Diplomarbeit

Entwurf und Spezifikation eines elektronischen Dokumentenarchives mit Workflowmechanismen unter Anwendung objektorientierter Methoden

S. Winking

Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

5. Mai 1997

Gutachter:

Prof. Dr. E.-E. Doberkat

Dr. S. Dißmann

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

KAPITEL 1

1 Einleitung

1.1 Motivation

Trotz weitreichenden Einsatzes von Computern und Datennetzen in Unternehmen, insbesondere im Büro- und Fertigungsbereich, ist es selten gelungen, die möglichen Vorteile in allen Bereichen sinnvoll umzusetzen. Eingehende Papierdokumente werden noch immer von Hand zur weiteren Verteilung kopiert, Anmerkungen auf den Originalen vermerkt, Kommentare angeheftet und weiteres mehr. Die Kopien werden zeitaufwendig mit der Hauspost verschickt oder in zentralen Verteilern vom Adressaten persönlich abgeholt. Andererseits erreichen wichtige Informationen oder Schreiben mögliche Interessenten nicht, da die Existenz der Information nicht bekannt ist. Auch andere Papierdokumente, wie Gebrauchsanleitungen oder Altdokumente, liegen meist nicht am Arbeitsplatz vor und müssen zeitaufwendig beschafft und kopiert werden.

Die Möglichkeiten zur Umwandlung von Papierdokumenten in computernutzbare Information und deren Verwaltung durch eine besondere Software, einem zu entwickelnden elektronischen Dokumentenarchiv, kann hier Abhilfe schaffen. Zum Verteilen der Information kann ein schon bestehendes Datennetz eines Unternehmens genutzt werden. Vorhandene und im Einsatz befindliche Anwendungen der einzelnen Benutzer, zum Beispiel Textverarbeitungen oder Grafikprogramme, sollen individuell integriert werden können.

Jeder Benutzer soll in wesentlichen Punkten seiner Arbeit unterstützt, aber nicht eingeschränkt werden. Natürliche Abläufe sollen nachmodelliert werden. Der Verzicht auf Papierkopien und andere papierbasierte Aufzeichnungen soll keine nachteiligen Auswirkungen auf die Produktivität und Kreativität haben. Arbeitsabläufe sollen beschleunigt werden. Eine finanzielle Ersparnis wird durch die Reduzierung der Anzahl der Papierkopien erreicht. Durch die automatische Verwaltung von Bearbeitungsergebnissen soll eine unternehmensweite Konsistenz der Daten ermöglicht werden.

Sich wiederholende Bearbeitungsvorgänge können durch den Einsatz von Workflowmechanismen vereinfacht werden. Der Benutzer soll die Möglichkeit haben, den Weg eines Dokumentes durch das Unternehmen selbst auf einfache Weise festzulegen. Es sollen Möglichkeiten für die Weiterleitung eines Dokumentes an andere Benutzer bereitgestellt werden.

Bei der Entwicklung des Systems ist insgesamt ein Schwerpunkt in der Bearbeitung von papierorientierten Dokumenten zu sehen, die zur computerunterstützten Archivierung durch einen Papiervorlagenscanner erfaßt werden. Die Unterstützung elektronischer Dokumente, wie computererstellter Texte und Grafiken, aber auch Ton- oder Videodateien, ist aber genauso vorgesehen.

Die Besonderheit liegt bei diesem System in der Integration von Datenhaltung und Datenarchivierung auf der einen und der Benutzung von Workflow- und Kommunikationselementen auf der anderen Seite. Dazu sollen Konzepte aus den Bereichen Computer Supported Cooperative Work (CSCW) und Workflow-Management auf ihre Eignung und Verwendung hin überprüft werden.

1.2 Aufbau der Arbeit

Ziel ist es, ein System zu entwerfen, welches dem Benutzer eine möglichst breite Arbeitsunterstützung und Integration der Arbeitsabläufe bietet. Nach der Analyse der Anforderungen und dem Entwurf ist im gegebenen Zeitrahmen noch eine prototypische Implementierung vorgesehen, welche die Benutzung und Funktionalität des Systems veranschaulichen soll.

Im 2. Kapitel findet eine Einführung in die Thematik der Archivsysteme durch ausgewählte Szenarien statt. Diese Szenarien zeigen Beispiele der Arbeitsabläufe in einem Büro mit Posteingang, in einem Konstruktionsbüro mit CAD-Arbeitsplätzen und in der Softwareentwicklung.

Im 3. Kapitel werden Problempunkte in den Szenarien angesprochen und vorteilhaftere Lösungen durch eine Unterstützung der Arbeitsabläufe durch Computereinsatz gezeigt.

Im 4. Kapitel werden diese Lösungen auf die Szenarien angewendet und eine Systemvision beschrieben, die diese Änderungen berücksichtigt und als Grundlage für die Anforderungen an das spätere System dient.

Im 5. Kapitel werden Dokumenten-Management-Systeme vorgestellt und es wird kurz auf Archivsysteme, Retrievalsysteme und Workflowsysteme eingegangen, um eine Abgrenzung und Einordnung des hier entwickelten Systems zu ermöglichen.

Im 6. Kapitel werden dann Archivsysteme genauer betrachtet.

Im 7. Kapitel erfolgt eine Beschreibung der Vorgehensweise bei der Systemkonstruktion.

Im 8. Kapitel findet dann die Analyse des Archivsystems statt.

Im 9. Kapitel wird die Implementierung eines Prototypen aufgrund der vorangegangenen Analyse dokumentiert werden. Der Prototyp veranschaulicht die Benutzung und Funktionalität des Systems, ohne eine vollständige Implementierung zu sein.

Im abschließenden 10. Kapitel findet dann die Bewertung der prototypischen Implementierung und der Ausblick auf ausstehende Arbeit statt.

KAPITEL 2

2 Einführung in die Anwendungsgebiete durch Szenarien

Um einen Einblick in die Anwendung von Archivsystemen und den damit zusammenhängenden Bereichen zu geben, erfolgt eine Einführung in die Anwendungsgebiete durch drei Szenarien. In den ausgewählten Szenarien werden mögliche Einsatzgebiete für Archivsysteme vorgestellt. Dadurch sollen grundlegende Aufgaben und Funktionen solcher Systeme praxisnah beschrieben werden. Außerdem werden Begriffe aus der Welt der Archivsysteme eingeführt und erläutert. Anhand der Szenarien werden dann in den weiteren Kapiteln die Anforderungen, die als Grundlage für den Entwurf des Archivsystems dienen, definiert.

Die Auswahl der Szenarien soll einen möglichst weiten Anwendungsbereich abdecken und orientiert sich an Kriterien wie Praxisnähe, Erweiterbarkeit und Vielseitigkeit. Die Auswahl umfaßt daher Beispiele aus dem kaufmännischen, dem technischen und dem an der Informationstechnik orientierten Anwendungsbereich. In den folgenden Kapiteln dieser Arbeit wird immer wieder Bezug auf diese Szenarien oder Teile dieser Szenarien genommen, um die Anforderungen an das Archivsystem verständlich zu machen. Fett geschriebene Begriffe (z.B. **„Schriftverkehr")** sind für das weitere Vorgehen bei der Systemkonstruktion von Bedeutung und werden in Kapitel 8.1.2 in dem Glossar der Problembereichsobjekte wieder aufgegriffen. Dort wird die Bedeutung der Begriffe für das Archivsystem definiert.

2.1 Büro

In **Büros** fällt eine Unmenge von **Schriftverkehr** an. Dieser besteht aus eingehender und ausgehender Post und kann weiter zum Beispiel in Rechnungen, Produktinformationen oder Bestellungen unterteilt werden. Die Arten des Schriftverkehrs können vielfältig und von Büro zu Büro je nach Aufgabengebiet unterschiedlich sein, so daß hier weitere Arten nicht aufgezählt werden.

Unter dem anfallenden Schriftverkehr gibt es Schriftverkehr, der archiviert werden soll oder muß. Veranlassung dazu sind Notwendigkeiten zu einer langfristigen Aufbewahrung. Für diese Aufbewahrung können rechtliche oder geschäftliche Gründe ausschlaggebend sein. Es gibt eine Vielzahl von rechtlichen Gründen: steuerrechtliche Gründe beispielsweise erfordern für Buchungsbelege eine **Aufbewahrungsfrist** von vier bis zehn oder mehr Jahren für eine etwaige Buchprüfung. Die vorgeschriebenen Aufbewahrungsfristen sind für die einzelnen rechtlichen Bereiche jeweils im Einzelfall festzustellen und einzuhalten. Auch ohne eine rechtlich zwingende Aufbewahrungpflicht kann es im Interesse eines Unternehmens sein, Unterlagen über einen längeren Zeitraum hin aufzubewahren. Solche geschäftlichen Gründe könnten die Aufbewahrung von Informationsmaterialien oder Projekt unterlagen betreffen.

Wie sieht heute die entsprechende Vorgehensweise zur Bearbeitung des Schriftverkehrs in einem nach herkömmlicher Weise geführten Büro aus? Der Vorgang der Bearbeitung des Schriftverkehrs kann in folgende Punkte unterteilt werden:

Weiterleitung

Die eingehende Post wird von einer **Büromitarbeiter** geöffnet und mit einem Stempel versehen, der das Eingangsdatum festhält. Je nach Art des Schriftverkehrs werden die **Dokumente** an die zuständige Sachabteilung oder an den zuständigen **Sachbearbeiter** weitergeleitet. Sachabteilungen sind beispielsweise Buchhaltung, Einkauf oder Verkauf; Sachbearbeiter sind beispielsweise **Buchhalter**, **Einkäufer** oder **Verkäufer**. Zur Verteilung der **Dokumente** werden in der Regel **Kopien** erzeugt, wenn das Dokument an mehr als einer Stelle benötigt wird. Zur Weiterverteilung wird die Post dann in Postfächern zur Abholung bereitgestellt oder über die Hauspost zugestellt. Zum Zeitpunkt der Erstellung der Kopien ist nicht bekannt, ob die verteilte Information von Interesse am Zielort sein wird, so daß die Kopie eventuell unnötigerweise erzeugt wurde. Andererseits kann es vorkommen, daß eine Information durch den fest definierten Verteiler einen möglichen Interessenten nicht erreicht, und dieser so auch nichts von der Existenz einer für ihn interessanten Information er fährt.

Ablage

Das **Originaldokument** wird in **Aktenordnern** oder anderen Ablagesystemen abgelegt. In den einzelnen Sachabteilungen oder an den einzelnen Arbeitsplätzen wird mit den **Kopien** des **Originaldokumentes** genauso verfahren. Die Unterlagen werden dabei in der Regel zuerst nach der Art des Schriftverkehrs und dann in der zeitlichen Reihenfolge des Eingangs, also chronologisch, sortiert.

Suche

Wird der Zugriff auf eine bestimmte Information benötigt, muß zuerst der **Aktenordner** gefunden werden, in dem das Schriftstück abgelegt wurde. In diesem **Aktenordner** muß dann nach dem entsprechenden Schriftstück geblättert werden. Die Schriftstücke liegen durch das Ablageverfahren meist in einer chronologischen Ordnung vor, so daß es leicht ist, eine Rechnung eines bestimmten Datums zu finden, aber die Suche nach Rechnungen eines bestimmten Lieferanten, über ein bestimmtes Produkt, eine bestimmte Summe oder eine bestimmte Kostenstelle erfordert dann viel Zeit und Geduld, da für diese Informationen durch das Ablageverfahren keine Suchunterstützung gegeben ist.

Reproduktion

Unter **Reproduktion** wird die Anfertigung einer Kopie eines **Originaldokumentes**, die der Vorlage bestmöglich gleicht, verstanden. Eine **Reproduktion** wird angefertigt, wenn Dokumente erneut benötigt werden, zum Beispiel bei einem Versand an Kunden oder zur weiteren Benutzung durch einen Sachbearbeiter. Zur Anfertigung einer **Reproduktion** wird das **Originaldokument** benötigt. Findet man das gesuchte **Originaldokument** nicht am Arbeitsplatz, so ist herauszufinden, an welcher Stelle das **Originaldokument** abgelegt ist, um dann dort danach zu suchen. Wurde das **Originaldokument** gefunden, kann zum Bearbeiten eine **Kopie** angefertigt werden.

Je nach Material der Vorlage werden dazu verschiedene Geräte benötigt. Wenn das Originaldokument in Papierform vorliegt, benötigt man zur Reproduktion eine **Kopiermaschine**, wenn das Originaldokument in Form einer **Mikrofilmkarte** vorliegt, benötigt man einen **Mikrofilmkartenrückvergrösserer**.

Archivierung

Die Lagermöglichkeiten für Schriftverkehr innerhalb eines Büros selber sind in der Regel beschränkt. Deshalb wird **Schriftverkehr**, der nicht mehr direkt im **Büro** benötigt wird, durch einen **Archivar** ausgelagert, um Platz für aktuellere Unterlagen zu schaffen. Lagerorte sind dann Keller oder Lagerhallen; die Zugriffe auf die Dokumente werden dadurch entsprechend zeitaufwendig. Da der archivierte **Schriftverkehr** in Form von Papieroriginalen und **Aktenordnern** viel Platz beansprucht, kann er mikroverfilmt werden. **Mikrofilmkarten** entstehen durch eine optische Verkleinerung des **Originaldokumentes** auf einen Mikrofilm durch einen fotografischen Prozeß. Der eigentliche Film ist dabei in einem für alle Formate gleich großen Pappträger montiert, der mit weiteren Dokumenten- und Suchinformationen bedruckt ist oder diese Information als maschinenlesbare Lochung enthält. Allerdings kann man die **Mikrofilmkarten** danach nur mit speziellen Geräten betrachten, und auch die **Reproduktion** ist nur mit Spezialgeräten wie einem Mikrofilmkartenrückvergrösserer möglich.

Die **Originaldokumente** können im Anschluß an die Archivierung vernichtet werden, wenn die Archivierungsmethode, wie in Kapitel 6.4 diskutiert, als ausreichend anerkannt wird.

Eine Übersicht über das Szenario zeigt die *Abb. 1: Szenario „Büro“*. Dargestellt sind die Wege der Dokumente beziehungsweise des Schriftverkehrs zwischen den verschiedenen erwähnten Ablage- und Bearbeitungsorten.

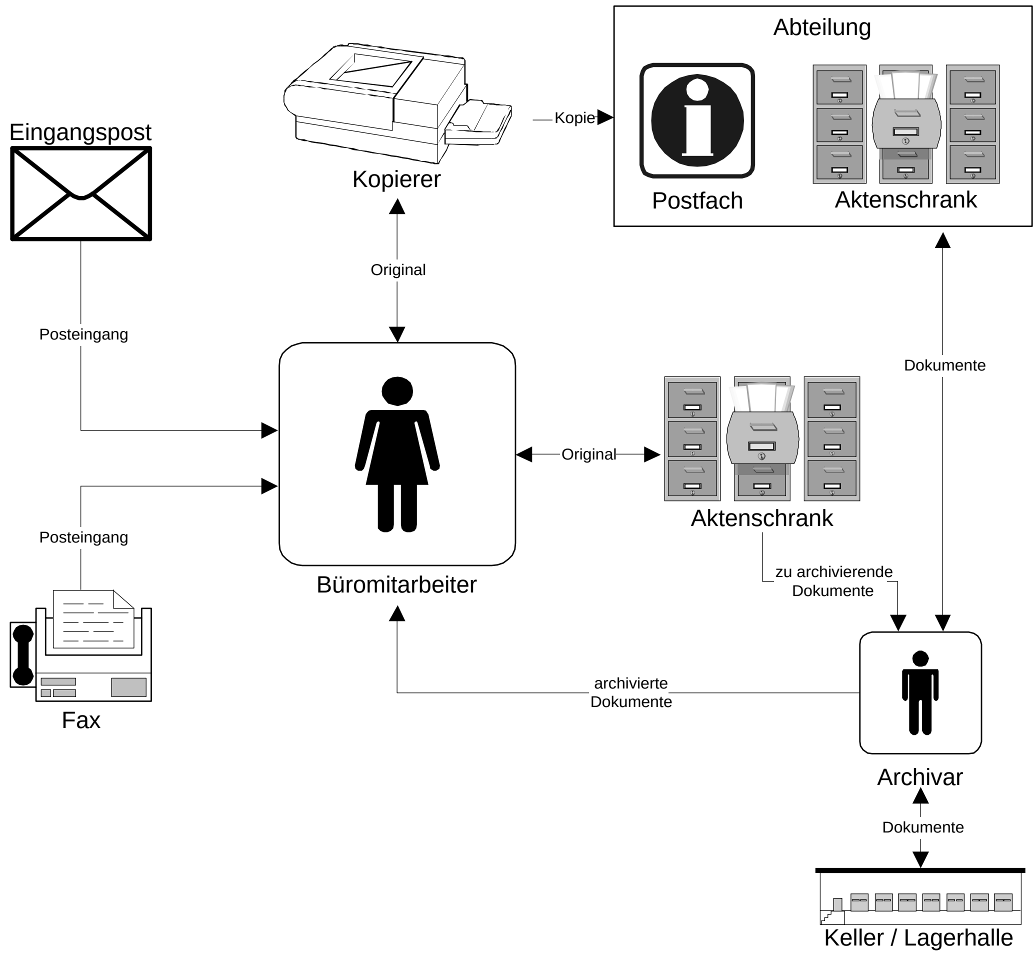

Abb. 1: Szenario „Büro“

2.2 Konstruktionsbüro

Die Aufgabe in einem Konstruktionsbüro ist es unter anderem, mit unterschiedlichen Hilfsmitteln im Kundenauftrag technische **Zeichnungen** anzufertigen. Eine technische **Zeichnung** kann aus mehreren **Blättern** bestehen und jedes dieser **Blätter** kann in verschiedenen **Versionen** vorliegen.

Wie im Szenario „Büro“ folgt hier eine Betrachtung der Vorgänge bei der Anfertigung einer Zeichnung in einem Konstruktionsbüro.

