellt, das einen Chat-Raum realisiert. Das Dokument folgt dem in Abschnitt 4.2.3 vorgestellten Dokumentmuster, je nach Zahl der anwenderseitigen Dokumentmethoden wird eine Server-Methode gestartet bzw. gestoppt.

Das hier vorgestellte Chat-Dokument besitzt fünf Operationen: Present, Describe, private.localPresent, private.startServer und private.stopServer. Die Present-Operation initiiert die Präsentation auf dem angegebenen Ausführungs-Server. Sie prüft zunächst, ob bereits ein Chat-Server auf dem lokalen Ausführungs-Server exis-

**Listing 6.7** Ausschnitte der Present-Methode des Online-Dokuments

```
// Zugriff auf Indigo-spezifische Ressourcen
import de.uni_frankfurt.indigo.*;
// Zugriff auf elementare Ressourcen
import java.net.*;
import javax.media.*;

public class Present {
    // RTP-Server erstellen und starten.
    public static void serve(String source, String dest) throws Exception {

            ...

        // RTP-Server starten.
        aProcessor.start();
    }

    // Die Ausführung der Methode begint hier, als Argumente werden
    // der Name des Dokuments, die IP-Adresse und Portnummer des
    // Ziel-Servers übergeben.
    public static void main(String args[]) throws Exception {

        // SAP erzeugen.
        String SAP = "rtp://" + InetAddress.getByName(args[1]).getHostAddress()
            + ":" + Integer.toString((randSource.nextInt() % 1000) * 2) + "/video";

        // Führe methodenbasierten Transport durch.
        String newId = ExecServer.m_deliver(args[1], number, args[0]);

        // Starte Video-Server.
        serve(rtpSource, SAP);

        // Starte private.localPresent-Operation an der Kopie
        // des Dokuments, der SAP wird als Argument übergeben.
        ExecServer.invoke(args[1], args[2], newId,
            "private.localPresent", SAP);
        return;
    }
}
```

tiert. Wenn dies nicht der Fall ist, wird der Chat-Server durch Initiierung der private.startServer-Operation gestartet. Existiert dagegen bereits ein Chat-Server, wird das Dokument auf den Ziel-Server kopiert und die private.localPresent-Operation initiiert, wobei der SAP des Chat-Servers als Argument übergeben wird. Nachdem die private.localPresent-Operation beendet ist, prüft die Present-Operation, ob weitere anwenderseitige Dokumentmethoden den Chat-Server nutzen, wenn das nicht der Fall ist, wird der Chat-Server beendet.

Die Implementierung der Present-Operation des Chat-Dokuments in Python ist ausschnittsweise in Listing 6.9 dargestellt.

Die der Present-Operation zugeordnete Dokumentmethode verwendet die enter- und leave-Funktionen der Funktionsgruppe Dokument (siehe Abschnitt 3.3.5.2) zur Sicherung der Prüfung auf anwenderseitige Dokumentmethoden. Dies ist notwendig, da die Ausführung der Present-Operation nebenläufig initiiert werden kann.

Die der private.startServer-Operation zugeordnete Methode erzeugt einen Server-Thread, der das Chat-Protokoll ausführt. Die Kennung dieses Threads wird in dem statischen Dictionary des Dokuments (siehe Abschnitt 3.3.5.2) gespeichert. Die der

**Listing 6.8** Die private.localPresent-Methode des Online-Dokuments

```
// Zugriff auf Indigo-spezifische Ressourcen
import de.uni_frankfurt.indigo.*;
// Zugriff auf elementare Ressourcen
import java.net.*;
import javax.media.*;

      ...

public class localPresent {

    // Die Ausführung der private.localPresent-Methode
    // startet hier.
    public static void main(String args[]) throws Exception {

        MediaPlayer aPlayer = new MediaPlayer();
        aPlayer.setMediaLocation(args[1]);

          ...

        aPlayer.start();
        return;
    }
}
```

**Abbildung 6.3.** Präsentation des Online-Dokuments


private.stopServer-Operation zugeordnete Dokumentmethode liest die Kennung aus dem statischen Dictionary und beendet den Server-Thread. Der Server-Thread nimmt von den anwenderseitigen Dokumentmethoden Texte entgegen und verteilt diese zusammen mit dem Namen des Absenders an alle an ihn gebundenen anwenderseitigen Dokumentmethoden. Darüber hinaus erlaubt das Protokoll die Bekanntgabe von Namen.

Die der private.localPresent-Operationen zugeordnete Dokumentmethode erfragt vom Anwender einen Namen zur Identifikation im Chat-Raum und baut anschließend eine Verbindung mit dem Server-Thread auf. Die anwenderseitigen Dokumentmethoden senden vom Anwender eingegebenen Text an den Chat-Server und präsentieren alle vom Chat-Server erhaltenen Texte dem Anwender. Abbildung 6.4 stellt die Benutzeroberflächen zweier Teilnehmer an einer Unterhaltung dar.

An der hier dargestellten Unterhaltung sind drei Personen beteiligt, die einen Termin und einen Ort für ein Treffen festlegen. Die Benutzeroberflächen stellen den Beiträgen

**Listing 6.9** Die Present-Methode des Chat-Dokuments

```
# Present.py

# Diese Funktion wird mit dem lokal eindeutigen Namen des Dokuments
# und der IP-Adresse und Portnummer des Ziel-Servers aufgerufen.
def main(self, host, port):
    # Zugriff auf Indigo-spezifische Ressourcen.
    import execserver, document

    # Existenz eines Servers prüfen.
    document.enter('server_sem')
    clients = document.get_static('clients') + 1
    if clients == 1:
        server_address = execserver.invoke(
            execserver.host(), execserver.port(), self,
            'private.startServer')
        server_address = document.set_static('server_address',
            server_address)
    else:
        server_address = document.get_static('server_address')
    document.set_static('clients', clients)
    document.leave('server_sem')

    # Das eigene Dokument auf den Ziel-Server kopieren und die
    # Ausführung von private.localPresent an der Kopie initiieren.
    new_id = execserver.s_deliver(host, port, self)
    execsever.invoke(host, port, new_id, 'private.localPresent',
        server_address)

    # Server beenden, falls keine weiteren anwenderseitigen
    # Dokumentmethoden existieren.
    document.enter('server_sem')
    clients = document.get_static('clients') - 1
    if clients == 0:
        execserver.invoke(
            execserver.host(), execserver.port(), self,
            'private.stopServer', server_address)
    document.set_static('clients', clients)
    document.leave('server_sem')
    return
```