Anfertigung einer Zeichnung

Im Rahmen von Aufträgen und Projekten wird ein **Projektleiter** damit beauftragt, die Anfertigung von Zeichnungen zu veranlassen und zu überwachen. Er schätzt mit seiner Projekterfahrung den Umfang der für ein bestimmtes Projekt anzufertigenden Zeichnungen ab und läßt sich für diese Zeichnungen einen projektbezogenen **Nummernkreis** reservieren. Dieser **Nummernkreis** beschreibt einen Zahlenbereich von Zeichnungsnummern und wird mit dem Projektnamen benannt. Aus diesem

Nummernkreis erhalten die Projektzeichnungen gültige Zeichnungsnummern, wodurch ausgeschlossen wird, daß Zeichnungsnummern durch mehrere parallel abgewickelte Projekte doppelt vergeben werden oder sich Zeichnungsnummern aus mehreren Projekten durchmischen. Durch diese Maßnahme wird ein eindeutiger Rückschluß aus der Zeichnungsnummer auf ein Projekt ermöglicht. Der Projektleiter verteilt die Konstruktionsarbeit auf die **Konstrukteure**, die dann die Zeichnungen in einer ersten **Version** am Zeichenbrett, mit einem **CAD-System** oder mit einem anderen Hilfsmittel anfertigen. Benötigt ein Konstrukteur für seine Zeichnung mehrere Blätter, kann er dafür selbständig neue fortlaufende Blattnummern verwenden. Sind alle Zeichnungsnummern eines Nummernkreises vor Projektende verbraucht, muß der Projektleiter nachträglich einen neuen, weiteren Nummernkreis reservieren. Ist ein Projekt abgeschlossen und sind Zeichnungsnummern eines Nummernkreises nicht vollständig benötigt worden, kann der nicht benötigte Block von Zeichnungsnummern am Ende des Nummernkreises für neue Projekte freigegeben werden.

Qualitätskontrolle

Ist der Konstruktionsprozeß abgeschlossen, werden die Zeichnungen durch einen mit der Qualitätssicherung beauftragten Mitarbeiter auf die Einhaltung geltender Vorschriften, Richtlinien und Normen überprüft. Dazu werden die Papierzeichnungen per Hauspost diesem **Qualitätssicherungsmitarbeiter** zugestellt. Zur Überprüfung einer mit einem **CAD-System** erstellten Zeichnung muß diese zuvor ausgedruckt werden, wenn dem **Qualitätssicherungsmitarbeiter** nicht das gleiche **CAD-System** zur Verfügung steht. Wurde bei der Überprüfung ein Mangel an einer Zeichnung festgestellt, wird die Zeichnung zusammen mit einem **Notizzettel** zur Überarbeitung an den Konstrukteur zurückgegeben. Auf dem **Notizzettel** wird der Mangel beschrieben, so daß dieser vom Konstrukteur behoben werden kann. Wird kein Mangel festgestellt, wird die Zeichnung mit einem Freigabestempel versehen und in Kopie an den Auftraggeber weitergegeben.

Archivierung

Die Lagermöglichkeiten für Zeichnungen innerhalb eines Konstruktionsbüros selber sind in der Regel beschränkt. Deshalb werden freigegebene Zeichnungen, die nicht mehr direkt im Konstruktionsbüro benötigt werden, durch einen **Archivar** für den späteren Zugriff archiviert. Wie bei dem im Szenario „Büro“ behandelten Schriftverkehr gibt es auch bei diesen Zeichnungen viele Gründe für eine Archivierung. Bei Änderungs- und Erweiterungswünschen seitens des Kunden ist auch nach längerer Zeit noch Zugriff auf die Zeichnung erforderlich, um die Änderungen an der Zeichnung durchführen zu können. Dazu erhält die neue, geänderte Zeichnung zur Unterscheidung von der alten Zeichnung eine neue Versionsnummer. Der Zugriff auf eine bereits fertiggestellte Zeichnung kann auch bei Neuprojekten Zeit sparen, wenn für die neue Zeichnung bestehende Zeichnungsbereiche mit Konstruktionselementen durch Kopieren und Einfügen wiederverwendet werden können. Weitere Archivierungsgründe können wieder rechtlicher Natur sein, zum Beispiel Produkthaftungsgesetzte oder andere Nachweispflichten.

Wie in dem Szenario „Büro“ werden auch in einem Konstruktionsbüro archivierte Zeichnungen aus Platzmangel in Keller oder Lagerhallen ausgelagert. Dabei werden die Zeichnungen als Papieroriginal archiviert. Durch die Größe der Zeichnungen bedingt, können diese nicht einfach in Ordnern ablegt werden. Zeichnungen liegen in der Regel in verschiedenen DIN-Formaten von DIN A4 bis DIN A0, in Einzelfällen auch größer, vor. Zur Ablage sind vielmehr spezielle **Zeichnungsschränke** nötig, in denen die Zeichnungen nach Zeichnungsnummern und DIN-Formaten sortiert liegend gelagert werden. Da diese Art der Ablage einen hohen Platzbedarf hat und die großen Zeichnungen schlecht zu handhaben sind, können zur Platzreduzierung **Mikrofilmkarten** erzeugt werden. **Mikrofilmkarten** entstehen durch eine optische Verkleinerung des **Originaldokumentes** auf einen Mikrofilm durch einen fotografischen Prozeß. Der eigentliche Film ist dabei in einem für alle Formate gleich großen Papptäger montiert, der mit weiteren Dokumenten- und Suchinformationen bedruckt ist oder diese Information als maschinenlesbare Lochung enthält. Allerdings kann man die **Mikrofilmkarten** danach nur mit speziellen Geräten betrachten, und auch die **Reproduktion** ist nur mit Spezialgeräten wie einem Mikrofilmkartenrückvergrösserer möglich. Durch die gleiche Größe der den Film tragenden Pappträger ist eine nach DIN-Formaten sortierte Ablage wie bei dem **Zeichnungsschrank** nicht nötig. Die Zeichnungen können vielmehr zum leichteren Wiederfinden nach Zeichnungsnummer, Blattnummer und Version sortiert abgelegt werden.

Die **Originaldokumente** können im Anschluß an die Archivierung, wenn die Archivierungsmethode von den zuständigen Stellen, wie in Kapitel 6.4 diskutiert, als ausreichend anerkannt wird.

Suche und Reproduktion

Zum Wiederfinden einer Zeichnung, die nach dem oben beschriebenen Ablageverfahren archiviert wurde, wird die Zeichnungsnummer benötigt. Da aber meistens nach anderen bekannten Informationen wie Erstellungsdatum der Zeichnung, Name des Konstrukteurs oder Zeichnungsbenennung gesucht wird, muß die Zuordnung dieser Informationen zu den Zeichnungsnummern verwaltet werden. Diese Zuordnung kann in handgeführten **Zeichnungslisten** oder in rechnergeführten Datenbanken geschehen. Erst nachdem der Konstrukteur, eventuell unter Zuhilfenahme einer solchen Zuordnung, die Zeichnungsnummer der zu suchenden Zeichnung herausgefunden hat, kann die Zeichnung im Archiv gefunden werden. Nach einer Überprüfung durch den Konstrukteur, ob die gefundene Zeichnung tatsächlich die gesuchte Information beinhaltet, kann eine Kopie durch ein **Vervielfältigungsgerät** zur weiteren Bearbeitung erzeugt werden. Liegt die Zeichnung als Papieroriginal vor, so benötigt man zur Reproduktion eine **Pausmaschine**, liegt die Zeichnung als **Mikrofilmkarte** vor, benötigt man zur Reproduktion einen **Mikrofilmkartenrückvergrösserer**.

Eine Übersicht über das Szenario zeigt die Abb. 2: Szenario „Konstruktionsbüro“. Dargestellt sind die Wege der Dokumente beziehungsweise der Zeichnungen zwischen den verschiedenen erwähnten Ablage- und Bearbeitungsorten.

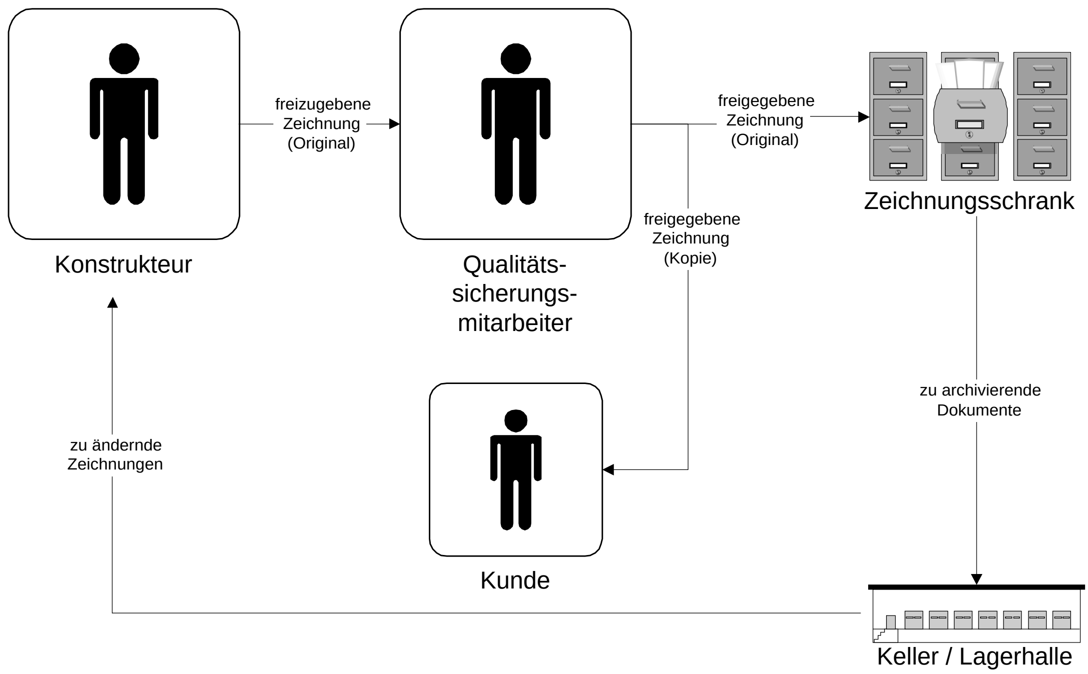

Abb. 2: Szenario „Konstruktionsbüro“

2.3 Softwareentwicklung

Bei der Softwareentwicklung entwickelt eine Gruppe von Programmierern und anderen Softwareentwicklern im Rahmen von Projekten Software.

Wie in den Szenarien „Büro“ und „Konstruktionsbüro“ folgt hier eine Betrachtung der Vorgänge bei der Softwareentwicklung.

Gruppen

Ein Gruppenleiter leitet die Softwareentwicklung und teilt das Programm in logische Strukturen auf, die dann von einzelnen Personen oder Untergruppen bearbeitet werden. Die Zusammensetzung und die Strukturen der Gruppen kann dabei bei veränderten Voraussetzungen jederzeit geändert werden.

Erstellung der Dokumente

Während der Entwicklung der Software entstehen verschiedene Dokumente wie Programmquelltexte, Programmdokumentation, Handbücher und weitere Projektunterlagen. Diese Dokumente werden heutzutage fast ausschließlich mit einem Computersystem erstellt.

Bearbeitung

Die Bearbeitung eines Projektes mit Schreiben der Dokumentation und mit Entwickeln der Programme geschieht über einen längeren Zeitraum. Durch die Arbeitsaufteilung ergibt sich eine große Parallelität der Arbeitsabläufe. Bearbeitungsstände können regelmäßig durch ein Versionskontrollsystem gesichert werden, um jederzeit wieder auf eine funktionierende Programmversion zurückgreifen zu können. Die Dokumente unterliegen zur Projektlaufzeit einer großen Änderungsrate, und so sind die gesicherten Bearbeitungsstände selten aktuell und entsprechen nur einem Augenblickszustand.

Ablage der Dokumente

Während der Projektlaufzeit werden die Dokumente gesichert, um einem Datenverlust vorzubeugen. Die Dokumente, an denen aktuell gearbeitet wird, verbleiben bis zum Projektende auf dem Computersystem. Nach Projektende werden die Datenträger und Unterlagen mit den Programmquelltexten und Dokumenten projektbezogen abgelegt. Das dabei entstehende Platzbedarf und Datenvolumen ist vergleichsweise gering.

Änderungen

Nach Projektende erfolgt beispielsweise ein Zugriff auf die Projektdokumente, wenn Änderungen oder Erweiterungen an der Software vorgenommen werden sollen. Bei kleineren Änderungen wie der Behebung von Fehlern werden die entsprechenden Programmquelltexte geändert und alle davon abhängigen Dokumente wie Handbücher aktualisiert.

Bei größeren Änderungen oder Erweiterungen werden in der Regel alle Projektdokumente wieder herangezogen, da die Dokumente untereinander stark abhängig sind. Änderungen im Programmquelltext beispielsweise ziehen meistens auch eine Änderung an den Handbüchern nach. Die weiteren Arbeitsabläufe entsprechen denen, die bei einem neuen Projekt ebenso durchgeführt werden und enden bei Projektende wieder in der Ablage der Dokumente.

Wiederverwendung

Dokumente eines Projektes können auch bei der Bearbeitung eines anderen Projektes hilfreich sein und dort wiederverwendet werden. Das Kopieren der in elektronischer Form vorliegenden Dokumente erfolgt auf einfache Weise. Die kopierten Dokumente können dann in dem neuen Projekt wiederverwendet werden. Wurde allerdings bei der Bearbeitung eines Projektes keine Aufmerksamkeit darauf verwendet, eventuell Teile des Programmquelltextes wiederverwenden zu können, so besteht ein Interesse an den Programmquelltexten nur während der Projektlaufzeit und kaum noch danach.

Eine Übersicht über das Szenario zeigt die Abb. 3: Szenario "Softwareentwicklung". Dargestellt sind die Wege der Dokumente zwischen den verschiedenen erwähnten Ablage- und Bearbeitungsorten.

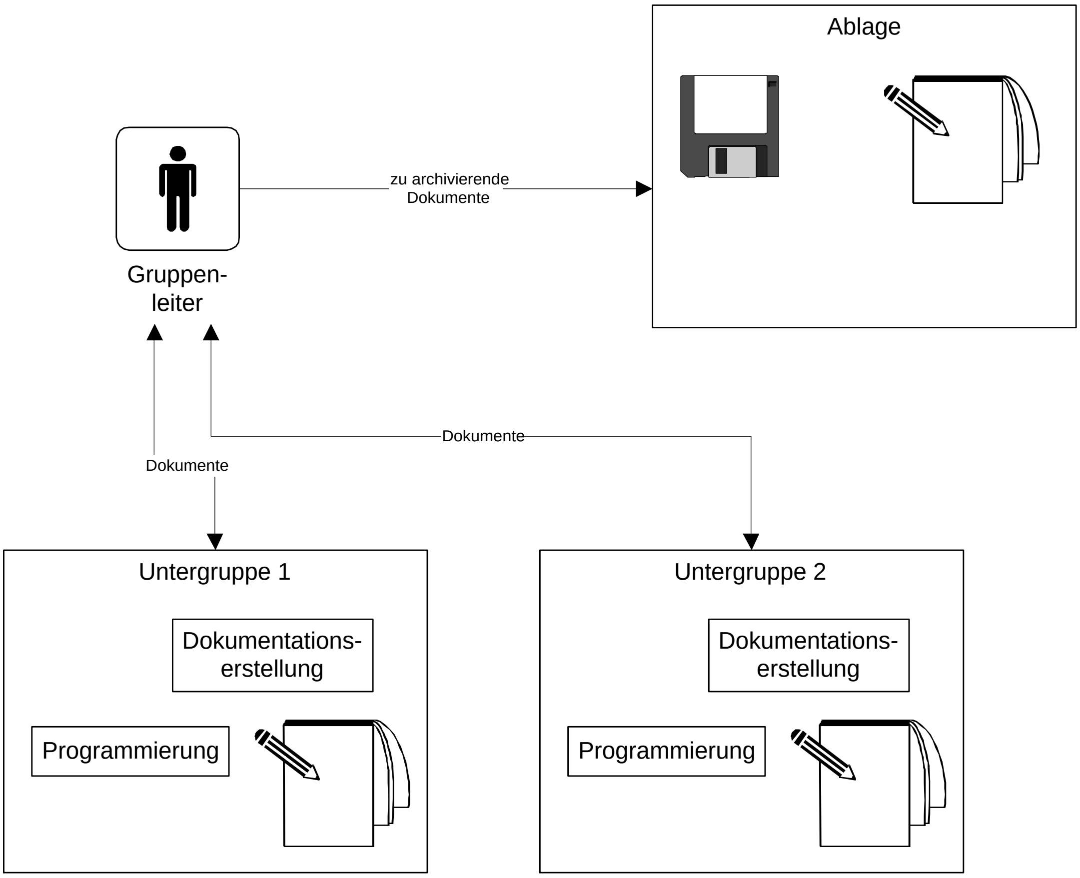

Abb. 3: Szenario "Softwareentwicklung"

2.4 Gemeinsame Begriffe der Szenarien

Als weitere Diskussionsgrundlage werden die wichtigsten Begriffe, die in allen Szenarien verwendet werden, beschrieben. Weitere Begriffe werden in Kapitel 8.1.2 in dem Problembereichsmodell erläutert.

Dokument

Da der Begriff „Dokument“ in dieser Arbeit sehr häufig verwendet wird und in Archivsystemen von zentraler Bedeutung ist, soll hier darauf eingegangen werden, was unter einem „Dokument“ im Rahmen dieser Arbeit zu verstehen ist.

In [Brock78] wird ein Dokument als Urkunde oder Beweisstück definiert. Dies wird auch durch die etymologische Deutung unterstützt, denn das Wort Dokument ist von dem lateinischen Wort „documentum“ abgeleitet und bedeutet dort „Beweis“ [Mack85]. Diese beiden Definitionen sind aber noch zu allgemein gehalten. Eine genauere Definition wird in einem Wörterbuch für Archivbegriffe [Waln88] gegegeben. Dort wird ein Dokument wie folgt definiert: Ein Dokument ist „eine Kombination aus einem Medium mit der Information darauf oder darin. Ein Dokument kann als Beweis dienen oder zur Konsultation herangezogen werden.[Waln88]“

Weiterhin beschreibt [Waln88] das Dokument als die „kleinste archivtechnisch unteilbare“ Einheit, bestehend aus einem oder mehreren Blättern, einem Heft oder Band. „Archivtechnisch“ bedeutet in diesem Zusammenhang, daß die Dokumente aus Archivsicht betrachtet werden, und eine Behandlung kleinerer Einheiten aus Sicht eines Archivs nicht sinnvoll ist.

In den angegeben Szenarien kann man also Rechnungen, Bestellungen und den anderen Schriftverkehr, Zeichnungen oder Handbücher als Dokumente auffassen, da sie „eine Kombination aus einem Medium (hier Papier) mit der Information darauf oder darin“ sind und so die Merkmale nach [Waln88] erfüllen.

In vielen Bereichen hat sich aber heute die Technik weiterentwickelt und die elektronische Datenverarbeitung (EDV) bestimmt das Bild in vielen Bereichen, in denen Dokumente behandelt werden. Im Bereich der elektronischen Datenverarbeitung kann man nach obiger Definition unter dem Begriff „Dokument“ alle auf oder in einem Medium, hier im Computerspeicher, vorliegende Informationsobjekte wie Texte, Grafiken, Tabellen, Bilder, Tonsequenzen, Videosequenzen, Animationen und anderes verstehen. Auch die Programmquelltexte in der elektronischen Form aus dem Szenario „Softwareentwicklung“ können so als Dokument aufgefaßt werden.

Archiv

Ein Archiv dient der strukturierten Ablage von Dokumenten. Mit Hilfe eines Archivs kann man große Mengen Dokumente sinnvoll verwalten. Dokumente sind in einem Archiv so abgelegt, daß sie anhand verschiedener Informationen wiedergefunden werden können. Um die große Menge von Dokumenten unterzubringen, werden im allgemeinen Kellerräume oder Lagerhallen verwendet.

2.5 Gemeinsame Merkmale der Szenarien

Bereits jetzt lassen sich für alle Szenarien gemeinsame Merkmale feststellen. Die in den verschiedenen Bereichen entstehende Dokumentenmenge erfordert durch die langen Aufbewahrungszeiträume eine besondere Vorgehensweise zur Ablage der Dokumente. Die genutzte Ablageform für diese Dokumente ist ein Archiv, daß aus Platzgründen von den eigentlichen Bearbeitungspunkten räumlich entfernt liegt. Dokumente werden anhand weniger Merkmale sortiert abgelegt. Bei Bedarf werden Dokumente in dem Archiv gesucht und kopiert oder direkt weiterverarbeitet.

Die in den Szenarien durchgeführte Archivierung weist aber auch Probleme und Schwachstellen in vielen Bereichen auf. Deshalb werden in dem nächsten Kapitel mögliche Verbesserungen dieser Probleme und Schwachstellen durch Computereinsatz diskutiert.

KAPITEL 3

3 Unterstützung durch Computereinsatz

Die oben vorgestellten Szenarien können durch Computerunterstützung verbessert und optimiert werden. Eine Verbesserung wird erreicht, wenn man eine Zeit- oder Kostenersparnis erzielt, den beteiligten Personen einen größeren Komfort bietet oder die Qualität der Reproduktion eines Originals steigert. Potential zur Vereinfachung liegt also im zeitaufwendigen Kopieren, Verteilen und Suchen des Schriftverkehrs und in der teuren raumintensiven Ablage. Es muß eine Lösung gefunden werden, die diese Punkte berücksichtigt und vereinfacht. Die Faktoren (siehe auch [Bull93],[GSZ93]) die dabei eine Rolle spielen, werden jetzt genauer betrachtet.

Faktor Zeit

Im Szenario „Büro“ besteht die manuelle Erzeugung und Weiterleitung der Kopien aus vielen einzelnen zeitintensiven Tätigkeiten. Die zur Erzeugung einer Kopie zurückzulegenden Wege zu den Kopiergeräten sind zu berücksichtigen. Die Kopierunterlagen müssen durch Entklammern und Entheften auf das Kopieren vorbereitet werden. Die Anfertigung der Kopien selbst mit den Kopiergeräten benötigt ebenfalls Zeit. Auch die abschließende Endbearbeitung, die aus dem Zusammenstellen der Kopien zu Sätzen und dem Zusammenheften besteht, muß wieder manuell erfolgen. Nach Fertigstellung der Kopien erfolgt das Anbringen der Verteilerinformation sowie die Weiterleitung der Kopien an den Empfänger mit den dabei auftretenden Laufzeiten.

Im Szenario „Konstruktionsbüro“ benötigt die Freigabeprozedur einen langen Zeitraum. Alle Zeichnungen aus dem CAD-System müssen vor der Freigabe ausgegeben werden, wenn dem Qualitätssicherungsmitarbeiter nicht das gleiche CAD-System zur Verfügung steht. Die Zeichnungen müssen per Hauspost an den Qualitätssicherungsmitarbeiter gesendet werden. Bei Mängeln wird die fehlerhafte Zeichnung mit dem Notizzettel wieder per Hauspost zurückgeschickt. Nach Bearbeitung der Mängelliste und einer eventuellen Neuausgabe aus dem CAD-System beginnt diese Freigabeprozedur wieder von vorne und dauert bis zur endgültigen Freigabe der Zeichnung.

Darüber hinaus entstehen in diesen beiden Szenarien Suchzeiten bis zur Auffindung eines Dokumentes, wenn später ein Schriftstück oder eine Zeichnung benötigt wird. Ist ein Dokument nicht direkt am Arbeitsplatz vorhanden, entstehen Zugriffszeiten und bei einer benötigten Kopie auch wieder die erwähnten Kopierzeiten.

Faktor Material

Im Szenario „Büro“ werden Kopien immer nach einem fest definierten Verteiler erzeugt und verteilt. Auch wenn sie später nicht benötigt werden, wurde bei der Kopienerstellung Papier verbraucht, welches hätte eingespart werden können. Eine bedarfsgerechte Erzeugung der Kopien zu einem späteren Zeitpunkt würde einen unnötigen Materialverbrauch verhindern. Je nach Informationsbedarf genügt eventuell auch eine Einsichtnahme in die Originaldokumente.

Im Szenario „Konstruktionsbüro“ werden während der Freigabeprozedur durch den Qualitätssicherungsmitarbeiter fehlerhafte Zeichnungen wieder zum Konstrukteur zurückgeschickt. Der Konstrukteur verbessert die Zeichnung mit Hilfe seines CAD-Systems und produziert, um die Freigabeprozedur fortzusetzen, einen erneuten Ausdruck der Zeichnung, wenn dem Qualitätssicherungsmitarbeiter nicht das gleiche CAD-System zur Verfügung steht. Dieses führt zu einem unnötigen Papierverbrauch.

Faktor Raum

Originalunterlagen in Papierform benötigen viel Raum. Eine Platzersparnis kann erzielt werden, wenn statt der Originale die Daten in elektronischer Form auf geeigneten Datenträgern gespeichert werden. Auf einer Compact-Disc (CD), die ungefähr 640 MB faßt, kann der Inhalt mehrerer Aktenordner mit Dokumenten gespeichert werden. Möglich macht dies eine Komprimierung der Daten. Bei der Komprimierung werden die Daten durch verschiedene Techniken so codiert, daß sie weniger physikalischen Platz beanspruchen. Bei der Entkomprimierung werden dann die ursprünglichen Daten wiederhergestellt.

Interessant ist ein direkter Vergleich des Raumbedarfs zwischen einer CD und der darauf speicherbaren Anzahl der Aktenordner. Überschlägige Berechnungen ergeben, daß eine CD inklusive Hülle einen Raum von etwa 14 x 12 x 1 cm (168 cm^3) benötigt. Die auf der CD speicherbaren fünfzehn Aktenorder mit Dokumenten aber haben aneinandergereiht eine Länge von ungefähr einem Meter und benötigen ungefähr 100 x 32 x 29 cm (92800 cm^3) Raum, also ungefähr fünfhundert mal soviel Platz wie die CD.

Eine Übersicht über den Platzbedarf gibt die folgende Tabelle (Tab. 1: Platzbedarf). Alle Dokumente werden mit 300 dpi eingescannt und gespeichert. Der angegebene Speicherbedarf ergibt sich aus den Durchschnittswerten, die üblicherweise auftreten.

Art des Dokumentes	Größe Original	Speicherbedarf unkomprimiert. Schwarzweiß	Speicherbedarf komprimiert Schwarzweiß	Speicherbedarf unkomprimiert Farbe 24bit	Speicherbedarf komprimiert Farbe 24bit
Rechnung	A4	1.1 MB	80 KB	26 MB	1 MB
Zeichnung	A0	18 MB	500 KB	427 MB	16 MB
Ordner	500 A4	550 MB	40 MB	13 GB	500 MB
Zeichnungs-schrank	500 A0	9 GB	250 MB	213.5 GB	8 GB

Tab. 1: Platzbedarf von Dokumenten

Faktor Qualität

Bei der Erzeugung von Kopien eines Dokumentes tritt aus technischen Gründen ein Qualitätsverlust auf. Dieser Qualitätsverlust wird noch größer, wenn von Kopien erneut Kopien erzeugt werden. Gerade bei technischen Zeichnungen und Rechnungsunterlagen ist aber jederzeit eine eindeutige Lesbarkeit der Dokumente gefordert. Kopien gleichbleibender Qualität können erzeugt werden, wenn die Dokumente elektronisch gespeichert werden und Kopien von dem gespeicherten Originaldokument als Ausdrucke in gleichbleibender Qualität erzeugt werden.

Papierdokumente sind empfindlich gegenüber äußeren Einflüssen und vergilben, bekommen Risse oder werden durch Feuchtigkeit und Hitze geschädigt. Empfangene Faxe auf Thermopapier sind empfindlich gegenüber Lichteinflüssen und verblassen. Nach der Erfassung durch Scanner auf Datenträgern gespeicherte Dokumente hingegen können auch noch nach langer Zeit in der ursprünglichen Qualität ausgedruckt werden, zumindest solange, wie weiterhin die entsprechenden Lesegeräte zur Verfügung stehen. Bei Verwendung einer weitverbreiteten Technologie zur Datenspeicherung und einer rechtzeitigen Migration der Daten auf Medien einer neueren Technologie ist dies aber kein prinzipielles Problem.

KAPITEL 4

4 Systemvision

In der hier beschriebenen Systemvision werden erste Ansätze zur Lösung der Archivproblematik mit Hilfe eines Archivsystems unter Berücksichtigung der in Kapitel 3 genannten Punkte gezeigt. Auch hier werden wieder die oben eingeführten Szenarien unterschieden und es wird für jedes Szenario eine eigene Systemvision entworfen.

4.1 Systemvision für das Szenario „Büro“

Die eingehende Post wird weiterhin von der **Büromitarbeiter** geöffnet und gesichtet. Anhand der Art des **Schriftverkehrs** wird ein **Büromitarbeiter** eine eventuell erforderliche Archivierung einleiten. Zur Bearbeitung des Schriftverkehrs mit einem Computer ist es erforderlich, daß der in Papierform vorliegende Schriftverkehr über ein spezielles Gerät in eine für den Computer direkt verwendbare Form überführt wird. Diese speziellen Geräte werden **Scanner** genannt.

Erfassen

Bei der Erfassung eines Dokumentes durch das Archivsystem können automatisch Daten von dem System festgestellt und gespeichert werden. Solche **Systemdaten** sind das **Erfassungsdatum**, an dem ein Dokument archiviert wird, und der Name des Büromitarbeiters beziehungsweise allgemein der Name des Benutzers. Das Dokument wird von dem Büromitarbeiter durch Eingabe von zusätzlichen Informationen, den **Userdaten**, in der **Userdaten-Eingabemaske** attributiert. In den **Userdaten** können abhängig von der Art des **Dokuments** und seines durch den **Dokumententext** gegebenen Inhalts diejenigen Informationen durch den Büromitarbeiter erfaßt werden, die ein einfaches Wiederauffinden des Dokumentes ermöglichen. Die **Userdaten** sind von der Art des Dokumentes abhängig, da nicht für alle Dokumentenarten die gleichen Attribute benötigt werden. Es gibt also verschiedene **Dokumententypen**. In diesem Szenario werden die Dokumententypen **Rechnung**, **Bestellung** und **Produktinformation** benutzt. Zu einer **Rechnung** werden Rechnungsdatum, Eingangsdatum, Rechnungsbetrag, Kostenstelle und Name des Rechnungsstellers gespeichert. Zu einer **Bestellung** werden Bestelldatum, Eingangsdatum, Name des bestellten Artikels, Bestellbetrag und Name des Bestellers gespeichert. Zu einer **Produktinformation** oder Werbung werden der Name der Produktgattung, Name des Produktes, Preis, Datum des Beginns und des Endes der Gültigkeit des Preises und der Name des Lieferanten gespeichert.

Übernahme

Die **Eintragsdaten,** die aus den **Userdaten** und den **Systemdaten** gebildet werden, ermöglichen ein späteres Auffinden des Dokuments. Die Auswahl des zu archivierenden Dokumentes erfolgt mit Hilfe einer **Dateiauswahlliste**, in der die Dateien des Computersystems zur Auswahl angezeigt werden. Das Dokument wird zusammen mit den **Eintragsdaten** in das **Archivsystem** übernommen und bildet dort einen **Eintrag.**

Zugriff

Durch die Übernahme des Dokumentes mit den **Eintragsdaten** in das Archivsystem können auch andere Benutzer, die an das Archivsystem angeschlossen sind, bei Bedarf sofort darauf zugreifen, da entsprechende Suchanfragen sofort auch die neuen Dokumente zeigen.

Suche

Wird ein bestimmtes Dokument gesucht, können **Suchkriterien** angegeben werden. Es gibt zwei unterschiedliche Arten von Suchmöglichkeiten. Die erste Suchmöglichkeit benutzt zur Suche die bei der Erfassung der Dokumente eingegebenen **Eintragsdaten**. Die zweite Suchmöglichkeit sucht in dem Dokumententext nach einem beliebigen eingegebenen Begriff, was auch **Volltextsuche** genannt wird. Das Archivsystem sucht bei abgeschlossener Eingabe der Suchkriterien nach passenden Einträgen. Die Dokumente der gefundenen Einträge können auf dem Computerbildschirm angezeigt werden.

Reproduktion

Wird für den Postversand oder für andere Zwecke eine Papierausgabe benötigt, kann zur Ausgabe des Dokumentes ein lokal vorhandener oder ein zentral aufgestellter **Drucker** genutzt werden.

Eine grafische Übersicht über eine mögliche Umgebung zeigt die Abbildung 4 (Abb. 4: Systemvision „Büro“).

Verbesserungen

Durch einen Vergleich mit den in Kapitel 3 genannten Verbesserungsmöglichkeiten können die durch eine solche Vorgehensweise erreichten Vorteile bewertet werden. Die unter dem Faktor Zeit erwähnten zeitintensiven Tätigkeiten entfallen weitgehend. Das Verteilen der Dokumente geschieht automatisch durch das Archivsystem. Alle Dokumente können zeitsparend am Arbeitsplatz gesucht und betrachtet werden. Durch den kompletten Wegfall der Papierkopien wird dem Faktor Material Rechnung getragen. Die elektronische Speicherung unterstützt durch den erheblich reduzierten Platzbedarf den Faktor Raum. Durch die Möglichkeit, von den gespeicherten Dokumenten Ausdrucke in stets gleichbleibend guter Qualität zu erzeugen, wird schließlich der Faktor Qualität berücksichtigt.

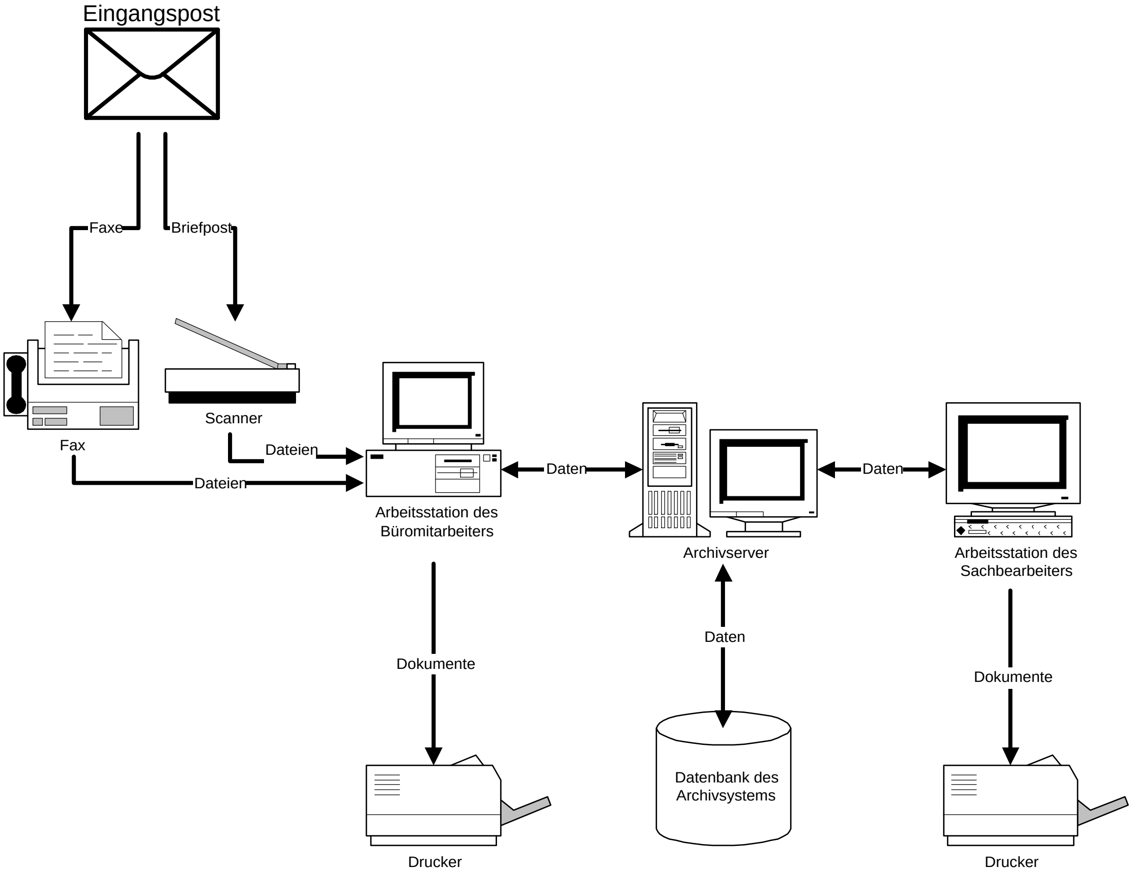

Abb. 4: Systemvision „Büro“

4.2 Systemvision für das Szenario „Konstruktionsbüro“

Zur Vereinfachung dieses Szenarios durch Computerunterstützung kann man entsprechende Überlegungen anstellen wie im Büroszenario. Die Zeichnungserstellung durch den **Konstrukteur** wird wie vorher von Hand oder mit darauf spezialisierten **CAD-Systemen** geschehen und findet unabhängig vom **Archivsystem** statt. Nach Beendigung der Zeichnungserstellung erfolgt die **Freigabe** durch den **Qualitätssicherungsmitarbeiter** mit Unterstützung durch das Archivsystem.