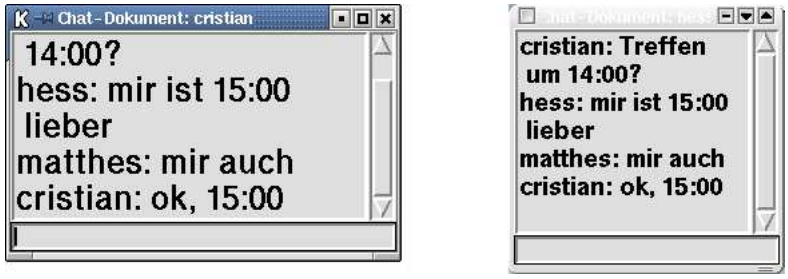
der Teilnehmer die jeweiligen Namen voran. Aufgrund der Serialisierung der Nachrichten durch den Chat-Server erhalten alle anwenderseitigen Dokumentmethoden die Beiträge in der gleichen Reihenfolge.

### 6.3.4. Ein Container-Dokument

In den vorangegangenen Abschnitten wurde eine Reihe von Dokumenten vorgestellt, die unterschiedlichste Funktionen besitzen. Diese Dokumente können kombiniert werden, um z. B. ein multimediales Dokument zur Wissensvermittlung zu realisieren, das gleichzeitig die Möglichkeit zur Interaktion der Beteiligten oder zur Lernkontrolle bietet. Die Integration unterschiedlicher Dokumente in einem Dokument kann z. B. durch die Kombination und Reimplementierung der entsprechenden Dokumentmethoden vorgenommen werden.

In diesem Abschnitt wird ein Dokument vorgestellt, das eine Alternative zur Integration bietet, die keine Reimplementierung von Dokumentmethoden erfordert. Dieses

**Abbildung 6.4.** Präsentation von Chat-Dokumenten




Dokument basiert auf der in Abschnitt 4.4.1 beschriebenen Aggregation von Dokumenten.

Das Dokument zur Aggregation, im folgenden als Container-Dokument bezeichnet, erlaubt die Aggregation existierender Dokumente, indem es die orthogonalen Present- und Describe-Operationen auf die Ausführung der Present- und Describe-Operationen der enthaltenen Dokumente abbildet. Das in der vorliegenden Arbeit implementierte Container-Dokument referenziert die in ihm enthaltenen Dokumente anhand ihrer eindeutigen Dokumentnamen. Das Metadokument eines solchen Container-Dokuments, ist ausschnittsweise in Listing 6.10 dargestellt. Jede Zeile des Inhalts des Container-Dokuments enthält den systemweit eindeutigen Namen eines Dokuments, der hier in der Form *Server-Adresse/lokal eindeutiger Dokumentname* notiert wird.

**Listing 6.10** Metadokument eines Container-Dokuments

```
      ⋮

Content-Type: application/x-metadoc-methods

Present http://141.2.14.28/container/present.zip application/x-python
Describe http://141.2.14.28/container/describe.zip application/x-python
--ekidpUnhoJ
Content-Type: application/x-metadoc-content

dbib.uni-frankfurt.de:7333/2420-989830249-8597
dbib.uni-frankfurt.de:7333/2439-989830273-9697
--ekidpUnhoJ--
```

Um die Aggregation der einzelnen Dokumente vorzunehmen, müssen die Present- und Describe-Operationen geeignet implementiert werden. Die Present-Operation, die als Argument den Ziel-Server der Präsentation erhält, initiiert die Ausführung der Present-Operation an jedem enthaltenen Dokument, wobei sie jeweils den Ziel-Server als Argument übergibt. Die der Present-Operation zugeordnete Methode ist in Listing 6.11 dargestellt.

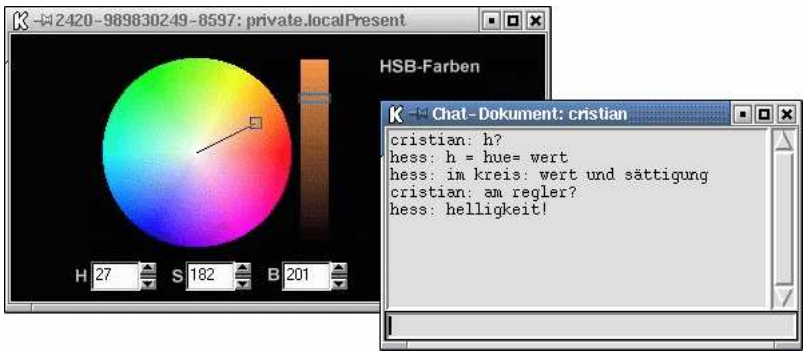
Die Ausführung der Present-Operation des Container-Dokuments führt dazu, daß die referenzierten Dokumente, im vorliegenden Beispiel handelt es sich dabei um ein Bild-

**Listing 6.11** Die Present-Methode des Container-Dokuments

```
# Present.py

def main(argv):
    import document, execserver
    import string

    # An jedem referenzierten Dokument Present initiieren.
    for line in document.open().readlines():

        # Im Dokumentinhalt gespeicherte Referenzen bearbeiten.
        server, doc = string.split(line, '/')
        host, port  = string.split(host, ':')

        # Die Adresse des Ziel-Servers (in argv) übergeben.
        execserver.invoke(host, port, doc, 'Present', argv)
    return

# Die Ausführung der Methode begint hier, als Argumente werden
# der Name des Dokuments, die IP-Adresse und Portnummer des
# Ziel-Servers übergeben.
if __name__ == '__main__':
    import sys
    sys.exit(main(sys.argv))
```

dokument und ein Chat-Dokument, auf dem Ziel-Server dargestellt werden. Wie diese Präsentation aussehen kann, ist in Abbildung 6.5 dargestellt.