Erfassung

Dazu wird die **CAD-Zeichnung** vom **Konstrukteur** mit bekannten Daten über die Zeichnung in das **Archivsystem** übernommen (vgl. Kapitel 4.1 „Systemvision für das Szenario „Büro“„). Die hier verwendeten **Userdaten** einer Zeichnung sind Zeichnungsnummer, Blattnummer und Versionsnummer sowie eine textuelle Beschreibung der Zeichnung. Die **Systemdaten** werden aus dem Namen des Konstrukteurs und dem **Erfassungsdatum** gebildet. Die aus den Userdaten und Systemdaten bestehenden **Eintragsdaten** werden zusammen mit der **Zeichnung** in das Archivsystem übernommen und bilden dort einen **Eintrag**.

Übernahme

Bei der Übernahme durch das Archivsystem wird vermerkt, daß diese Zeichnung noch zur Qualitätskontrolle ansteht. Zur **Freigabe** einer auf Papier vorliegenden **Zeichnung** muß diese zur

Übernahme in das Archivsystem erst durch einen **Scanner** in ein computerverarbeitbares Format gebracht werden.

Freigabe

Nach erfolgter Zeichnungsübernahme kann der **Qualitätssicherungsmitarbeiter** dann die **Zeichnung**, die er als nächstes bearbeiten will, auswählen, und die Qualitätskontrolle durchführen. Bei Mängeln kann er mit Hilfe des Archivsystems einen **Notizzettel** mit der Mängelbeschreibung zur Zeichnung verfassen, und über das Archivsystem beides dem Konstrukteur durch **Weiterleitung** zu einer Überarbeitung zurückleiten. Findet der **Qualitätssicherungsmitarbeiter** keinen Mangel, wird die erfolgreiche **Freigabe** der Zeichnung durch das **Archivsystem** vermerkt.

Zugriff

Durch die Übernahme der Zeichnung mit den **Eintragsdaten** in das Archivsystem können auch andere Benutzer, die an das Archivsystem angeschlossen sind, bei Bedarf sofort darauf zugreifen, da entsprechende Suchanfragen sofort auch die neuen Zeichnungen zeigen.

Suche

Über die mit der Zeichnung abgespeicherten Zeichnungsinformationen kann ein Konstrukteur bestimmte Zeichnungen suchen. Zur Kontrolle des Suchergebnisses kann er sich die gefundenen Zeichnungen direkt auf dem Bildschirm ansehen. Möchte der Konstrukteur Teile dieser Zeichnung in einer neuen Zeichnung wiederverwenden, kann er zur Bearbeitung der Zeichnung einen Editor starten.

Reproduktion

Eine in Papierform benötigte Zeichnung kann auf einfache Weise wieder ausgedruckt werden, indem zum Beispiel ein Druckauftrag mit der Zeichnung an ein **Plotmanagementsystem** übergeben wird.

Änderung

Soll eine bereits freigegebene **Zeichnung** geändert oder erweitert werden, wird die archivierte **Zeichnung** wieder an das **CAD-System** übergeben und kann dort geändert werden. Die archivierte Zeichnung hingegen ist gegen Änderung und Erweiterung geschützt, damit den Nachweispflichten genüge getan wird. Die geänderten oder erweiterten Zeichnungen werden als neue **Version** über den oben beschriebenen Weg wieder in das Archiv übernommen. Auf diese Weise stehen jederzeit auch ältere Versionen zur Verfügung.

Der **Projektleiter** legt fest, welche Daten zu einer Zeichnung gespeichert werden und wer diese Daten verändern darf. Außerdem muß er festgelegen können, welche Personen **Qualitätssicherungsmitarbeiter** sind und damit Zeichnungen freigeben dürfen.

Eine grafische Übersicht über eine mögliche Umgebung zeigt die Abbildung 5 (Abb. 5: Systemvision „Konstruktionsbüro").

Verbesserungen

Durch einen Vergleich mit den in Kapitel 3 genannten Verbesserungsmöglichkeiten können die durch eine solche Vorgehensweise erreichten Vorteile bewertet werden. Die unter dem Faktor Zeit erwähnten zeitintensiven Tätigkeiten bei der Freigabeprozedur entfallen weitgehend. Die Freigabe der Dokumente geschieht mit Unterstützung durch das Archivsystem. Alle Dokumente können zeitsparend am Arbeitsplatz gesucht und betrachtet werden. Durch den kompletten Wegfall der Papierausgaben aus den CAD-Systemen zur Qualitätskontrolle wird dem Faktor Material Rechnung getragen. Die elektronische Speicherung unterstützt durch den erheblich reduzierten Platzbedarf den Faktor Raum. Durch die Möglichkeit, von den gespeicherten Dokumenten Ausdrucke in stets gleichbleibend guter Qualität zu erzeugen, wird schließlich der Faktor Qualität berücksichtigt.

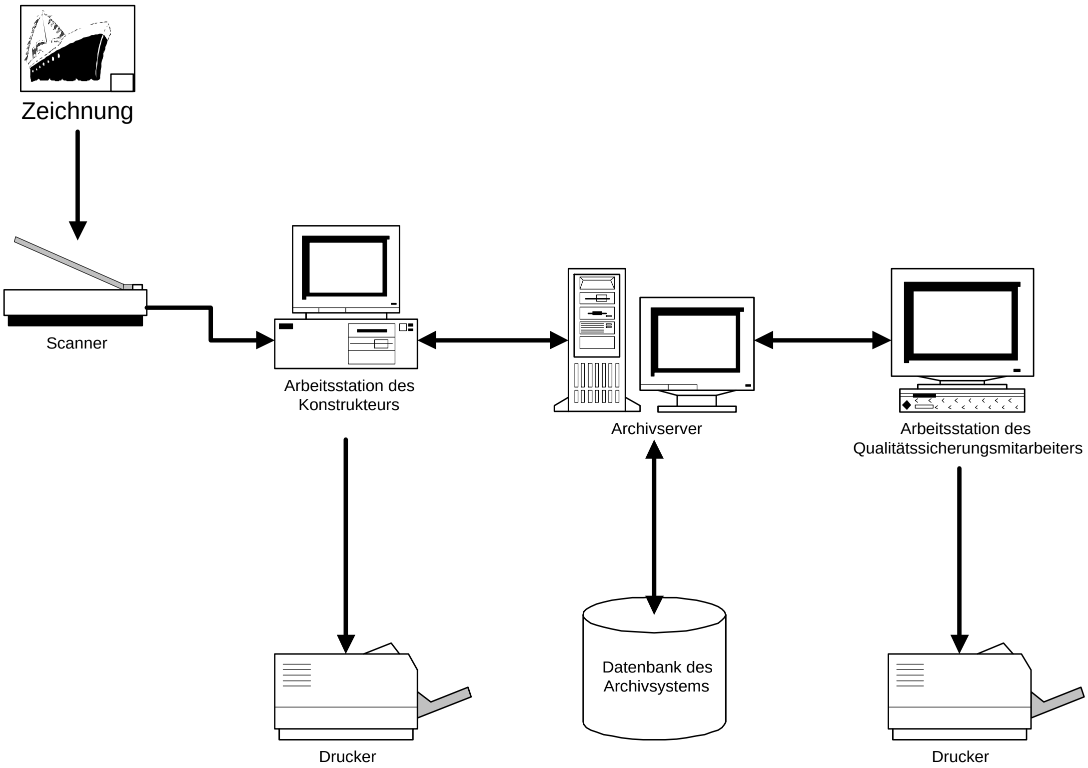

Abb. 5: Systemvision „Konstruktionsbüro“

4.3 Überlegungen zur Systemvision für das Szenario „Softwareentwicklung“

Die Arbeitsabläufe bei der Softwareentwicklung lassen sich unter Berücksichtigung der in Kapitel 3 Punkte durch ein Archivsystem nicht wesentlich verbessern, da die Probleme bei der Softwareentwicklung in Bereichen liegen, die nicht durch Archivsysteme abgedeckt werden.

Zeit

Die durch die Verteilung der Dokumente entstehenden Zeitverluste sind gering, und gegenüber der Projektlaufzeit sogar zu vernachlässigen. Dokumente werden von einer Person oder Gruppe erstellt und in der Regel durchgängig bearbeitet, so daß Dokumententransporte oder Vervielfältigungen selten sind.

Material

Auch der Materialverbrauch ist gering, da Programmquelltexte in der Regel nicht und die andere Dokumente selten auf Papier ausgegeben werden.

Raum

Der Platzbedarf der während eines Projektes erzeugten Dokumente ist nicht so groß, daß die Dokumente notwendigerweise in Kellerarchive oder Lagerhallen ausgelagert werden müßten. Ein Großteil der Dokumente liegt bereits in elektronischer Form vor und kann platzsparend auf Datenträgern abgelegt werden.

Qualität

Auch das Problem der Qualitätsverluste stellt sich nicht, da die Dokumentation in elektronischer Form vorliegt und jederzeit ohne Verluste ausgegeben werden kann.

Verbesserungen

Die Diskussion zeigt, daß Archivsysteme nur eine mangelhafte Unterstützung der Arbeitsabläufe bei der Softwareentwicklung bieten können und der Einsatz daher nicht sinnvoll ist. Aus diesem Grund wird das Szenario „Softwareentwicklung“ im Verlauf dieser Arbeit nicht weiter betrachtet.

Alternativen

Zu einer Computerunterstützung der Arbeitsabläufe bei der Softwareentwicklung eignen sich andere Softwaresysteme als Archivsysteme besser. Eine besondere Behandlung der Programmquelltexte ermöglichen Versionsverwaltungssysteme, die auch Funktionen zum Vergleich von Programmquelltexten oder zur Zusammenführung verschiedener Versionen beherrschen. Die Unterstützung der stark ausgeprägten Gruppenfunktionen erfüllen CSCW-Software und Workflowsysteme (siehe Kapitel 0) besser als Archivsysteme.

4.4 Zielsetzung der Diplomarbeit

Die Systemvisionen zeigen, daß durch ein Archivsystem viele der in Kapitel 3 genannten Verbesserungsmöglichkeiten realisiert werden können. Die dort genannten Faktoren werden durch den Einsatz eines Archivsystems berücksichtigt.

In dieser Arbeit wird Archivsystem entwickelt, welches die in den Systemvisionen beschriebene Funktionalität besitzt. Um Ergebnisse aus Untersuchungen von ähnlichen Bereichen nutzen zu können, wird eine Betrachtung anderer konzeptioneller Ansätze aus dem Bereich Dokumenten-Management-Systeme durchgeführt. Mit den gewonnenen Erkenntnissen über Archivsysteme wird dann das Archivsystem entwickelt.

Ziel der Arbeit ist es, bei der Entwicklung des Archivsystems durchgängig eine geeignete objektorientierte Softwareentwicklungsmethode anzuwenden. Dabei soll der Weg von der ersten Anforderungsanalyse bis zu einer prototypischen Implementierung dokumentiert werden.

Da Archivsysteme komplexe Softwaresysteme sind, und die Entwicklung eines vollständigen Archivsystems den Rahmen dieser Arbeit sprengen würde, wird nur die Spezifikation und die prototypische Implementierung elementarer Funktionen durchgeführt. Elementare Funktionen sind hier die Funktionen, welche die Grundideen der Systemvisionen umsetzen. Außerdem soll die prototypische Implementierung die Architektur des Archivsystems veranschaulichen.

Funktionen, die nicht unmittelbar zu Verständnis der Grundideen notwendig sind, werden dagegen nicht näher betrachtet. Dies umfaßt unter anderem die Funktionen zur Installation, Administration und Wartung des Systems.

KAPITEL 5

5 Dokumenten-Management-Systeme

Über Archivsysteme hinaus gibt es noch weitere Systeme, die sich mit der Behandlung von Dokumenten beschäftigen. Diese Systeme werden zusammenfassend Dokumenten-Management-Systeme genannt und bilden hinsichtlich ihrer Funktionalität und ihrer Anwendungsgebiete verschiedene Schwerpunkte. Die verschiedenen konzeptionellen Schwerpunkte werden in diesem Kapitel diskutiert, um Komponenten der verschiedenen Dokumenten-Management-Systeme dahingehend bewerten zu können, ob eine Übernahme bestimmter Komponenten bei der Erstellung des zu entwickelnden Archivsystems hilfreich sein kann.

Dokumenten-Management-Systeme haben nach [BuMa94] die *Erfassung, Erstellung, Verwaltung, Weiterleitung, Ablage, Archivierung,* das *Abrufen* und das *Suchen von Dokumenten* als Aufgabe. Sie werden eingesetzt, um „die Produktivität durch eine Verkürzung der Dokumentendurchlaufzeit und eine sofortige Bereitstellung notwendiger Information zu erhöhen“[BuMa94]. Arbeitsabläufe in Büros (Bürovorgänge) werden unterstützt, und durch diverse Hilfsmittel sogar optimiert.

Dokumenten-Management-Systeme unterstützen in der Regel nicht alle oben genannten Aufgaben im gleichen Maße, sondern bilden Schwerpunkte hinsichtlich der Funktionalität. „Bezüglich der Schwerpunkte ihrer Funktionalität können Dokumenten-Management-Systeme in drei Gruppen eingeteilt werden, nämlich in *Archivierungsysteme, Retrieval*- oder Recherchesysteme und *Workflow*- oder Vorgangssteuerungssysteme.“ [BuMa94]

Diese drei Gruppen werden im folgenden erläutert.

5.1 Archivsysteme

„Archivierungssysteme legen den Schwerpunkt auf die langfristige Ablage von Dokumenten. Sie geben Hilfestellung bei der Erfassung, Digitalisierung, Attributierung und langfristigen Speicherung von Dokumenten. Archivierungssysteme eignen sich insbesondere für die langfristige Ablage von Massenbelegen (z.B. Buchungsabschnitte) und Formularen.“ [BuMa94, S.14f]

Archivierungssysteme oder kurz Archivsysteme sind also Systeme zur Archivierung und Verwaltung großer Mengen gleichstrukturierter Dokumente. Die archivierten Dokumente beinhalten langlebige Informationen, sie werden aus rechtlichen oder geschäftlichen Gründen für einen längeren Zeitraum archiviert. Wegen der großen Datenmenge und der geringen und teuren Kapazität von Magnetplatten werden die Daten häufig auf billigere Medien ausgelagert. Dies können Bänder, magnetische oder optische Platten oder beschreibbare Compact Discs (CD-R) sein. Die Entscheidung für den Einsatz bestimmter Speichermedien hängt von Faktoren wie dem Preis und der für eine sinnvolle Anwendung benötigten Geschwindigkeit ab. Im allgemeinen sind schnellere Medien auch teurer im Preis.

Haupteinsatzgebiet von Archivsystemen ist bevorzugt die Archivierung von ursprünglich auf Papier vorhandenen Dokumenten, wie beispielsweise Korrespondenz, Akten oder Zeichnungen. Es ist aber auch eine Archivierung von Gesprächsmitschnitten, Video- und Tondokumenten denkbar. Diese Dokumente werden vor der Archivierung mit geeigneten Mitteln in eine Form gebracht, die dem Computer eine Verwaltung der Dokumente ermöglicht. Bereits mit Hilfe eines Computers erstellte Dokumente können hingegen ohne Umwandlung auf einfachere Weise archiviert werden.

Die zu archivierenden Dokumente sind zum Zeitpunkt der Archivierung schon sehr weit in ihrer Bearbeitung fortgeschritten, so daß eine geringe Zugriffs- und Änderungsrate in der weiteren Zeit besteht. Solche Dokumente mit geringer Zugriffs- und Änderungsrate sind zum Beispiel Buchungsbelege, die nach ausgeführter Buchung noch eine lange, rechtlich vorgeschriebene Aufbewahrungszeit besitzen, um im Einzelfall eine Überprüfung zu ermöglichen. Auch in anderen Bereichen müssen Dokumente aus juristischen Gründen über Jahre hinweg aufbewahrt werden.

Das Problem der Archivierung einer großen Dokumentenmenge kann klassisch durch Archivierung der Papierdokumente in einem Aktenarchiv gelöst werden. Auch ein Aktenarchiv kann durchaus, wenn es bestimmte Voraussetzungen erfüllt, als Archivsystem angesehen werden. Archivsysteme müssen also nicht zwangsläufig durch ein Computerlösung realisiert werden, aber eine Computerlösung ist mit vielen Vorteilen verbunden, wie im Kapitel 3 erläutert wurde. Der Platzbedarf der Papieroriginale ist in der Regel sehr hoch. Ein Wiederfinden einer bestimmten Information ist mit einem sehr hohen Zeitaufwand verbunden. Wenn mit Hilfe eines Archivsystems die Informationen in geeigneter Weise in eine digitale Form überführt werden und auf Datenträgern gespeichert werden, ist der Platzverbrauch durch die Dokumente geringer, denn die Papieroriginale können, wenn rechtliche Gründe dem nicht entgegenstehen, nach der Archivierung vernichtet werden. Weiterhin ist der Zugriff auf die Dokumente einfacher und durch die Computerunterstützung nicht mehr so zeitintensiv.