**Abbildung 6.5.** Präsentation eines Container-Dokuments




Die Realisierung der Describe-Operation des Container-Dokuments entspricht der Realisierung der Present-Operation. Die der Describe-Operation zugeordnete Methode führt die Describe-Operationen der enthaltenen Dokumente aus und vereinigt die Ergebnisse zu einem Gesamtergebnis. Ausschnitte aus der der Describe-Operation zugeordneten Methode sind in Listing 6.12 dargestellt.

Die Operationen des Container-Dokuments sind generisch, da sie nur über orthogonale Operationen auf die referenzierten Dokumente zugreifen. Daher kann das Container-Dokument allein durch Änderung des Dokumentinhalts im Metadokument zur Aggregation beliebiger Dokumente genutzt werden.

**Listing 6.12** Die Describe-Methode des Container-Dokuments

```
# Describe.py

def main(argv):
    import document, execserver
    import string

    result = []
    # An jedem referenzierten Dokument Describe initiieren.
    for line in document.open().readlines():

        # Im Dokumentinhalt gespeicherte Referenzen bearbeiten.
        server, doc = string.split(line, '/')
        host, port  = string.split(host, ':')

        # Die Describe-Operation initiieren und die Ergebnise
        # in result sammeln.
        desc = execserver.invoke(host, port, doc, 'Describe')
        result = result + desc[1:]

    # Das Gesamtergebnis an den Aufrufer übergeben.
    print "Keywords:"
    for word in result:
        print " " + word
    print
    return
```

# 7. Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, eine verteilte Infrastruktur zu entwickeln, die die Realisierung skalierbarer erweiterbarer orthogonaler Digitaler Bibliotheken erlaubt. Dabei sollte die Skalierbarkeit sowohl hinsichtlich der Zahl der unterstützten Anwender als auch hinsichtlich der Zahl der gespeicherten Dokumente gewährleistet sowie die Erweiterbarkeit um neue Typen und um neue Dienste sichergestellt werden.

In einem ersten Schritt wurde ein Modell skalierbarer erweiterbarer orthogonaler Digitaler Bibliotheken entworfen, das die für Erweiterbarkeit und Orthogonalität notwendigen Elemente und Mechanismen identifiziert. Anhand dieses Modells erfolgte dann eine Untersuchung existierender Systeme zur Verarbeitung digitaler Dokumente im Hinblick auf ihre Eignung zur Realisierung einer skalierbaren, erweiterbaren, orthogonalen Digitalen Bibliothek. Resultat dieser Untersuchung war, daß in existierenden Systemen zur Verarbeitung digitaler Dokumente Erweiterbarkeit nur auf Kosten der Orthogonalität oder Skalierbarkeit unterstützt wird. Als Grund dafür wurde eine mangelnde Unterstützung der transparenten Erweiterung und Interpretation der Zuordnungsfunktion durch diese Systeme erkannt. Die Ursache dieses Mangels ist die unzureichende Benennung der Elemente der Zuordnungsfunktionen in den existierenden Systemen.

Um eine Infrastruktur für Digitale Bibliotheken zu entwickeln, die die genannten Anforderungen erfüllt, wurden drei Maßnahmen getroffen: die Einführung einer systemweit eindeutigen Benennung der Elemente der Zuordnungsfunktion, der Entwurf eines Mechanismus zur transparenten Verteilung der Zuordnungsfunktion in der Digitalen Bibliothek und die Entwicklung eines Mechanismus zur transparenten Bereitstellung von Dokumentmethoden in den, an der Digitalen Bibliothek beteiligten Rechnerknoten.

Die eindeutige Benennung wurde durch die Definition orthogonaler Operationen ermöglicht. Die Verteilung der Zuordnungsfunktion in der Digitalen Bibliothek konnte durch die Einführung von Metadokumenten erreicht werden. Das Konzept der Metadokumente basiert auf der Erkenntnis, daß die Komponenten der Digitalen Bibliothek nur die Teile der Zuordnungsfunktion benötigen, die sich auf die Dokumente beziehen, die sie bearbeiten. Diese dokumentspezifischen Teile der Zuordnungsfunktion erhält man durch Partitionieren der Zuordnungsfunktion entlang der Dimension der Dokumente. Die dokumentspezifischen Zuordnungsfunktionen werden dann zusammen mit dem Dokumentinhalt in Form eines Metadokuments zusammengefaßt. Aufgrund des Verzichts auf eine Typabbildung ist in jedem Metadokument die vollständige dokumentspezifische Zuordnungsfunktion gespeichert. Die Verteilung der Zuordnungsfunktion in der Digitalen Bibliothek ist damit allein durch den Transport des Dokumentinhalts in Form der Metadokumente möglich geworden.

Die transparente Bereitstellung der Dokumentmethoden konnte durch Verwendung von mobilen Programmen zur Implementierung von Dokumentmethoden erreicht werden. Digitale Bibliotheken lassen sich so durch Erstellung eines entsprechenden Metadokuments durch den Dokumentautor transparent um neue Dokumenttypen erweitern.

Es wurde gezeigt, wie auf der Basis dieser Infrastruktur eine Vielzahl verschiedener Dokumenttypen realisiert werden können. Dazu zählen Dokumente, die unterschiedliche Formen der Präsentation realisieren, sowie Dokumente zur verteilten Datenhaltung, zur Aggregation von Dokumenten und zur Realisierung zugriffsgeschützter und vertraulicher Dokumente. Die Erweiterung um neue Dienste wurde durch die Definition mobiler Dokumente ermöglicht, die die Verteilung neuer Dienstfunktionen innerhalb der Digitalen Bibliothek erlauben. Mobile Dokumente können, analog zu nicht mobilen Dokumenten, durch den Autor des Dokuments, in diesem Fall den Gestalter des Dienstes, transparent in die Digitale Bibliothek integriert werden. Zusammen mit der Möglichkeit zur Einführung neuer orthogonaler Operationen läßt sich dadurch das Dienstspektrum der Digitalen Bibliothek dynamisch erweitern.

Die Elemente der Infrastruktur wurden unter der Verwendung standardisierter Protokolle und existierender Laufzeitumgebungen für interpretierte Sprachen realisiert. Auf der Basis dieser Realisierung wurden verschiedene Dokumente implementiert, anhand derer die Umsetzbarkeit der entwickelten Konzepte demonstriert werden konnte. Der Einsatz plattformunabhängiger Sprachen zur Implementierung von Dokumentmethoden ermöglicht eine Integration zukünftiger Plattformen in die Infrastruktur, ohne daß dazu eine Änderung der existierenden Dokumente und Methoden notwendig wird.

In dieser Arbeit wurde eine Infrastruktur entworfen, auf deren Grundlage sich skalierbare erweiterbare orthogonale Digitale Bibliotheken realisieren lassen. Das resultierende System läßt sich durch die Dokumentautoren und Dienstgestalter transparent um neue Dokumenttypen und Dienste erweitern. Durch die konsequente Vermeidung zentraler Komponenten konnte die Skalierbarkeit des Systems in der Zahl der unterstützten Anwender sowie in der Zahl der verwalteten Dokumente sichergestellt werden.

Ausgehend von den in dieser Arbeit entwickelten Konzepten können weitergehende Fragestellungen diskutiert werden. So kann die Möglichkeit zur einer engeren Integration der Präsentation aggregierter multimedialer Dokumente, wie sie z. B. im Informedia-Projekt bei der synchronisierten Darstellung geographischer Regionen und darauf bezogener Video-Daten vorgenommen wird (vgl. [13]), untersucht werden. Eine Integration unterschiedlicher Dokumente im Präsentationsraum könnte durch die Definition einer orthogonalen MultimediaPresent-Operation geschehen, die die Angabe von Koordinaten im Dokument- und Präsentationsraum, wie sie z. B. in HyTime [64] möglich ist, zur Kontrolle der Präsentation erlaubt.

In der vorliegenden Arbeit wurde der Schutz einzelner Ausführungs-Server gegen böswillige Dokumentmethoden behandelt. Mit der Möglichkeit zur Erstellung mobiler Dokumente verdient der Schutz des Server-Verbundes zur Begrenzung der Ressourcennutzung durch einen Initiator ebenfalls eine eingehendere Betrachtung. Hier könnten Konzepte aus Infrastrukturen für mobile Agenten, z. B. AgentTcl [42], angepaßt werden, z. B. die Kontingentierung der Ressourcennutzung auf den Rechnerknoten innerhalb einer administrativen Domäne und die Verwendung elektronischen

Geldes zur Limitierung der Ressourcennutzung durch mobile Dokumente, die sich zwischen mehreren administrativen Domänen bewegen.

Zur Effizienzsteigerung könnten Verfahren zur Übersetzung von plattformunabhängigem Zwischencode in nativen Code der Zielmaschine, wie sie beispielsweise in [33] beschrieben sind, eingesetzt werden. In diesem Zusammenhang sind geeignete Mittel für eine Durchsetzung der Sicherheitsanforderungen auszuwählen und ihr Einfluß auf den zu erwartenden Performance-Gewinn zu untersuchen.

# Literaturverzeichnis


[1] ADL-TABATABAI, G. LANGDALE, S. LUCCO und R. WAHBE: *Efficient and language-indipendent mobile programs*. In: *Proceedings of the SIGPLAN'96 Conference on Programming Language Design and Implementation*, Seiten 127–136, Mai 1996.

[2] AWERBUCH, BARUCH: *A new distributed depth-first-search algorithm*. Information Processing Letters, 20(3):147–150, April 1985.

[3] BERNERS-LEE, T., R. FIELDING, UC IRVINE und H. FRYSTYK: *Hypertext Transfer Protocol — HTTP/1.0*. <url:ftp://ftp.denic.de/pub/rfc/rfc1945.txt> [7.12.1999], 1996. RFC 1945.

[4] BERNERS-LEE, T., R. FIELDING und L. MASINTER: *Uniform Resource Identifiers (URI): Generic Syntax*. <url:ftp://ftp.denic.de/pub/rfc/rfc2396.txt> [13.1.2001], 1998. RFC 2396.

[5] BERNERS-LEE, T., R.T. FIELDING und H. FRYSTYK NIELSEN: *Uniform Resource Locators (URL)*. <url:ftp://ftp.denic.de/pub/rfc/rfc1738.txt> [7.12.1999], 1994. RFC 1738.

[6] BIRMINGHAM, WILLIAM P.: *An Agent-Based Architecture for Digital Libraries*. D-Lib magazine, Juli 1995. <urn:hdl:cnri.dlib/july95-birmingham>.

[7] BIRMINGHAM, WILLIAM P., KAREN M. DRABENSTOTT, CAROLYN O. FROST, AMY J. WARNER und KATHERINE WILLIS: *The University of Michigan Digital Library: This Is Not Your Father's Library*. In: *Proceedings of the first Annual Conference on the Theory and Practice of Digital Libraries*, Seiten 53–60, College Station, Texas, Juni 1994.

[8] BORENSTEIN, NATHALIE S. und N. FREED: *MIME (Multipurpose Internet Mail Extensions)*. <url:ftp://ftp.denic.de/pub/rfc/rfc1521.txt> [7.12.1999], 1993. RFC 1521.

[9] BOWMAN, C.M., PETER B. DANZIG, DARREN R. HARDY, UDI MANBER und MICHAEL F. SCHWARTZ: *The Harvest information discovery and access system*. Computer Networks and ISDN Systems, 28(1-2):119–125, Dezember 1995.

[10] CASE, J., K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *Management Information Base for Version 2 of the Simple Network Management Protocol*

*(SNMPv2)*. <url:ftp://ftp.denic.de/pub/rfc/rfc1907.txt> [7.12.1999], Januar 1996. RFC 1907.

[11] CHANG, ERNEST J. H.: *Echo Algorithms: Depth Parallel Operations on General Graphs*. IEEE Transactions on Software Engineering, 8(4):391–401, 1982.

[12] CHESS, DAVID, BENJAMIN GROSOF, COLIN HARRISON, DAVID LEVINE, COLIN PARRIS und GENE TSUDIK: *Itinerant Agents for Mobile Computing*. IEEE Personal Communications, 2(5):34–49, Oktober 1995.

[13] CHRISTEL, MICHAEL G., ANDREAS M. OLLIGSCHLAEGER und CHANG HUANG: *Interactive Maps for a Digital Video Library*. IEEE Multimedia, 7(1):60–67, Januar–März 2000.

[14] CIDON, ISREAL: *Yet another distributed depth-first-search algorithm*. Information Processing Letters, 26(6):301–305, 1988.