Zusätzlich zu den Daten mit einer geringen Änderungs- und Zugriffsrate gibt es noch Daten mit einer größeren Änderungs- und Zugriffsrate, also zum Beispiel Daten aus noch laufenden Projekten oder Vorgängen. Auch hier kann eine Computerlösung viele Vorteile einbringen. Archivsysteme sind dazu aber nicht geeignet, da eine ausreichende Unterstützung von Funktionen zur Gruppenbearbeitung und Vorgangsunterstützung fehlen. Die möglichen Lösungen für die Behandlung solcher Dokumente mit einer großen Änderungs- und Zugriffsrate werden weiter unten unter dem Stichwort Workflow-Systeme in Kapitel 5.3 diskutiert.

5.2 Retrieval-Systeme

„Retrieval-Systeme legen den Schwerpunkt auf die Bereitstellung von Informationen und Dokumenten, die in Datenbanken und Archiven abgelegt sind. Sie greifen dazu auf den Inhalt der Dokumente zu und eignen sich daher besonders für Aufgaben, bei denen der Zugriff auf Informationen und nicht auf ein spezifisches Dokument notwendig ist.“ [BuMa94, S.15]

Voraussetzung für den Zugriff auf die Dokumenteninhalte ist eine Aufbereitung der Informationen in einer Form, die dem Computer eine Textsuche ermöglicht. Wurde ein Dokument mit Textinhalt bereits mit Hilfe eines Computers erzeugt, stellt der Zugriff auf die Dokumenteninhalte kein Problem dar. Wird jedoch ein auf Papier vorhandenes Dokument durch einen Scanner erfaßt, muß eine Aufbereitung der Informationen des Dokumentes erfolgen. Diese Aufbereitung wird durch eine optische Zeichenerkennung (Optical Character Recognition - OCR) erledigt. Bei dieser Aufbereitung werden die graphischen Informationen des gescannten Dokumentes in Textinformationen umgesetzt.

Der Zugriff auf die Dokumenteninhalte ermöglicht eine Suche nach Textelementen im gesamten Dokumententext aller gespeicherten Dokumente. Diese Suche wird auch Volltextsuche genannt.

Die Funktionalität der optischen Zeichenerkennung wird in der Regel durch ein Archivsystem nicht abgedeckt.

5.3 Workflow-Systeme

„Workflowsysteme sind Computersysteme, die den Arbeitsfluß zwischen beteiligten Stellen entsprechend einer zuvor definierten Prozedur organisieren und kontrollieren. Sie haben ihren Ursprung auf der einen Seite in den Dokumentenarchivierungssystemen und auf der anderen Seite in den integrierten Bürosystemen. Sie bieten Funktionen zur Gruppenbearbeitung an und reichen bearbeitete Dokumente automatisch gemäß einem zuvor vereinbarten Ablauf weiter.“ [BuMa94, S.28]

Der Arbeitsablauf zwischen verschiedenen Stellen gemäß einem zuvor definierten Ablauf wird auch kurz „Vorgang“ genannt.

Workflowsysteme legen also den Schwerpunkt auf die Unterstützung von Vorgängen und unterstützen Aufgaben, bei denen mehrere Personen an der Lösung beteiligt sind.

Workflowsysteme sind sehr komplex und bestehen im allgemeinen aus verschiedenen Komponenten, den Workflowkomponenten (siehe [Jabl95], [Jabl95b], [Stor95], [WfMC94]). Da Workflowkomponenten später bei der Diskussion der Einordnung des in dieser Arbeit entwickelten Archivsystem noch eine Rolle spielen werden, folgt hier eine detaillierte Betrachtung dieser Workflowkomponenten.

5.3.1 Workflowkomponenten

Die ein Workflowsystem auszeichnenden Komponenten können in die fünf Gruppen Modellierung, Simulation, Steuerung, Administration und Integration aufgeteilt werden.

Modellierung

Die Modellierungskomponente erzeugt die Modelle für die Ablaufstruktur und die Aufbaustruktur, welche die Grundlage für die Abarbeitung von Vorgängen bilden.

Die Ablaufstruktur beschreibt den Ablauf eines Vorgangs. Der Ablauf erfolgt anhand einer vorher definierten Vorgangsbeschreibung und wird über Zeiten, Reihenfolge und Bedingungen gesteuert.

Die Organisation eines Unternehmens wird durch die Aufbaustruktur beschrieben. Ein Mitarbeiter kann aufgrund seiner Kompetenzen verschiedene Arbeitsstellen innerhalb eines Unternehmens annehmen, und eine Arbeitsstelle kann von mehreren Mitarbeitern besetzt werden. Arbeitsstellen mit gleichen Kompetenzen und Funktionen werden in Rollen zusammengefaßt. Mitarbeiter können also mehrere Rollen innehaben, und eine Rolle kann durch mehrere Mitarbeiter besetzt sein. Rollen werden aufgabenbezogen zu Organisationseinheiten zusammengefaßt.

Simulation

Die Simulation ist nötig, um die Ablauf- und Aufbaumodelle auf ihre Korrektheit hin zu überprüfen. Dabei können verschiedene Ablaufalternativen getestet werden. Durchlaufzeiten und Ressourcenbedarf eines Vorgangs werden abgeschätzt und potentielle Engpässe ermittelt.

Steuerung

Die Ablaufsteuerung ist das Kernstück eines Workflowsystems und steuert die Bearbeitung der Vorgänge. Entsprechend des definierten Ablaufs des Vorgangs werden die Dokumente den zuständigen Stellen zur Bearbeitung vorgelegt. Bei Bedarf kann das Dokument zu einer Stelle weitergeleitet werden, um dort bearbeitet zu werden. Eine Wiedervorlage in Abhängig einer Fristüberschreitung oder eines Ereignisses ist ebenfalls möglich. Sind zur Bearbeitung eines Dokumentes externe Programme nötig, übernimmt die Steuerung die Programmaufrufe.

Administration

Im Rahmen der Administration kann ein Supervisor offene Vorgänge und Überlastung der Mitarbeiter kontrollieren und bei Problemen zur Behebung manuell Eingreifen, beispielsweise durch eine Änderung der Priorität eines Vorgangs.

Integration

Die Bearbeitung eines Vorgangs mit Hilfe eines Workflowsystems erfordert eine Integration bereits bestehender Anwendungsprogramme und eine in das Workflowsystem integrierte Nutzung von Kommunikationsdiensten. Dazu zählen beispielsweise die Möglichkeit zum Senden und Empfangen von elektronischer Post oder Faxdokumenten

5.4 Computer Supported Cooperative Work

Eine gegenüber Workflowsystemen weniger stark vorgangsorientierte und allgemeinere Vorgehensweise bei der Problemlösung bietet Computer Supported Cooperative Work, kurz CSCW. Darunter versteht man die gemeinsame computerunterstützte Bearbeitung eines Vorgangs durch mehrere Personen, unabhängig von der räumlichen oder zeitlichen Situation der Zusammenarbeit (vgl. [Teu95],[Gre88],[Lit95]). Die Initiative zur Bearbeitung geht dabei aktiv vom Benutzer aus und wird nicht passiv wie bei einem Workflowsystem durch das System vorgegeben. Hilfsmittel dabei sind unter anderem Kommunikationsdienste und die Unterstützung der gleichzeitigen Bearbeitung eines Dokumentes durch mehrere Personen.

5.5 Abgrenzung und Einordnung

In dieser Arbeit wird ein Softwaresystem mit einem Schwerpunkt für die elektronische Archivierung entworfen und spezifiziert. Zur Erfüllung von Aufgaben, die über die eines Archivsystem hinausgehen, die aber eine Rolle in den Szenarien und Systemvisionen spielen, sollen Workflowkomponenten und Elemente von Retrievalsystemen näher betrachtet und im zur Abdeckung der Aufgaben nötigen Umfang übernommen werden.

Die komplette Übernahme aller Workflowkomponenten würde über das Ziel dieser Arbeit hinausgehen und für die Konstruktion eines Archivsystems auch nicht notwendig sein. Workflowsysteme in ihrer Komplexität würden über den Umfang dieser Arbeit hinausgehen. Als Abgrenzung sollen beispielsweise unter anderem die Workflowkomponenten Simulation und Administration nicht näher betrachten werden.

Als Veranschaulichung der Einordnung des zu entwickelnden Systems innerhalb der drei Gruppen Archivierungsysteme, Retrievalsysteme und Workflowsysteme soll folgendes Klassifizierungsmodell dienen:

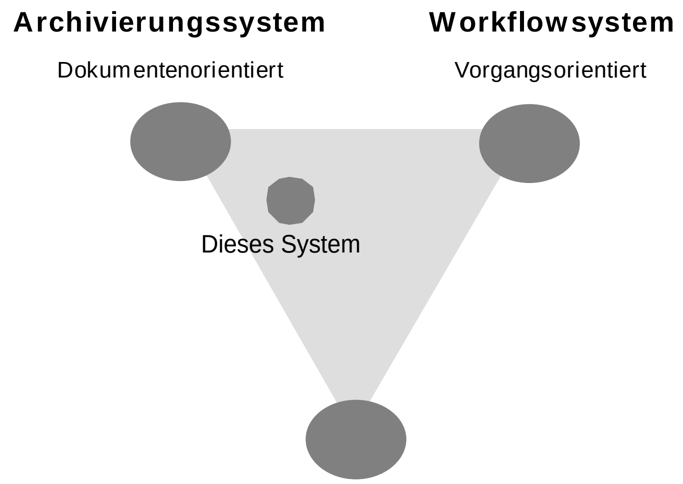

Abb. 6: Klassifizierungsmodell

In diesem Klassifizierungsmodell sind die drei Konzepte so angeordnet, daß sie ein Dreieck bilden. In diesem Dreieck lassen sich Systeme so anordnen, daß die konkreten Systemschwerpunkte aus ihrer Position abgelesen werden können. Archivierungssysteme legen den Schwerpunkt auf die Dokumentenverwaltung und arbeiten daher dokumentenorientiert. Workflowsysteme legen den Schwerpunkt auf die Vorgänge und arbeiten daher vorgangsorientiert. Retrievalsysteme schließlich legen den Schwerpunkt auf die Informationsfindung und arbeiten daher informationsorientiert.

Systeme, die den Schwerpunkt ausschließlich auf eines der Konzepte legen, liegen auf den Eckpunkten des Dreieckes. Je mehr ein System den Schwerpunkt in Richtung auf ein anderes Konzepte verschiebt, desto näher bewegt sich die Systemposition im Dreieck in Richtung des entsprechenden Eckpunkt. Die Position im Dreieck veranschaulicht also die Gewichtung der Konzepte eines Systems.

Aus den Diskussionen in den vorangegangenen Kapiteln, insbesondere in Kapitel 4, läßt sich die Position des Systems, welches in dieser Arbeit beschrieben wird, wie gezeigt darstellen. Aus dem Bereich der Retrievalsysteme ist das Element Volltextsuche, wie aus der Systemvision für das Szenario „Büro“ ersichtlich, auch für diese Arbeit interessant. Aus dem Bereich der Workflowsysteme sind die Elemente Aufbaustruktur und Weiterleitung für diese Arbeit interessant. Die Weiterleitung wird in der Systemvision für das Szenario „Konstruktionsbüro“ benutzt.

Daraus ergibt sich dann die dargestellte Positionierung des in dieser Arbeit entwickelten Archivsystems im Modell.

KAPITEL 6

6 Archivsysteme

In diesem Kapitel wird näher auf die Vor- und Nachteile von elektronischen Archivierungssystemen eingegangen. Außerdem werden die Voraussetzungen geklärt, unter denen Archivsysteme eingesetzt werden können, und welche Bedingungen dabei für die Dokumente gelten.

6.1 Vorteile elektronischer Archive

Eine elektronische Lösung für das Archivierungsproblem bietet verschiedene Vorteile, die sich aus den bereits in Kapitel 3 diskutierten Punkten ableiten lassen und durch ein Archivsystem umgesetzt werden können.

Wird ein Dokument aus dem Archivsystem benötigt, steht es ohne großen Zeitverlust am Bildschirm oder bei Bedarf als Reproduktion am Arbeitsplatz zur Verfügung. Gegenüber einem Aktenarchiv entfallen also die langen Zugriffszeiten, die sich aus den zurückzulegenden Wegstrecken ergeben. Durch stark verbesserte Suchmöglichkeiten ergeben sich kürzere Suchzeiten. Die Arbeitskraft eines Mitarbeiters kann durch ein Archivsystem sinnvoller und effektiver eingesetzt werden und auf die Kerntätigkeiten konzentriert werden.

Liegen die Daten in elektronischer Form vor, ist über Computerdatennetze eine Übertragung sehr einfach und schnell möglich. Dabei ist es dann unerheblich, ob dieses Netz ein lokales Netzwerk (LAN) oder ein Weitverkehrsnetz (WAN) ist. Je größer die räumliche Entfernung vom Lagerort zum Zielort ist, um so größer ist der Geschwindigkeitsvorteil, der sich durch eine Datenübertragung erzielen läßt, da lange Wegzeiten zum Transport der physikalischen Medien wie Papier oder Mikrofilm entfallen. Durch geeignete Vergleichs- und Überprüfungsmethoden ist die Möglichkeit gegeben, eine Konsistenz des Datenbestandes sicherzustellen. Dadurch wird erreicht, daß Dokumente nicht mehrfach archiviert werden oder daß verschiedene Dokumente nicht mit identischen Informationen attributiert werden.

Da Kopien von Dokumenten bedarfsgerecht erzeugt werden können, also erst zu dem Zeitpunkt, wo tatsächlich das Dokument in Papierform benötigt wird, ergibt sich eine Materialersparnis und damit ein Kostenvorteil.

Die durch die Ablage der Dokumente in Kellerarchiven und Lagerhallen benötigte Raumfläche wird nicht weiter benötigt und kann anderweitig genutzt werden. Wurden die Räume angemietet, ergibt sich daraus ein direkter Kostenvorteil. Durch den sehr geringen Platzbedarf der elektronischen Medien ist eine Unterbringung der Medien in den Büro- oder EDV-Räumen möglich.

Die Qualität der Ausgaben ist gleichmäßig gut, da immer Ausdrucke der gespeicherten Originale erzeugt werden und keine Kopien mehr von Kopien gemacht werden müssen. Auch nach langer Zeit werden die Ausdrucke noch in der ursprünglichen Qualität erzeugt, wohingegen Papiervorlagen vergilben können oder Faxe auf Thermopapier ausbleichen.

6.2 Nachteile elektronischer Archive

Naturgemäß bringt eine neu eingeführte Technik, hier das Archivsystem, auch eine Reihe von Nachteilen mit sich. Zur Neuanschaffung der Geräte wie Computer, Server, Scanner oder Drucker müssen Investitionen getätigt werden, und auch die Schulung der Mitarbeiter muß berücksichtigt werden. Außer den Kosten der Einführung entsteht auch eine Abhängigkeit von der Technik, denn ohne funktionierende Technik kann das Archivsystem nicht benutzt werden. Abhilfe schaffen hier verschiedene Ausfallkonzepte wie Notstromversorgung und doppelte Auslegung wichtiger Komponenten, aber ein Restrisiko läßt sich auch so nicht ausschließen.

Die traditionell gewachsenen Arbeitsabläufe bei der Dokumentenarchivierung müssen bei der Einführung eines Archivsystems neu überdacht und gegebenenfalls geändert werden. Dazu ist im Vorfeld der

Einführung Aufwendungen nötig, deren Kosten berücksichtigt werden müssen. Diese Aufwendungen führen im Idealfall zu effizienteren Arbeitsabläufen, die auch ohne den Einsatz eines Archivsystems bereits nutzbringend angewendet werden können.

Auf wechselnde Anforderungen oder auf neue oder geänderte Arbeitsabläufe und Organisationsstrukturen kann ein zu unflexibel gestaltetes System nicht ausreichend reagieren. Die dadurch entstehenden Probleme können nur durch Neuprogrammierung beispielsweise kostenintensiv gelöst werden .

Die Mitarbeiter müssen geschult werden, um die neue Technik bedienen zu können. Wurden vorher noch keine Computer eingesetzt, ist eine Vermittlung von Grundkenntnissen der Computerbedienung erforderlich. Vorteilhaft ist die Schulung zur Bedienung direkt in dem Unternehmen, welches das Archivsystem einsetzen möchte, und nicht an einem externen Schulungsort. So können Probleme bei dem Einsatz des Archivsystem in der konkreten Arbeitsumgebung rechtzeitig erkannt und gelöst werden.

Wurden Mitarbeiter nicht ausreichend geschult, weist das System Fehler auf oder schränkt das System die Mitarbeiter zu sehr ein, entstehen dadurch Akzeptanzprobleme seitens der Mitarbeiter. Die Akzeptanzprobleme können zur Folge haben, das die durch die Vorteile vorgegebenen Ziele nicht erreicht werden können.

6.3 Archivierungsvorgang

In diesem Abschnitt wird der Vorgang bei der Archivierung eines Dokumentes näher betrachtet. Dokumente werden zu Zeitpunkten archiviert, in denen sich das Dokument in einem Status befindet, in dem weitere Änderungen in absehbarer Zeit unwahrscheinlich sind. Eine Rechnung wird nach ihrer Überprüfung und Begleichung vorläufig nicht mehr benötigt und kann archiviert werden. Eine Bestellung wird nach Auslieferung der bestellten Produkte archiviert. Eine Zeichnung kann archiviert werden, nachdem sie freigegeben und dem Kunden übergeben worden ist.

Eine Archivierung aus Sicherheits- oder Wiederverwendbarkeitsaspekten kann ebenfalls sinnvoll erscheinen. Ein Sicherheitsaspekt könnte sein, daß der aktuelle Datenbestand einer Zwischensicherung bedarf, um ihn vor Systemausfällen oder unbeabsichtigtem Löschen zu schützen. Im Gegensatz zu regelmäßigen Backups kann der Benutzer diese Archivierung bei Bedarf selber anstoßen. Das Archivierungssystem übernimmt dann die Funktion der Datensicherung. Der Benutzer kann nach Systemproblemen oder bei Bedarf gezielt eine bestimmte Version wiederherstellen. Diese Funktionalität kann zwar auch durch den Einsatz eines reinen Versionskontrollsystems erreicht werden, diese Lösung bringt jedoch zwei nennenswerte Nachteile mit sich. Zum einen entstehen durch die Einführung einer eigenständigen Versionskontrolle Kosten, die bei Nutzung des Archivsystem zu dem Zwecke der Versionskontrolle nicht entstehen. Zum anderen ist die Integration der Versionskontrolle gewährleistet, während bei einer Einführung einer eigenständigen Versionskontrolle die integrierte Nutzung mit dem Archivsystem gelöst werden muß. Ein Wiederverwendbarkeitsaspekt könnte sein, daß das Dokument zwar noch geändert und damit erweitert wird, aber mehrere Dokumente aus dieser gemeinsamen Grundlage erstellt werden können. Dieses Dokument kann dann als Mustervorlage dienen, ohne selbst ein verwendbares Dokument zu sein.