[15] COUSINS, STEVE B.: *Reification and Affordances in a User Interface for Interavting with Heterogeneous Distributed Applications*. Doktorarbeit, Department of Computer Science, Stanford University, 1997.

[16] COUSINS, STEVE B., ANDREAS PAEPCKE, TERRY WINOGRAD, ERIC A. BIER und KEN PIER: *The Digital Library Integrated Task Environment (DLITE)*. In: *Proceedings of the Second ACM International Conference on Digital Libraries*, Seiten 142–151, Juli 1997.

[17] CROCKER, DAVID H.: *Standard for the format of ARPA Internet text messages*. <url:ftp://ftp.denic.de/pub/rfc/rfc822.txt> [7.12.1999], 1982. RFC 822.

[18] DAIGLE, L., P. DEUTSCH, B. HEELAN, C. ALPAUGH und M. MACLACHLAN: *Uniform Resource Agents (URAs)*. <url:ftp://ftp.denic.de/pub/rfc/rfc2016.txt> [7.12.1999], 1996. RFC 2016.

[19] DAVIS, J., D. KRAFFT und C. LAGOZE: *Dienst: Building a Production Technical Report Server*. In: *Proceedings of the IEEE Advances in Digital Libraries ADL '95*, Seiten 211–222, McClean, Virginia, USA, Mai 1995. Springer-Verlag.

[20] DAVIS, JAMES R.: *Creating a Networked Computer Science Technical Report Library*. D-Lib Magazine, September 1995. <urn:hdl://cnri.dlib/september95-davis>.

[21] DAVIS, JAMES R.: *Creating a Networked Computer Science Technical Report Library*. In: *Proceedings of the 1st ACM Internation Conference on Digital Libraries DL '96*, Seiten 177–178, Bethesda, Maryland, USA, März 1996.

[22] DJERABA, CHABANE, NARINETTE BOUET, HENRI BRIAND und ALI KHENCHAF: *Visual and textual content based indexing and retrieval*. International Journal on Digital Libraries, (2):269–287, 2000. Springer.

[23] ENCARNACAO, J., M. FRÜHAUF und T. KIRSTE: *Mobile Visualization: Challenges and Solution Concepts*. In: *Proceedings of IFIP Conference on Computer Applications in Production and Engineering CAPE'95*, Seiten 16–18, Beijing, China, Mai 1995.

[24] ENDRES, ALBERT und DIETER W. FELLNER: *Digitale Bibliotheken — Informatik-Lösungen für globale Wissensmärkte*. dpunkt.verlag, Heidelberg, 2000.

[25] EU-NFS DIGITAL LIBRARY COLLABORATIVE WORKING GROUPS: *Resource Discovery in a Globally-Distributed Digital Library*, 1998. <url:http://www.iei.pi.cnr.it/DELOS/NSF/resourcediscovery.htm> [18.10.2000].

[26] EU-NFS DIGITAL LIBRARY WORKING GROUP ON INTEROPERABILITY BETWEEN DIGITAL LIBRARIES: *Position Paper*, 1998. <url:http://www.iei.pi.cnr.it/DELOS/NSF/interop.htm> [18.10.2000].

[27] EU-NFS WORKING GROUPS ON FUTURE DIRECTIONS FOR DIGITAL LIBRARIES RESEARCH: *Summary Report*, 1998. <url:http://www.iei.pi.cnr.it/DELOS/NSF/Brussrep.htm> [18.10.2000].

[28] FARMER, WILLIAM M., JOSHUA D. GUTTMAN und VIPIN SWARUP: *Security for Mobile Agents: Issues and Requirements*. In: *Proceedings of the 19th National Information Systems Security Conference (NISSC'96)*, Seiten 591–597, Baltimore, USA, Oktober 1996.

[29] FATEMI, N. und P. MULHEM: *A Conceptual Graph Approach for Video Data Represenantion and Retrieval*. In: *Proceedings of the third Symposium on Intelligent Data Analysis (IDA'99)*, Nummer 1642 in *Lecture Notes in Computer Science*, Seiten 525–534, Amsterdam, August 1999. Springer.

[30] FIELDING, R., J. GETTYS, J. MOGUL, H. FRYSTYK, L. MASINTER, P. LEACH und T. BERNERS-LEE: *Hypertext Transfer Protocol — HTTP/1.1*. <url:ftp://ftp.denic.de/pub/rfc/rfc2616.txt> [13.01.2001], Juni 1999. RFC 2616.

[31] FLICKNER, MARION, HARPREET SAWHNEY, WAYNE NIBLACK, JONATHAN ASHLEY, QIAN HUANG, BYRON DOM, MONIKA GORKANI, JIM HAFNER, DENIS LEE, DRAGUTIN PETKOVIC, DAVID STEELE und PETER YANKER: *Query by Image and Video Content: The QBIC System*. IEEE Computer, 28(9):23–32, September 1995.

[32] FRANKS, J., P. HALLAM-BAKER, J. HOSTETLER, S. LAWRENCE, P. LEACH, A. LUOTONEN und L. STEWART: *HTTP Authentication: Basic and Digest Access Authentication*. <url:ftp://ftp.denic.de/pub/rfc/rfc2617.txt> [13.01.2001], Juni 1999. RFC 2617.

[33] FRANZ, MICHAEL und THOMAS KISTLER: *Slim Binaries*. Communications of the ACM, 40(12):87–94, Dezember 1997.

[34] FREED, N. und N. BORENSTEIN: *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. <url:ftp://ftp.denic.de/pub/rfc/rfc2045.txt> [7.12.1999], November 1996. RFC 2045.

[35] FREED, N. und N. BORENSTEIN: *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. <url:ftp://ftp.denic.de/pub/rfc/rfc2046.txt> [7.12.1999], November 1996. RFC 2046.

[36] FREIER, ALAN O., PHILIP KARLTON und PAUL C. KOCHER: *The SSL Protocol — Version 3.0*. <url:http://home.netscape.com/eng/ssl3/draft302.txt> [14.4.2000].

[37] GEIER, MARTIN, MARTIN STECKERMEIER, ULRICH BECKER, FRANZ J. HAUCK, ERICH MEIER und UWE RASTOFER: *Support for mobility and replication in the AspectIX architecture*. Technischer Bericht TR-I4-08-05, IMMD IV, Universität Erlangen-Nürnberg, 1998.

[38] GHOSH, HIRANMAY, SANTANU CHAUDHURY, CHETAN ARORA und PARAMJET NIRANKARI: *ImAge: an extensible agent-based architecture for image retrieval*. International Journal on Digital Libraries, 2(4):236–250, 2000. Springer.

[39] GLADNEY, HENRY M., EDWARD A. FOX, ZAHID AMED, RON ASHANY, NICHOLAS J. BELKIN und MARIA ZEMANKOVA: *Digital library: Gross Structure and Requirements: Report from a March 1994 Workshop*. In: *Proceedings of the First Annual Conference on the Theory and Practice of Digital Libraries*, Seiten 101–107, College Station, Texas, USA, Juni 1994.

[40] GOLDBERG, A. und D. ROBSON: *Smalltalk–80: The language and its implementation*. Addison-Wesley, 1983.

[41] GOSLING, JAMES und HENRY MCGILTON: *The Java Language Environment — A White Paper*. <url:http://java.sun.com/docs/white/langenv/> [23.1.2001], Mai 1996.

[42] GRAY, ROBERT S.: *Agent Tcl: A flexible and secure mobile-agent system*. Doktorarbeit, Dartmouth College, Hanover, New Hampshire, Juni 1997.

[43] INTERNATIONAL STANDARDS ORGANISATION: *Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language (VRML — Part 1: Functional specification and UTF-8 encoding)*, Oktober 2000.

[44] KAHN, ROBERT und ROBERT WILENSKY: *A Framework for Distributed Digital Object Services*, Mai 1995. <urn:hdl:cnri.dlib/tn95-01>.

[45] KIRSTE, T.: *An infrastructure for Mobile Information Systems based on a Fragmented Object Model*. Distributed Systems Engineering Journal, 2(3):161–170, 1995.

[46] LAGOZE, CARL und JIM DAVIS: *Dienst: An Architecture for Distributed Document Libraries*. Communications of the ACM, 38(4):47, April 1995.

[47] LAGOZE, CARL und SANDRA PAYETTE: *An Infrastructure for Open-Architecture Digital Libraries*. Technischer Bericht TR98-1690, Cornell University, Juni 1998.

[48] LARKIN, DON und GREG WILSON: *Object-Oriented Programming and the Objective-C Language*. Next Software Inc., 1995. <url:http://www.gnustep.org/resources/documentation/ObjectivCBook.pdf> [18.3.2000].

[49] LESK, MICHAEL: *Practical Digital Libraries — Books, Bytes & Bucks*. Morgan Kaufmann, 1997.

[50] LINGNAU, A., P. DÖMEL und O. DROBNIK: *A HTTP-based Infrastructure for Mobile Agents*. In: *Proceedings of the 4th International World Wide Web Conference*, Boston, MA, USA, Dezember 1995.

[51] LYNCH, NANCY A.: *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.

[52] MAKPANGOU, MESAAC, YVON GOURHANT, JEAN-PIERRE LE NARZUL und MARC SHAPIRO: *Fragmented objects for distributed abstractions*. In: CASAVANT, T. L. und SINGHAL M. (Herausgeber): *Readings in distributed computing systems*, Seiten 170–186. IEEE Computer Society Press, 1994.

[53] MALY, KURT, MOHAMMAD ZUBAIR, STEWART N. T. SHEN und MICHAEL L. NELSON: *Generalizing an Existing Digital Library*. Technischer Bericht TR_99_01, Department of Computer Science, Old Dominion University and NASA Langley Research Center, Februar 1999.

[54] MARSHALL, CATHRINE C., GENE GOLOVCHINSKY und MORGAN N. PRICE: *Digital Libraries and Mobility*. Communications of the ACM, 44(5):55–56, 2001.

[55] MCNAB, RODGER J., LLOYD A. SMITH, IAN H. WITTEN, CLARE L. HENDERSON und SALLY JO CUNNINGHAM: *Towards the Digital Music Library: Tune Retrieval from Acoustic Input*. In: *Proceedings of the 1st ACM International Conference on Digital Libraries DL '96*, Seiten 11–18, Bethesda, Maryland, USA, März 1996.

[56] MEMON, NASIR und PING WAH WONG: *Protecting Digital Media Content*. Communications of the ACM, 41(7):35–43, Juli 1998.

[57] MÖNCH, CHRISTIAN: *INDIGO — An Approach to Infrastructures for Digital Libraries*. In: BORBINHA, JOSÉ und THOMAS BAKER (Herausgeber): *Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries, ECDL 2000*, Nummer 1923 in *Lecture Notes in Computer Science*, Seiten 158–167, Lissabon, Portugal, September 2000. Springer-Verlag.

[58] MÖNCH, CHRISTIAN und OSWALD DROBNIK: *Integrating New Document Types into Digital Libraries*. In: *Proceedings of the IEEE Forum on Research and Technology Advances in Digital Libraries IEEE ADL '98*, Seiten 56–65. IEEE, April 1998.

[59] MOCKAPETRIS, P.: *Domain Names — Concepts and Facilities*. <url:ftp://ftp.denic.de/pub/rfc/rfc1034.txt> [31.3.2000], 1987. RFC 1034.

[60] NATIONAL SCIENCE FOUNDATION: *Digital Libraries Initiative — Phase 2*. <url:http://www.nsf.gov/pubs/1998/nsf9863/nsf9863.htm>, 1998. Announcement Number NSF 98–63.

[61] NELSON, MICHAEL L.: *Buckets: Smart Objects for Digital Libraries*. Doktorarbeit, Old Dominion University, August 2000.

[62] NELSON, MICHAEL L. und KURT MALY: *Buckets: Smart Objects for Digital Libraries*. Communications of the ACM, 44(5):60–63, 2001.

[63] NELSON, MICHAEL L., KURT MALY, STEWART N. T. SHEN und MOHAMMAD ZUBAIR: *NCSTRL+: Adding Multi-Discipline and Multi-Genre Support to the Dienst Protocol Using Clusters and Buckets*. In: *Proceedings of Advances in Digital Libraries 98 (ADL'98)*, Seiten 128–136, Santa Barbara, CA, April 1998. IEEE.