In der Regel wird demnach der Benutzer entscheiden, wann ein Dokument archiviert werden soll, um dann den Archivierungsvorgang anstoßen.

Der gesamte Archivierungsvorgang läßt sich in einen externen und einen internen Teilvorgang aufteilen. Im externen Bereich, auf den das Archivsystem keinen Einfluß hat, findet die komplette Erstellung eines Dokumentes statt. Der Arbeitsablauf dabei liegt ebenfalls im externen Teilbereich und damit außerhalb der Kontrolle des Archivsystems. Erst nach erfolgter Erstellung kann das entstandene Dokument in ein Archiv übernommen werden, die weiteren Aktionen finden dann im internen Teilbereich statt. Zur Veranschaulichung dient folgende Abbildung (Abb. 7: Vorgangsbereiche):

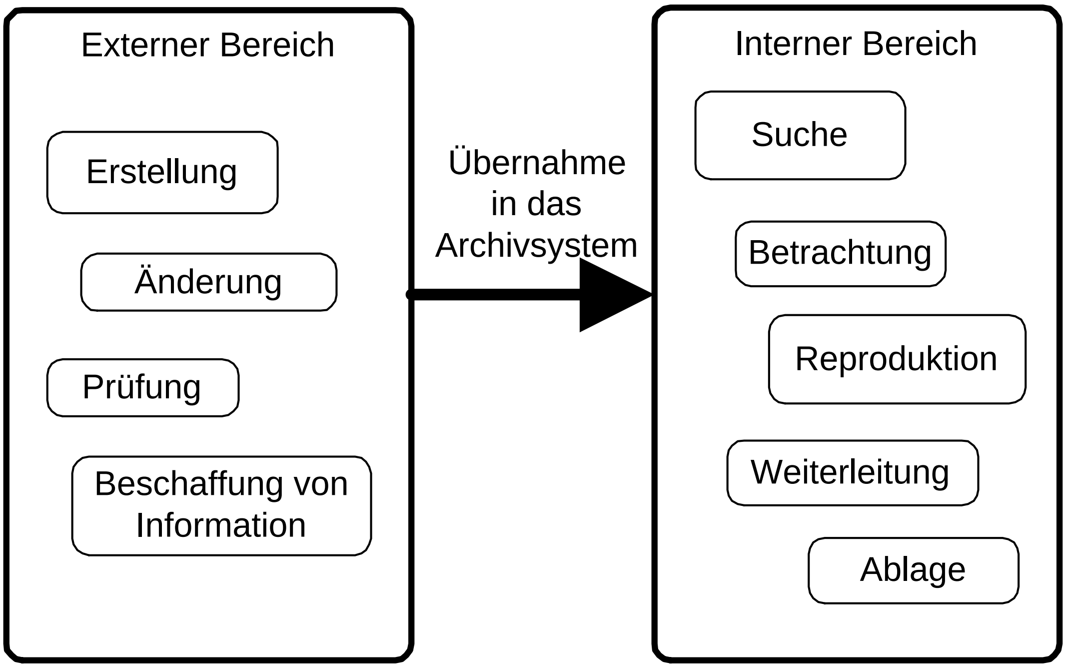

Abb. 7: Vorgangsbereiche

6.4 Anerkennung der Archivierung

Bei der Archivierung von Papieroriginalen in einem Archivsystem stellt sich die Frage, ob und wann die Papieroriginale vernichtet werden können. In vielen Bereichen gibt es eine gesetzliche Aufbewahrungs- und Nachweispflicht für Daten. Dabei sind eine Vielzahl von Bestimmungen im Einzelfall zu betrachten, so daß hier nur eine kurzer Überblick gegebenen wird.

Im Handels- und Steuerrecht wird *„eine ordnungsgemäße, qualifizierte Ablage und Aufbewahrung der Nachweise“* gefordert (siehe [GSZ93]). Die Entscheidung, ob dies zutrifft, wird im Einzelfall von dem Steuerprüfer oder Wirtschaftsprüfer beurteilt. Grundlage dazu ist, daß die notwendigen Unterlagen *„problemlos, zeitnah und in der erforderlichen Reihenfolge vorgeführt werden“* können. *„Problemlos“* heißt, daß die Unterlagen mit angemessenem Aufwand vorgeführt werden können. *„Zeitnah“* heißt, daß die Unterlagen in angemessener Zeit vorgeführt werden können. Das Ablageverfahren muß zusätzlich eine *„bildliche Übereinstimmung“* von reproduzierter Abbildung und Papieroriginal sicherstellen.

Im Zivilrecht hingegen gilt nur das Originaldokument als stichhaltiges Beweismittel. Andere Ablageverfahren haben einen geringeren Beweiswert und unterliegen der *freien richterlichen Beweiswürdigung.* Die Einschätzung der Glaubwürdigkeit der Reproduktion bleibt dem Richter also im Einzelfall überlassen.

Die Entscheidung, ob ein archiviertes Dokument nach erfolgter Archivierung im Original vernichtet werden darf, hängt also stark vom jeweiligen Einsatzgebiet ab und muß im Einzelfall sorgfältig geprüft werden. Eine Nutzung eines Archivsystems kann aber auch dann sinnvoll sein, wenn das Archivsystem die Nachweispflicht für Daten nicht erfüllen kann. Die Vorteile durch Zeit- und Materialersparnis und Qualitätsgewinn können trotzdem genutzt werden.

6.5 Anforderungen an Dokumente als mögliche Archivdaten

Um Dokumente mit einem Archivsystem bearbeiten zu können, müssen die Dokumente bestimmte Anforderungen erfüllen. Nicht alle Dokumente können sofort übernommen werden: sie benötigen eine Vorbehandlung. Grund dafür ist, daß zur Bearbeitung eines Dokumentes mit einem Computer dieses in der computergerechten äußeren Form einer Datei vorliegen muß. Computersysteme legen ihre Daten aus physikalischer Sicht gesehen in sogenannten Dateien ab. Diese Dateien können verschiedenen Inhalt haben, und werden unabhängig davon vom Computersystem gleich behandelt. Erst Anwendungsprogramme unterscheiden und interpretieren dann die Inhalte der Dateien, zum Beispiel als Textdatei oder Datenbank. Auch ein Archivsystem legt die Dokumente in Dateien ab oder nutzt dazu ein Datenbanksystem. Selbst ein Datenbanksystem legt letztendlich die Daten in Dateien ab.

Mittels eines Computerprogrammes erstellte Dokumente bestehen aus einer oder mehreren Dateien. Beinhaltet ein Textdokument beispielsweise eine Grafik, kann diese Grafik genauso wie das Textdokument als eigene Datei vorliegen. Das Dokument besteht also aus zwei Dateien. Das Archivierungssystem muß zur Übernahme eines Dokumentes in das Archivsystem Zugriff auf alle zum Dokument gehörigen Dateien haben.

Dokumente in Papierform und andere analoge Dokumente liegen noch nicht in einer computergerechten Form vor. Diese Dokumente müssen erst mit geeigneten Hilfsmitteln in die computergerechte äußere Form einer Datei überführt werden.

Die Behandlung von nicht computergerechten und computergerechten Dokumenten wird in den nächsten beiden Abschnitten behandelt.

6.5.1 Nicht computergerechte Dokumente

Unter nicht computergerechten Dokumenten werden hier Dokumente verstanden, die in einer Form vorliegen, die nicht direkt vom Computer verarbeitet werden kann. Dies kann ein Papierdokument, ein Foto, ein Video oder eine Gesprächsaufzeichnung sein. Konkrete nicht computergerechte Dokumente aus den Szenarien sind beispielsweise in Papierform vorliegende Rechnungen oder Zeichnungen. Um solche nicht computergerechten Dokumente mit einem elektronischen Archivierungssystem bearbeiten zu können, ist eine Umformung in eine computergerechtes Form notwendig. Zu dieser Umwandlung wird in der Regel spezielle Hardware benötigt.

Einen Überblick über diese Hardware gibt die folgende Tabelle:

Dokumentenart	**Hardware**
Papierdokument, Kleinformat	Kleinformatscanner
Papierdokument, Großformat	Großformatscanner
Mikrofilmkarte	Mikrofilmkartenscanner
Grafik, Fotografie	Farbscanner
Video	Framegrabber
Ton	Analog-Digital-Wandler

Tab. 2: Hardware zur Umwandlung

6.5.2 Computergerechte Dokumente

Unter computergerechten Dokumenten werden hier Dokumente verstanden, die in einer Form vorliegen, die direkt vom Computer verarbeitet werden kann. Computerkonforme Dokumente wurden in der Regel mit Hilfe eines Computers erstellt und können zum Beispiel Textdokumente, Grafiken oder gesendete Faxe sein. Archiviert werden können prinzipiell alle Dateien auf einem Computersystem, für die eigentliche Archivierung ist es daher irrelevant, mit welcher Applikation ein Dokument erstellt wurde. Um eine spätere Nutzung des Dokumentes zu ermöglichen, ist es allerdings notwendig, zusammen mit dem Dokument eine Information über das Computerprogramm, mit dem das Dokument erstellt worden ist, zu archivieren. Der Name und die Versionsnummer dieses Programms kann beispielsweise zu den Dokumenteninformationen aufgenommen werden.

6.6 Zusammenfassung

Der Einsatz von Archivsystemen bietet verschiedene Vor- und Nachteile, die im Einzelfall abgewägt werden müssen. Bei der Einführung eines Archivsystems müssen viele Punkte beachtet werden, unter anderem auch die Frage nach einer möglichen Vernichtung der Originaldokumente.

In einem Archivsystem ist die gleichzeitige Archivierung von ursprünglich nicht computergerechten und computergerechten Dokumenten gemeinsam möglich. Die beiden Dokumentenarten können in einem Archiv zusammengefaßt werden, wenn sie logisch zusammengehörige Dokumente umfassen. In einem Archiv kann ausgehender Schriftverkehr als Textdatei gemeinsam mit ursprünglich in Papierform vorliegender Post archiviert werden, oder in einem Archiv können mit einem CAD-System erstellte Zeichnungen gemeinsam mit manuell auf Papier erstellten Zeichnungen archiviert werden.

KAPITEL 7

7 Vorgehensweise bei der Systemkonstruktion

7.1 Allgemein

Ein größeres Softwareprojekt benötigt eine Methodik bei seiner Entwicklung. Es gibt viele Softwareentwicklungsmethoden, und in letzter Zeit gewinnen die objektorientierten Methoden immer mehr an Bedeutung. Die ursprünglich zur Konstruktion des Systems dieser Arbeit vorgesehene neue objektorientierte Methode „Unified Method“ als Vereinigung der Methoden von Booch (OOAD, vgl. [Boo94]) und Rumbaugh (OMT, vgl. [Rumb91]) war leider zu Beginn der Konstruktionstätigkeit entgegen den ursprünglichen Erwartungen in ihrer Definition noch nicht weit genug fortgeschritten und stand nur als Draft in der Version 0.8 [BoRu95] zur Verfügung, der nur die Spezifikation der einzelnen Modelle umfaßt und den eigentlichen Entwicklungsprozeß noch nicht beschreibt (siehe auch [MüSc96]). Grund zu der Verzögerung ist vielleicht auch die Tatsache, daß außer den beiden Methoden von Booch und Rumbaugh auch noch eine Methode von Jacobson in diese „Unified Method“ integriert werden sollte. Eine nähere Beschäftigung mit dieser Methode führte dann zu der Entscheidung, sich in dieser Arbeit ganz auf diese Methode „Use-Case-Driven-Approach“ von Jacobson et al. [JCJÖ92] zu konzentrieren.

7.2 Einführung in OOSE

Die Methode von Jacobson stellt sogenannte „Use-Cases“, also konkrete Anwendungsfälle, in den Mittelpunkt der Softwareentwicklung. Als Anforderungen an das System werden die Use-Cases aufgestellt und die von außen auf das System einwirkenden Faktoren in Form von Akteuren identifiziert. Aus den dabei gewonnen Erkenntnissen werden Modelle entwickelt, die verschiedene Aspekte der Anwendung darstellen.

Um einen Überblick über die Methode zu bekommen, folgt hier nun eine kurze Einführung.

Bei der Methode „Use-Case-Driven-Approach“ finden während der Systementwicklung verschiedene Prozesse statt. Diese Prozesse sind der Analyseprozeß, der Konstruktionsprozeß und der Testprozeß. Im Verlauf eines Prozesses werden Modelle in andere Modelle transformiert. Ein Modell stellt dabei eine spezielle Systemsicht dar.

Am Anfang des Softwareentwicklungsprozesses steht der Analyseprozeß, in dessen Verlauf das Anforderungsmodell und das Analysemodell entstehen. In das Anforderungsmodell fließen die funktionalen Anforderungen an das System aus der Sicht der späteren Benutzer ein. Das Anforderungsmodell wird unter Verwendung des Use-Case-Modells aufgestellt. Ist das Anforderungsmodell gegenüber weiteren Änderungen stabil, wird es in ein einfacher zu pflegendes Analysemodell transformiert. Das Analysemodell ist noch unabhängig von der späteren konkreten Implementationsumgebung. Dadurch beschreibt das Analysemodell eine gegenüber späteren Umgebungsänderungen robuste Struktur, da sich Umgebungsänderungen nicht auf dieses Modell auswirken.

Unter Verwendung des fertiggestellten Analysemodell folgt der Konstruktionsprozeß. Der Konstruktionsprozeß liefert das Entwurfsmodell und das Implementationsmodell. Das Entwurfsmodell verfeinert die Objektstruktur des Analysemodells unter Berücksichtigung der konkret vorgegebenen Implementationsumgebung. Bei Erstellung des Entwurfsmodells werden erste die Implementation betreffende Entscheidungen getroffen. Wenn das Entwurfsmodell weit genug verfeinert wurde, wird daraus das Implementationsmodell entwickelt, welches letztendlich den kommentierten Programmcode enthält. Durch diesen Programmcode wird das System implementiert.

Abschließend wird im Rahmen des Testprozesses das Testmodell entwickelt, um das entstandene System zu überprüfen. Anhand verschiedener Testmethoden wird das System verifiziert und validiert. Bei der

Verifizierung wird überprüft, ob das System korrekt arbeitet; bei der Validierung wird überprüft, ob das System entsprechend den ursprünglichen Anforderungen korrekt entworfen wurde. Das Testmodell umfaßt auch die Dokumentation von Testspezifikationen und Testergebnissen.

Die folgende Abbildung (Abb. 8: Die Prozesse von OOSE) zeigt die Prozesse und ihre Modelle in einer Übersicht. Zu jedem der drei Schritte Analyseprozeß, Konstruktionsprozeß und Testprozeß sind die Unterschritte gezeigt sowie die Modelle, die diese Schritte produzieren. Die einzelnen Schritte sind in ihrem logischen Ablauf angegeben. Die zeitliche Reihenfolge ist aber nicht festgelegt, und im Laufe des Softwareentwicklungsprozesses kann wieder zu einem Schritt zurückgegangen werden, um zum Beispiel iterativ neue Erkenntnisse in bereits behandelte Modelle einfließen zu lassen. Diese Möglichkeit wird durch die bidirektionalen Pfeile veranschaulicht.

Zu Beginn der Abschnitte, die sich mit den Softwareentwicklungsprozessen beschäftigen (Kapitel 8.1 „Anforderungsmodell„, Kapitel 8.2 „Analysemodell„ und Kapitel 9 „Konstruktion des Archivsystems„) werden die entsprechenden Prozesse und Modelle noch einmal etwas detaillierter behandelt.

Für eine tiefergehende Betrachtung der Softwareentwicklungsmethode „Use-Case-Driven-Approach“ sei auf das Buch von Ivar Jacobson et al. „Object-Oriented Software Engineering“ [JCJÖ92] verwiesen.

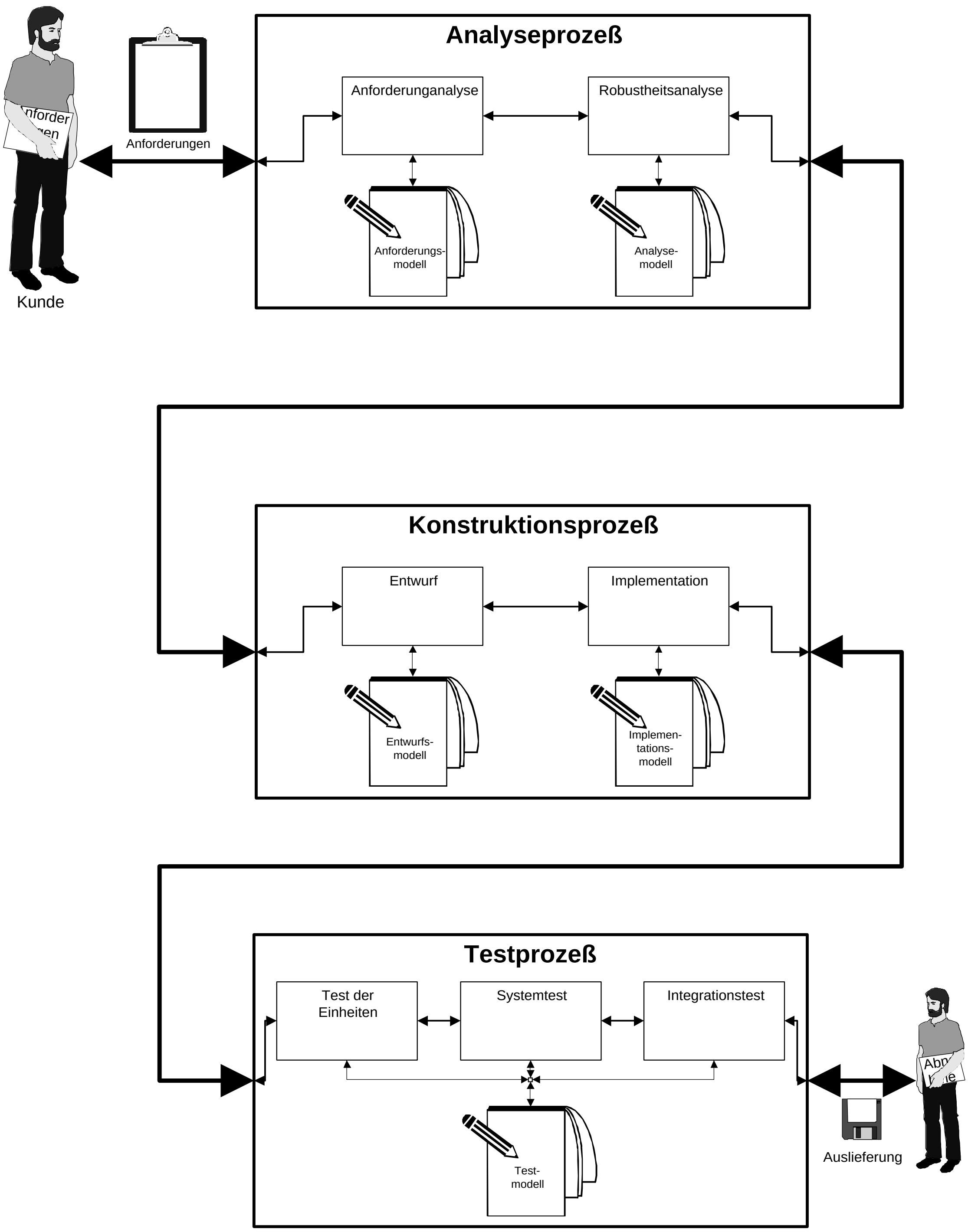

Abb. 8: Die Prozesse von OOSE

KAPITEL 8

8 Analyse des Archivsystems

8.1 Anforderungsmodell

Das Anforderungsmodell wird entwickelt, um das System in seinen Möglichkeiten abzugrenzen und die angebotene Funktionalität des Systems zu definieren. Das Anforderungsmodell besteht aus

- dem Use-Case-Modell,
- den Schnittstellenbeschreibungen und
- dem Problembereichsmodell.

Zur Beschreibung des Use-Case-Modells werden „Akteure“ und „Use-Cases“ benutzt. Ein Akteur interagiert von außen mit dem System und tauscht mit diesem Informationen aus. In der Regel sind dies die das System benutzenden Personen. Durch einen Akteur wird eine Rolle beschrieben, die eine bestimmte natürliche Person oder ein anderes System während der Nutzung des Systems einnehmen kann. Ein Akteur möchte bestimmte Tätigkeiten, deren Abläufe durch „Use-Cases“ beschrieben werden, mit dem System ausführen.

Um die Verwendung der Bezeichnungen der Akteure und Use-Cases besser hervorzuheben, wird die von Jacobson vorgeschlagene Notation benutzt. Namen von Akteuren werden mit dem Symbol „✌“ (z.B. „✌Büromitarbeiter“) und Namen von Use-Cases mit dem Symbol „✞“ verdeutlicht (z.B. „✞Anlegen von Nummernkreisen“).

Jacobson bietet in [JCJÖ92] über diese Kennzeichnung hinaus keine Methode an, die gefundenen Akteure oder Use-Cases gruppenweise strukturiert zu betrachten. Die „Einführung in die Anwendungsgebiete durch Szenarien„ in Kapitel 2 orientiert sich aber an verschiedenen Szenarien. Um nachvollziehen zu können, aus welchem Szenario die einzelnen Akteure oder Use-Cases ursprünglich stammen, und um die Übersichtlichkeit der Darstellung zu steigern, wird die Notation erweitert. Die Akteure und Use-Cases werden zu den Gruppen zusammengefaßt, welche die Szenarien widerspiegeln. Die Gruppennamen entsprechen den Szenariennamen und erhalten zur besseren Erkennbarkeit das Gruppensymbol „♠“ vorangestellt (z.B. „♠Büro“).

Die Akteure des Systems werden mit der eben erklärten Notation in strukturierter Form aufgelistet, bevor näher auf die Use-Cases eingegangen wird. Die in den Use-Cases benutzten Begriffe werden vorher im Abschnitt „Problembereichsmodell„ die verwendeten Schnittstellen danach im Kapitel „Schnittstellenbeschreibungen„ beschrieben.

8.1.1 Akteure des Systems

Anhand der obigen Szenarien (vgl. Kapitel 2 „Einführung in die Anwendungsgebiete durch Szenarien“ auf Seite 3) kann man die Akteure identifizieren. Eine Liste der das System benutzenden Akteure kann übersichtlich unter Anwendung der im Kapitel 8.1 beschriebenen Notation erstellt werden. Eine grafische Darstellung folgt im Anschluß an diese Liste (Abb. 9: Akteure des Systems). In dieser Grafik wird als zentrales Element das Archivsystem dargestellt, um welches alle beteiligten Akteure angeordnet sind.

Jacobson ermöglicht es in [JCJÖ92], abstrakte Akteure zu definieren. Einem abstrakten Akteur werden Tätigkeiten zugeordnet, die er an einen konkreten Akteur weitervererben kann. Von dieser Möglichkeit wird Gebrauch gemacht, wenn der abstrakte Akteur ✌**Sachbearbeiter** eingeführt wird, der die Tätigkeiten der konkreten Akteure ✌**Einkäufer**, ✌**Verkäufer** und ✌**Buchhalter** zusammenfaßt (siehe auch Abb. 10: Abstrakter Akteur "Sachbearbeiter").

♠*Büro:*

✌Büromitarbeiter

Der Büromitarbeiter erfaßt ein Dokument, gibt Daten dazu ein oder ändert Daten und kann einen Eintrag löschen.

✌Buchhalter

Der Buchhalter sucht ein Dokument, läßt es sich auf dem Bildschirm anzeigen und gibt es gegebenenfalls aus.

✌Einkäufer

Der Einkäufer sucht ein Dokument, läßt es sich auf dem Bildschirm anzeigen und gibt es gegebenenfalls aus.

✌Verkäufer

Der Verkäufer sucht ein Dokument, läßt es sich auf dem Bildschirm anzeigen und gibt es gegebenenfalls aus.

✌Sachbearbeiter

Der Verkäufer sucht ein Dokument, läßt es sich auf dem Bildschirm anzeigen und gibt es gegebenenfalls aus.

✌Archivar

Der Archivar verwaltet das Archivsystem und legt beispielsweise neue Benutzer an.

♠*Konstruktionsbüro:*

✌Projektleiter

Der Projektleiter leitet Projekte und legt dafür Nummernkreise für die Zeichnungen an.

✌Konstrukteur

Der Konstrukteur bearbeitet Zeichnungen und übergibt diese über das Archivsystem zur Freigabe an den Qualitätssicherungsmitarbeiter. Nicht freigegebene Zeichnung überarbeitet der Konstrukteur.

✌Qualitätssicherungsmitarbeiter

Der Qualitätssicherungsmitarbeiter überprüft Zeichnungen und entscheidet über die Freigabe. Nicht freigegebene Zeichnungen werden an den Konstrukteur zurückgeleitet.

✌Archivar

Der Archivar verwaltet das Archivsystem und legt beispielsweise neue Benutzer an.

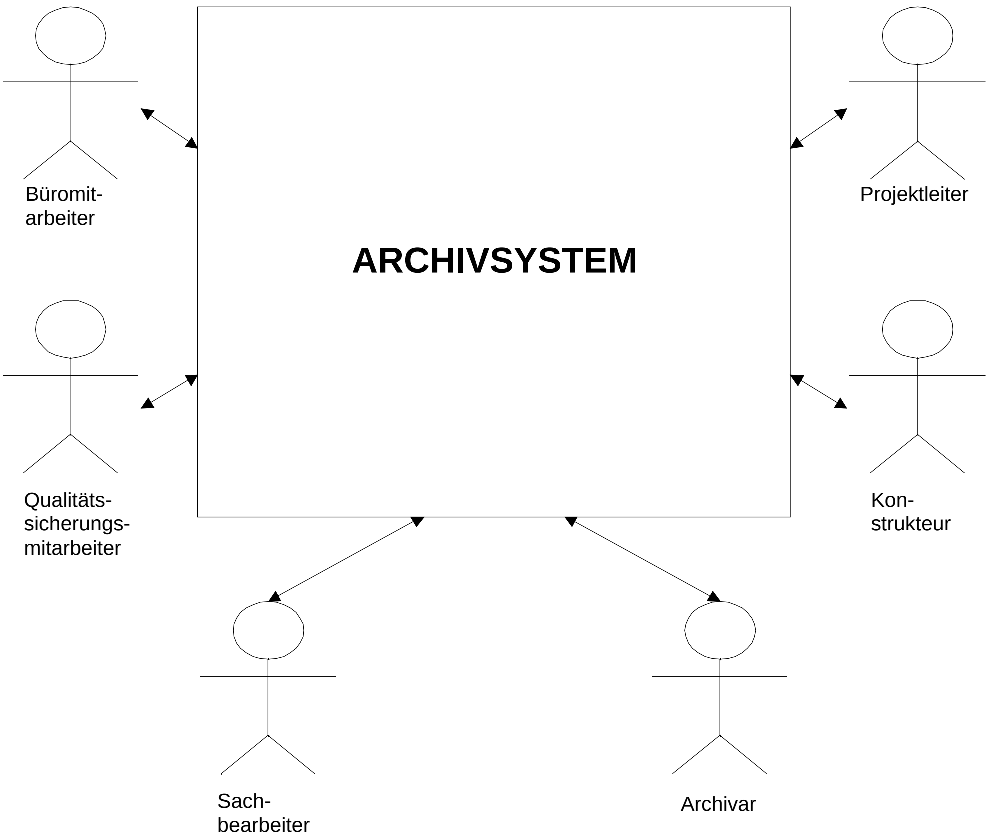

Abb. 9: Akteure des Systems

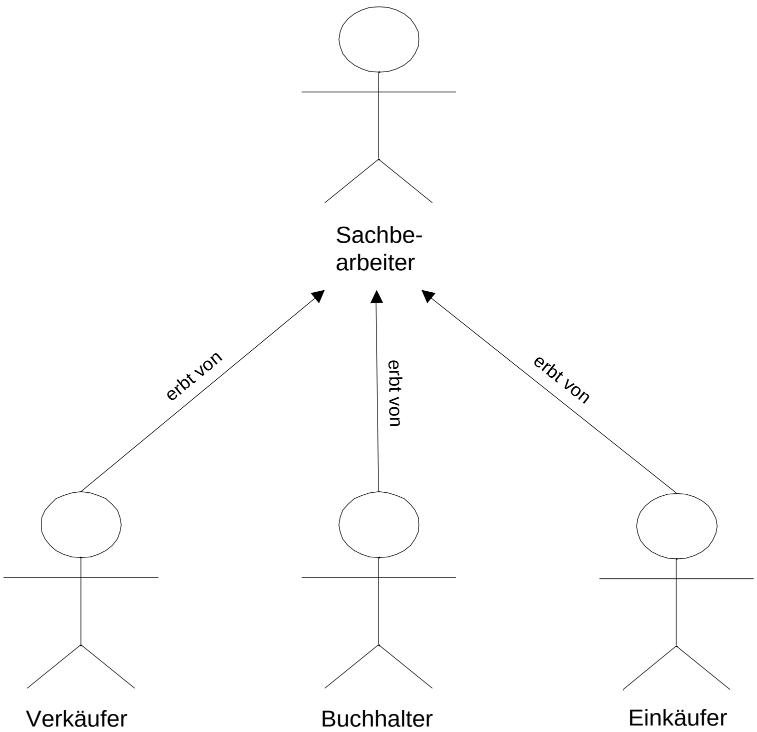

Abb. 10: Abstrakter Akteur "Sachbearbeiter"

8.1.2 Problembereichsmodell

Das Problembereichsmodell umfaßt die Problembereichsobjekte. Dabei handelt es sich um ein Glossar der Begriffe, die durch eine genaue Betrachtung der Szenarien und Systemvisionen gefunden werden. Begriffe, die dort häufig verwendet werden und die deshalb in diesem Archivsystem eine Rolle spielen, werden als Problembereichsobjekte in das Problembereichsmodell aufgenommen. Die Problembereichsobjekte haben dadurch eine direkte Entsprechung in dem Archivsystem. Sie werden in den Beschreibungen der Use-Cases wieder aufgegriffen und sorgen für eine Durchgängigkeit der verwendeten Begriffe.

Struktur der Auflistung

Die Struktur der Auflistung der Problembereichsobjekte orientiert sind an der in [JCJÖ92] auf Seite 168 vorgeschlagenen Vorgehensweise. Es werden Objektname, Attribute und Querverweise aufgeführt. Auf den Objektnamen, der das Problembereichsobjekt eindeutig benennt, folgt eine Beschreibung des Objektes. Danach werden die Attribute aufgelistet, die das Objekt näher beschreiben. Die Angabe der Attribute ermöglicht es, die Problembereichsobjekte näher zu spezifizieren und über die Begriffsbestimmung hinaus eine weitere Basis zu schaffen, die bei der Beschreibung der Use-Cases hilfreich ist. Es werden hier nur diejenigen Attribute aufgelistet, die für das Archivsystem wichtig sind. Die Anzahl der an einer Stelle möglichen Attribute wird durch die Kardinalität angegeben. Dazu wird die Schreibweise „Attribut[n...m]“ verwendet, um sprachliche Mißverständnisse durch die Pluralbildung auszuschließen. Ist die Kardinalität [1], gibt es genau ein Attribut. ist die Kardinalität [0...1], gibt es kein oder genau Attribut. Ist die Kardinalität [0...n], gibt es eine beliebige Anzahl von Attributen. Abschließend werden eventuell vorhandene Querverweise zu anderen Objekten genannt. Ein Querverweis wird dann angegeben, wenn es eine Beziehung zu einem anderen Objekt gibt, dieses aber kein Attribut des Objektes ist.

Es folgt die Liste der Problembereichsobjekte.

Liste der Problembereichsobjekte

Aktenordner

In einem **Aktenordner** werden **Dokumente** aufbewahrt.

Attribute

Dokument[0...n]

Querverweise

<Keine>

Archiv

Ein **Archiv** besteht aus **Dokumenten.**

Attribute

Dokument[0...n]

Querverweise

<Keine>

Archivar

Der **Archivar** ist eine **Rolle**, die ein **Benutzer** bei der Benutzung des **Archivsystems** einnehmen kann. Der **Archivar** verwaltet das **Archivsystem.**

Attribute

<Keine>

Querverweise

Archivsystem
Benutzer
Rolle

Archivsystem

Ein **Archivsystem** verwaltet ein **Archiv.**

Attribute

Archiv[1]

Querverweise

<Keine>

Aufbewahrungsfrist

Zeitpunkt, bis zu dem ein **Dokument** aufbewahrt werden muß.

Attribute

Zeitpunkt[1]

Querverweise

Dokument

Ausgabegerät

Gerät zur Ausgabe von Informationen, die in einem Computersystem vorliegen. Gängige Ausgabegeräte sind Bildschirm oder **Drucker**. Weiterhin sind unter anderem Hochleistungsdrucker, Plotter, Bandaufzeichnungsgeräte oder Belichter im Einsatz.

Attribute

Dokumentbeschreibung [1] der ausgebbaren Dokumente

Querverweise

Drucker

Auswahlliste

Eine **Auswahlliste** ist eine Liste von mehreren Möglichkeiten, aus denen eine Möglichkeit zur weiteren Bearbeitung ausgewählt werden kann.

Attribute

Auswahlmöglichkeiten[0...n]

Querverweise

<Keine>

Benutzer

Ein **Benutzer** ist eine konkrete Person, welche das **Archivsystem** benutzt. Der Benutzer wird zu Sitzungsbeginn anhand seines Namens und seines Paßworts identifiziert und kann eine **Rolle** auswählen, die er bei der Benutzung des Archivsystems einnimmt.

Attribute

Name[1]
Paßwort[1]
Rolle[1...n]

Querverweise

Archivsystem
Rolle

Bestellung

Eine **Bestellung** ist eine spezieller **Dokumententyp**. Zu einer **Bestellung** werden **Bestellungsdaten** gespeichert.

Attribute

Bestellungsdaten[1]

Querverweise

Dokumententyp

Bestellungsdaten

Bestellungsdaten speichern Informationen zu einer **Bestellung**.

Attribute

Bestelldatum[1]
Eingangsdatum[1]
Name des bestellten Artikels[1]
Bestellbetrag[1]
Name des Bestellers[1]

Querverweise

Bestellung

Betrachter

Ein **Programm**, mit dem ein **Dokument** betrachtet werden kann. Der Betrachter wird durch den Programmnamen identifiziert und wird durch eine Programmversion näher bestimmt.

Attribute

Programmbeschreibung[1]

Querverweise

<Keine>

Blatt

Eine **Zeichnung** besteht aus einem **Blatt** oder mehreren **Blättern.** Die Blätter werden durch eine eindeutige Blattnummer unterschieden. Der Blattinhalt wird durch die Beschreibung beschrieben.

Attribute

Blattnummer[1]
Beschreibung[1]

Querverweise

Zeichnung

Buchhalter

Eine **Rolle**, die ein **Benutzer** bei der Benutzung des **Archivsystems** einnehmen kann.

Attribute

<Keine>

Querverweise

Archivsystem
Benutzer
Rolle

Büro

Ort, an dem der **Büromitarbeiter** den **Schriftverkehr** bearbeitet.

Attribute

Büromitarbeiter[1]
Schriftverkehr[1]

Querverweise

<Keine>

Büromitarbeiter

Der **Büromitarbeiter** ist eine **Rolle**, die ein **Benutzer** bei der Benutzung des Archivsystems einnehmen kann. Der **Büromitarbeiter** bearbeitet den **Schriftverkehr** in einem **Büro**.