[64] NEWCOMB, STEVEN R., NEILL A. KIPP und VICTORIA T. NEWCOMB: *The "HyTime" hypermedia/time-based document structuring language*. Communications of the ACM, 34(11):67–83, November 1991.

[65] NÜRNBERG, PETER J., RICHARD FURUTA, JOHN L. LEGGETT, CATHERINE C. MARSHALL und FRANK M. SHIPMANN III: *Digital Libraries: Issues and Architectures*. In: *Proceedings of The Second International Conference on the Theory and Practice of Digital Libraries*, Seiten 147–153, Juni 1995.

[66] OUNIS, IADH und MARIUS PASCA: *Modeling, Indexing and Retrieving Images Using Conceptual Graphs*, August 1998.

[67] PAEPCKE, ANDREAS, MICHELLE BALDONADO, CHEN-CHUAN K. CHANG, STEVE COUSINS HECTOR und GARCIA-MOLINA: *Using distributed objects to build the Stanford digital library Infobus*. IEEE Computer, 32(2), Februar 1999.

[68] PAEPCKE, ANDREAS, CHEN-CHUAN K. CHANG, HÉCTOR GARCÍA-MOLINA und TERRY WINOGRAD: *Interoperability for Digital Libraries Worldwide*. Communications of the ACM, 41(4):33–43, April 1998.

[69] PAEPCKE, ANDREAS, STEVE B. COUSINS, HECTOR GARCIA-MOLINA, SCOTT W. HASSAN, STEVEN P. KETCHPEL, MARTIN RÖSCHEISEN und TERRY WINOGRAD: *Using Distributed Objects for Digital Library Interoperability*. IEEE Computer, 29(5):61–68, Mai 1996.

[70] PAYETTE, SANDRA und CARL LAGOZE: *Flexible and Extensible Digital Object and Repository Architecture (FEDORA)*. In: NIKOLAOU, CHRISTOS und CONSTANTINE STEPHANIDIS (Herausgeber): *Proceedings of the 2th European Conference on Research and Advanced Technology for Digital Libraries, ECDL'98*, Nummer 1513 in *Lecture Notes in Computer Science*, Seiten 41–59, Heraklion, Griechenland, September 1998. Springer-Verlag.

[71] PAYETTE, SANRA und CARL LAGOZE: *Policy-Carrying, Policy-Enforcing Digital Objects*. In: BORBINHA, JOSÉ und THOMAS BAKER (Herausgeber): *Proceedings of the 4th European Conference on Research and Advanced Technology for Digital Libraries, ECDL 2000*, Nummer 1923 in *Lecture Notes in Computer Science*, Seiten 144–157, Lissabon, PT, September 2000. Springer-Verlag.

[72] PHELPS, THOMAS A.: *Multivalent Documents: Anytime, Anywhere, Any Type, Every Way User-Improvable Digital Documents and Systems*. Doktorarbeit, University of California, Berkeley, 1998.

[73] PHELPS, THOMAS A. und ROBERT WILENSKY: *Multivalent Documents: A New Model for Digital Documents*. Technischer Bericht CSD-98-999, University of California Berkeley, Department of Computer Science, 1998.

[74] PHELPS, THOMAS A. und ROBERT WILENSKY: *Multivalent documents*. Communications of the ACM, 43(6):82–90, 2000.

[75] PORDESCH, ULRICH: *Nachweis der Präsentation signierter Daten*. Technischer Bericht 68, GMD, November 1999.

[76] RAMSDELL, B.: *S/MIME Version 3 Message Specification*. <url:ftp://ftp.denic.de/pub/rfc/rfc2633.txt> [13.1.2001], 1999. RFC 2633.

[77] RESCORLA, E. und A. SCHIFFMAN: *The Secure HyperText Tranfer Protocol*. <url:ftp://ftp.denic.de/pub/rfc/rfc2660.txt> [13.1.2001], August 1999. RFC 2660.

[78] ROSSUM, GUIDO VAN und FRED L. DRAKE JR.: *Python Library Reference — Release 2.1*. <url:http://www.python.org/ftp/python/doc/2.1/postscript-a4-2.1.tar.bz2%> [26.4.2001], April 2001.

[79] ROSSUM, GUIDO VAN und FRED L. DRAKE JR.: *Python Reference Manual — Release 2.1*. <url:http://www.python.org/ftp/python/doc/2.1/postscript-a4-2.1.tar.bz2%> [26.4.2001], April 2001.

[80] RÖSCHEISEN, MARTIN, MICHELLE BALDONADO, KEVIN CHANG, LUIS GRAVANO, STEVEN KETCHPEL und ANDREAS PAEPCKE: *The Stanford InfoBus and Its Service Layers: Augmenting the Internet with Higher-Level Information Management Protocols by Martin*. In: BARTH, A., M. BREU, A. ENDRES und A. DE KEMP (Herausgeber): *Digital Libraries in Computer Science: The MeDoc Approach*, Nummer 1392 in *Lecture Notes in Computer Science*, Seiten 213–230. Springer Verlag, 1998.

[81] RÖTTCHER, GÜNTER, KLAUS-PETER BÖTTGER und URSULA ANKERSTEIN: *Basiskenntnis Bibliothek*. Bock + Herchen, 3., überarbeitete und aktualisierte Auflage, 1995.

[82] RUMBAUGH, JAMES, MICHAEL BLAHA, WILLIAM PREMERLANI, FREDERICK EDDY und WILLIAM LORENSEN: *Object-Oriented Modelling and Design*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[83] SCHATZ, BRUCE und HSINCHUN CHEN: *Building Large–Scale Digital Libraries*. IEEE Computer, 29(5):22–26, Mai 1996.

[84] SCHULZRINNE, H., S. CASNER, R. FREDERICK und V. JACOBSON: *RTP: A Transport Protocol for Real-Time Applications*. <url:ftp://ftp.denic.de/pub/rfc/rfc1889.txt> [7.12.1999], Januar 1996. RFC 1889.

[85] SHAPIRO, MARC, YVON GOURHANT, SABINE HABERT, LAURENCE MOSSERI, MICHEL RUFFIN und C´ELINE VALOT: *SOS: An object-oriented operating system — assessment and perspectives*. Computing Systems, 2(4):287–338, Dezember 1989.

[86] STEEN, MAARTEN VAN, PHILIP HOMBURG und ANDREW S. TANENBAUM: *Globe: A Wide-Area Distributed System*. IEEE Concurrency, Seiten 70–78, Januar–März 1999.

[87] STEEN, M. VAN, A.S. TANENBAUM, I. KUZ und H.J. SIPS: *A Scalable Middleware Solution for Advanced Wide-Area Web Services*. Distributed Systems Engineering, 6(1):32–42, März 1999.

[88] STEINMETZ, RALF: *Multimedia Technologie — Einführung und Grundlagen*. Springer-Verlag, 1995. 1. korrigierter Nachruck.

[89] STRENSCH, MARTIN: *Entwicklung eines Systems für evolutionäre Dokumente als Teil einer Digitalen Bibliothek*. Diplomarbeit, J. W. Goethe-Universität, Frankfurt, Oktober 1999.

[90] STROUSTRUP, BJARNE: *Die C++ Programmiersprache*. Addison-Wesley Publishing Company, 3. überarbeitete Auflage, 1992.

[91] SUN MICROSYSTEMS: *Java(TM) Media Framework API Guide*. <url:http://java.sun.com/products/java-media/jmf/2.1.1/guide/> [23.1.2001], November 1999.

[92] TANNEN, VAL: *Heterogenous Data Integration with Mobile Information Managers*. In: JAJODIA, SUSHIL, M. TAMER ÖZSU und ASUMAN DOGAC (Herausgeber): *Proceedings of Advances in Multimedia Information Systems, MIS'98*, Nummer 1508 in *Lecture Notes in Computer Science*, Seiten 2–3. Springer, 1998.

[93] TEL, GERARD: *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[94] THE URN IMPLEMENTORS: *Uniform Resource Names: A Progress Report*. D-Lib Magazine, Februar 1996. <urn:hdl:cnri.dlib/february96-urn_implementors>.

[95] UNGAR, DAVID und RANDALL B. SMITH: *Self: The Power of Simplicity*. Lisp and Symbolic Computation: An International Journal, 4(3):187–205, 1991.

[96] W3 CONSORTIUM: *HTML 4.01 Specification*, Dezember 1999. <url:http://www.w3.org/TR/html4/> [15.1.2001].

[97] W3 CONSORTIUM: *Extensible Markup Language (XML) 1.0 (Second Edition)*, Oktober 2000. <url:http://www.w3.org/TR/REC-xml/> [6.10.2000].

[98] W3 CONSORTIUM: *Scalable Vector Graphics (SVG) 1.0 Specification — W3C Candidate Recommendation*, November 2000. <url:http://www.w3.org/TR/2000/CR-SVG-20001102/> [1.3.2001].

[99] W3 CONSORTIUM: *XML Linking Language (XLink) Version 1.0*, Dezember 2000. <url:http://www.w3.org/TR/xlink/> [13.1.2001].

[100] W3 CONSORTIUM: *Mathematical Markup Language (MathML) Version 2.0*, Januar 2001. <url:http://www.w3.org/TR/MathML2/> [15.1.2001].

[101] W3 CONSORTIUM: *Synchronized Multimedia Integration Language (SMIL 2.0) Specification — W3c Working Draft*, März 2001. <url:http://www.w3.org/TR/2001/WD-smil20-20010301/> [10.4.2001].

[102] WACTLAR, HOWARD W., TAKEO KANADE und SCOTT M. STEVENS MICHAEL A. SMITH: *Intelligent Access to Digital Video: Informedia Project*. IEEE Computer, 29(5):46–52, Mai 1996.

[103] WALDO, JIM: *The Jini architecture for network-centric computing*. Communications of the ACM, 42(7):76–82, Juli 1999.

[104] WEB3D CONSORTIUM: *Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language (VRML) — Part 4: XML encoding (X3D)*, 1999. <url:http://www.web3d.org/TaskGroups/x3d/specification/part4/Part4Index%.html> [16.12.1999].

[105] WEINSTEIN, PETER C., WILLIAM P. BIRMINGHAM und EDMUND H. DURFEE: *Agent-Based Digital Libraries: Decentralization and Coordination*. IEEE Communications Magazin, 37(1):110–115, Januar 1999.

[106] WELLMAN, MICHAEL P., EDMUND H. DURFEE und WILLIAM P. BIRMINGHAM: *The digital library as a community of information agents*. IEEE Expert – Intelligent Systems & Their Application, 11(3):10–11, Juni 1996.

[107] YANG, YANYAN, OMER F. RANA und CHRISTOS GEORGOUSOPOULOS: *Mobile Agents and the SARA Digital Library*. In: *Proceedings of the IEEE Advances in Digital Libraries 2000 (ADL 2000)*, Seiten 71–77. IEEE, Mai 2000.

# Lebenslauf

## Persönliche Daten:

Christian Alexander Mönch
geboren am 11.4.1965 in Aachen

## Schulbesuch:

| | |
|---|---|
| 1971 – 1975 | Max-Planck-Schule in Essen |
| 1975 – 1977 | Humbold-Gymnasium in Essen |
| 1977 – 1978 | Albertus Magnus Kolleg in Königstein im Taunus |
| 1978 – 1985 | Liebigschule in Frankfurt am Main |
| Mai 1985 | Abschluß: Abitur |

## Zivildienst:

| | |
|---|---|
| 1986 – 1988 | Deutsches Rotes Kreuz, Frankfurt am Main |

## Studium:

| | |
|---|---|
| 1988 – 1995 | Studium der Informatik an der<br>Johann Wolfgang Goethe-Universität in Frankfurt am Main<br>Nebenfach: Physik, Vordiplom: 1993 |
| Mai 1995 | Abschluß: Diplom |

## Berufstätigkeit:

| | |
|---|---|
| Juni 1995 – Juni 2000 | Wissenschaftlicher Mitarbeiter am Fachbereich Informatik<br>der Johann Wolfgang Goethe-Universität in Frankfurt am Main,<br>Professur für Architektur und Betrieb Verteilter Systeme<br>bei Prof. Dr. O. Drobnik. |
| seit August 2001 | Software Entwickler bei WebAgents GmbH Bad Homburg |