Attribute

Schriftverkehr[1]

Querverweise

Benutzer
Büro
Rolle

CAD-System

Ein Computersystem oder ein **Programm**, mit dessen Hilfe **CAD-Zeichnungen** erstellt und bearbeitet werden können.

Attribute

Programmbeschreibung[1]

Querverweise

CAD-Zeichnung

CAD-Zeichnung

Eine **Zeichnung**, die mit Hilfe eines **CAD-Systems** erstellt wurde.

Attribute

CAD-System[1]

Querverweise

Zeichnung

Compact-Disc

Datenträger, auf dem eine große Menge Daten gespeichert werden kann. Eine **Compact-Disc** kann nur einmal beschrieben werden.

Attribute

Datenträgerbenennung[1]

Querverweise

<Keine>

Datei

Daten werden in einem Computersystem in **Dateien** abgelegt, deren Verwaltung im allgemeinen das Betriebssystem übernimmt.

Attribute

Dateiname[1]
Speicherort[1]
Dateigröße[1]

Querverweise

<Keine>

Dateiauswahlliste

Liste aus den Namen von **Dateien**, aus denen einer zur weiteren Bearbeitung ausgewählt werden kann.

Attribute

Dateiname[0...n]

Querverweise

Datei

Dokument

Ein **Dokument** ist ein Medium mit Information (vgl. auch Kapitel 2.4.1 „Dokument“). Die textuelle Information des Dokumentes wird **Dokumententext** genannt. Dokumente können mit einem **Archivsystem** verwaltet werden.

Attribute

Information[1]
Medium[1]

Querverweise

Dokumententext
Archivsystem

Dokumententext

Der **Dokumententext** bestimmt den Inhalt eines **Dokumentes** und ist wichtig für die **Volltextsuche**.

Attribute

Text[1]

Querverweise

Dokument
Volltextsuche

Dokumententyp

Die unterschiedlichen Dokumentenarten werden durch **Dokumententypen** beschrieben. **Rechnung**, **Bestellung**, **Produktinformation** und **Zeichnung** sind **Dokumententypen**.

Attribute

Bezeichnung[1]

Querverweise

Rechnung
Bestellung
Produktinformation
Zeichnung

Drucker

Mit Hilfe eines **Druckers** kann ein mit einem Computersystem erstelltes **Dokument** auf Papier ausgegeben werden.

Attribute

Dokumentbeschreibung[0...n] der druckbaren Dokumente

Querverweise

Dokument

Einkäufer

Der **Einkäufer** ist eine **Rolle**, die ein **Benutzer** bei der Benutzung des Archivsystems einnehmen kann.

Attribute

<Keine>

Querverweise

Archivsystem
Benutzer
Rolle

Eintrag

Ein **Eintrag** umfaßt Informationen zu einem **Dokument** und das Dokument selbst. Die Informationen werden in den durch den **Eintragstypen** festgelegten **Eintragsdaten** gespeichert.

Attribute

Eintragsdaten[1]
Dokument[1]

Querverweise

Eintragstyp

Eintragsdaten

Die **Eintragsdaten** umfassen die Informationen zu einem **Eintrag**. Die Eintragsdaten bestehen aus den **Systemdaten** und den **Userdaten**. Die **Userdaten** werden durch den **Eintragstypen** festgelegt. Die Struktur der **Systemdaten** ist für alle **Eintragstypen** gleich.

Attribute

Systemdaten[1]
Userdaten[1]

Querverweise

Eintrag
Eintragstypen

Eintragsliste

Eine Liste von **Einträgen** zur Darstellung auf dem Bildschirm.

Attribute

Eintrag[0...n]

Querverweise

Eintragstyp

Durch den **Eintragstyp** wird beschrieben, welche **Eintragsdaten** zu einem **Dokument** gespeichert werden. Zu jedem **Dokumententyp** gibt es genau einen entsprechenden **Eintragstyp**.

Attribute

Name

Querverweise

Eintragsdaten
Dokument
Dokumententyp

Fensterbereich

Ein Teilbereich eines Fensters, beispielsweise des Hauptfensters.

Attribute

Inhalt[1]

Querverweise

Hauptfenster

Freigabe

Eine **Zeichnung**, die von einem **Qualitätssicherungsmitarbeiter** erfolgreich überprüft wurde, erhält die **Freigabe** zur Weitergabe an Kunden und zum Zugriff durch andere **Konstrukteure**.

Attribute

Freigabeentscheidung[1]
Benutzer[1]

Querverweise

Archiv
Konstrukteur
Qualitätssicherungsmitarbeiter
Zeichnung

Funktionsleiste

Über die **Funktionsleiste** werden die **Programmfunktionen** ausgelöst.

Attribute

Programmfunktion[0...n]

Querverweise

<Keine>

Hauptfenster

Das Fenster, das **Eintragsliste** und **Funktionsleiste** umfaßt.

Attribute

Eintragsliste[1]
Funktionsleiste[1]

Querverweise

<Keine>

Konstrukteur

Der **Konstrukteur** ist eine **Rolle**, die ein **Benutzer** bei der Benutzung des Archivsystems einnehmen kann. Ein **Konstrukteur** erstellt **Zeichnungen**.

Attribute

<Keine>

Querverweise

Archivsystem
Benutzer
Rolle
Zeichnungen

Konstruktionsbüro

Büro, in dem Konstruktionsergebnisse unter anderem in **Zeichnungen** festgehalten werden.

Attribute

Benutzer[0...n]
Zeichnung[0...n]

Querverweise

<Keine>

Kopie

Eine **Kopie** wird von einem **Dokument** gemacht.

Attribute

Dokument[1]

Querverweise

<Keine>

Kopiergerät

Mit Hilfe eines **Kopiergeräts** werden von einem **Dokument Kopien** erzeugt.

Attribute

Dokumentbeschreibung[1] der kopierbaren Dokumente

Querverweise

Dokument
Kopie

Mikrofilmkarte

Verfilmte Darstellung eines **Dokumentes**, die deutlich weniger Raum als das Original beansprucht.

Attribute

Aufbewahrungsort[1]

Querverweise

Dokument

Mikrofilmkartenrückvergrösserer

Gerät, um die verkleinerte Darstellung eines **Dokuments** auf einer **Mikrofilmkarte** wieder in Originalgröße als **Kopie** auf Papier auszugeben.

Attribute

Dokumentbeschreibung[1] der Dokumente, die zurückvergrößert werden können

Querverweise

Dokument
Kopie
Mikrofilmkarte

Notizzettel

Auf einem **Notizzettel** kann man Anmerkungen zu einem Dokument notieren. Anhand der vom **Qualitätssicherungsmitarbeiter** notierten Anmerkungen kann ein **Konstrukteur** seine **Zeichnung** überarbeiten.

Attribute

Inhalt[1]

Querverweise

Qualitätssicherungsmitarbeiter
Konstrukteur
Kopie
Zeichnung

Nummernkreis

Ein **Nummernkreis** wird von einem **Projektleiter** zu Projektbeginn erstellt. **Zeichnungsnummern** für **Zeichnungen** dieses Projektes werden ausschließlich aus diesem Nummernkreis vergeben, um eine Doppelvergabe auszuschließen. Ein Nummernkreis wird durch Startnummer und Endnummer eingegrenzt und mit dem Projektnamen benannt.

Attribute

Startnummer[1]
Endnummer[1]
Projektname[1]
Projektleiter[1]

Querverweise

Zeichnungsnummer
Zeichnung

Originaldatei

Datei auf einem Computersystem, welche die Informationen über ein **Dokument** bei der Übernahme in das **Archivsystem** enthält.

Attribute

Dateiname[1]
Speicherort[1]
Dateigröße[1]

Querverweise

Dokument
Archivsystem

Originaldokument

Das **Dokument**, welches in das **Archivsystem** übernommen wurde.

Attribute

Dokumentbeschreibung[1]

Querverweise

Dokument
Archivsystem

Papieroriginal

Liegt das **Originaldokument** in Papierform vor, wird diese Ursprungsversion auch **Papier-original** genannt.

Attribute

Dokumentbeschreibung[1]

Querverweise

Originaldokument

Pausmaschine

Mit Hilfe einer **Pausmaschine** werden **Kopien** von großformatigen **Zeichnungen** erzeugt, sie funktionieren ähnlich wie ein **Kopiergerät** für kleinformatige **Dokumente**.

Attribute

Dokumentbeschreibung[1] der pausbaren Dokumente

Querverweise

Dokument
Kopie
Kopiergerät
Zeichnung

Plotmanagementsystem

System, daß Plot- und Druckaufträge entsprechend der Auslastung und Verfügbarkeit von Plottern und **Druckern** verwaltet und abwickelt.

Attribute

Programmbeschreibung[1]
Dokumentbeschreibung[1] der druckbaren Dokumente

Querverweise

Drucker

Produktinformation

Eine **Produktinformation** ist eine spezieller **Dokumententyp**. Informationen zu einer **Produktinformation** werden in den **Produktinformationsdaten** gespeichert.

Attribute

Produktinformationsdaten[1]

Querverweise

Dokumententyp

Produktinformationsdaten

Produktinformationsdaten speichern Informationen zu einer **Produktinformation**.

Attribute

Name der Produktgattung[1]
Name des Produkts[1]
Produktpreis[1]
Beginn der Gültigkeit[1]
Ende der Gültigkeit[1]
Name des Lieferanten[1]

Querverweise

Dokumententyp

Programm

Mit einem Programm können Ein Programm wird durch eine **Programmbeschreibung** beschrieben.

Attribute

Programmbeschreibung[1]

Querverweise

<Keine>

Programmbeschreibung

Eine **Programmbeschreibung** beschreibt ein **Programm** und die **Dokumente**, die mit diesem **Programm** bearbeitet werden können.

Attribute

Programmname[1]
Programmversion[1]
Dokumentbeschreibung[0...n]

Querverweise

Dokument
Programm

Programmfunktion

Eine Funktion, die mit einem **Programm** ausgeführt werden kann. Eine Funktion kann zum Beispiel über die **Funktionsleiste** aktiviert werden.

Attribute

Funktion[1]

Querverweise

Funktionsleiste

Projektleiter

Der **Projektleiter** ist eine **Rolle**, die ein **Benutzer** bei der Benutzung des Archivsystems einnehmen kann. Ein **Projektleiter** leitet Projekte in einem **Konstruktionsbüro** und erzeugt unter anderem **Nummernkreise**.

Attribute

<Keine>

Querverweise

Archivsystem
Benutzer
Konstruktionsbüro
Nummernkreis
Rolle

Qualitätssicherungsmitarbeiter

Der **Qualitätssicherungsmitarbeiter** ist eine **Rolle**, die ein **Benutzer** bei der Benutzung des Archivsystems einnehmen kann. Ein **Qualitätssicherungsmitarbeiter** überprüft **Zeichnungen** auf Korrektheit und leitet die **Freigabe** ein.

Attribute

<Keine>

Querverweise

Archivsystem
Benutzer
Freigabe
Rolle
Zeichnung

Rolle

Ein **Rolle** beschreibt einen festgelegten Tätigkeitsbereich, den ein **Benutzer** bei der Benutzung des **Archivsystems** einnimmt. **Rollen** sind zum Beispiel **Büromitarbeiter**, **Konstrukteur** oder **Archivar**.

Attribute

Rollenname[1]

Querverweise

Benutzer

Rechnung

Eine **Rechnung** ist eine spezieller **Dokumententyp.** Informationen zu einer Rechnung werden in den **Rechnungsdaten** gespeichert.

Attribute

Rechnungsdaten[1]

Querverweise

Dokumententyp

Rechnungsdaten

Rechnungsdaten speichern Informationen zu einer **Rechnung**.

Attribute

Rechnungsdatum[1]
Eingangsdatum[1]
Rechnungsbetrag[1]
Kostenstelle[1]
Name des Rechnungsstellers[1]

Querverweise

Rechnung

Reproduktion

siehe **Kopie**.

Sachbearbeiter

Eine **Rolle**, die ein **Benutzer** bei der Benutzung des **Archivsystems** einnehmen kann. Sachbearbeiter sind **Verkäufer**, **Einkäufer** und **Buchhalter**.

Attribute

<Keine>

Querverweise

Archivsystem
Benutzer
Rolle
Verkäufer
Einkäufer
Buchhalter

Scanner

Gerät, mit dessen Hilfe ein **Papieroriginal** abgetastet und in eine computerverarbeitbare **Datei** umgewandelt wird.

Attribute

Beschreibung[1] der abtastbaren Papieroriginale

Querverweise

Datei
Papieroriginal

Schlüssel

Merkmal, anhand dessen ein **Eintrag** eindeutig identifiziert werden kann.

Attribute

Merkmal[1]

Querverweise

Eintrag

Schrank

Ein Schrank dient zur Aufnahme von **Aktenordnern** oder **Zeichnungen**.

Attribute

Aktenordner[0...n]
Zeichnung[0...n]

Querverweise

<Keine>

Schriftverkehr

Gesamtmenge der ein- und ausgehenden Post in Form verschiedener **Dokumente**.

Attribute

Dokument[0...n]

Querverweise

<Keine>

Suchkriterium

Kriterium, nach dem **Dokumente** in dem **Archivsystem** gesucht werden.

Attribute

Kriterium[1]

Querverweise

Archivsystem
Dokument

Systemdaten

Die **Systemdaten** sind Bestandteil der **Eintragsdaten** und speichern systemspezifische Daten wie Erfassungsdatum und den Benutzernamen des Benutzers, der den **Eintrag** erfaßt hat.

Attribute

Erfassungsdatum[1]
Benutzername[1]

Querverweise

Eintrag
Eintragsdaten

Userdaten

Die **Userdaten** sind Bestandteil der **Eintragsdaten** und speichern Daten, die anhängig vom **Eintragstyp** sind.

Attribute

Userdaten[1]

Querverweise

Eintragsdaten
Eintragstyp

Userdaten-Eingabemaske

Die **Userdaten-Eingabemaske** dient der Eingabe der **Userdaten.**

Attribute

Userdaten[1]

Querverweise

<Keine>

Verkäufer

Der **Verkäufer** ist eine **Rolle**, die ein **Benutzer** bei der Benutzung des Archivsystems einnehmen kann.

Attribute

<Keine>

Querverweise

Archivsystem
Benutzer
Rolle

Version

Wird ein **Blatt** einer **Zeichnung** geändert, wird von diesem **Blatt** eine neue **Version** erzeugt. Die Version wird durch die Versionsnummer eindeutig beschrieben. Die Beschreibung beschreibt die Version.

Attribute

Versionsnummer[1]
Beschreibung[1]

Querverweise

Blatt
Zeichnung

Volltextsuche

Bei der **Volltextsuche** wird ein Suchbegriff in dem **Dokumententext** gesucht und nicht in den **Eintragsdaten**.

Attribute

Dokumententext[1]

Querverweise

Dokument
Eintragsdaten

Zeichnung

Eine **Zeichnung** ist eine spezieller **Dokumententyp**. Eine Zeichnung kann anhand ihrer Zeichnungsnummer identifiziert werden. Die Zeichnung wird durch eine Beschreibung beschrieben. Eine **Zeichnung** kann durch den **Qualitätssicherungsmitarbeiter** die **Freigabe** erhalten.

Attribute

Zeichnungsnummer[1]
Beschreibung[1]
Freigabeentscheidung[1]

Querverweise

Dokumententyp

8.1.3 Beschreibung der Use-Cases

In den Szenarien in Kapitel 2 und in den Systemvisionen in Kapitel 4 wurden Tätigkeiten und Arbeitsabläufe der beteiligten Akteure beschrieben. Diese Abläufe wurden bisher nur in Kurzform und ohne besondere Struktur dargestellt. Eine detailliertere und strukturierte Form der Betrachtung dieser Abläufe sind die von Jacobson [JCJÖ92] beschriebenen Use-Cases. Ein Use-Case beschreibt den Ablauf einer Tätigkeit, die ein Akteur mit dem System ausführt. Jeder dieser Use-Cases beschreibt einen abgeschlossenen Ablauf, der von einem Akteur angestoßen wurde, und die Interaktion zwischen dem Akteur und dem System. Die Gesamtheit der Use-Cases spezifiziert die vollständige Funktionalität des Systems. Zur Identifizierung der Use-Cases werden die einzelnen Akteure der Reihe nach betrachtet und deren Tätigkeiten in Form von Use-Cases beschrieben.

Die Beschreibung der Use-Cases orientiert sich an der in [JCJÖ92] teilweise benutzten Schreibweise. Im Rahmen dieser Arbeit wird über diese Schreibweise hinaus eine für alle Use-Cases gleiche Struktur eingeführt. Als Einführung in den einzelnen Use-Case dient die Kurzbeschreibung, auf die der von Vor- und Nachbedingungen eingerahmte Ablauf folgt. Die Vorbedingungen beschreiben die Bedingungen, die erfüllt sein müssen, bevor der Use-Case ausgeführt werden kann. Die Nachbedingungen beschreiben die Bedingungen, die erfüllt sein müssen, nachdem der Use-Case ausgeführt worden ist. Sind Alternativabläufe beim Ablauf eines Use-Cases möglich, werden diese im Anschluß beschrieben (vgl. [PG96g]).

Die Gesamtheit der Use-Cases spezifiziert die vollständige Funktionalität des Archivsystems. Dieses Archivsystem läßt sich demnach in den beiden Szenarien „Büro“ und „Konstruktionsbüro“ einsetzen. Um dennoch nachvollziehen zu können, aus welchem Szenario die einzelnen Use-Cases ursprünglich stammen, und um die Übersichtlichkeit der Darstellung zu steigern, wird hier zur Gliederung der Use-Cases wieder die in Kapitel 8.1 „Anforderungsmodell„ bereits verwendete erweiterte Notation verwendet. Die Use-Cases werden zu den Gruppen zusammengefaßt, welche die Szenarien widerspiegeln. Die Gruppennamen entsprechen den Szenariennamen und erhalten zur besseren Erkennbarkeit das Gruppensymbol „♠" vorangestellt (z.B. „♠Büro").

Die folgende Liste gibt eine Übersicht über die Use-Cases. Die Use-Cases aus dem Szenario ♠Büro werden im Anschluß an diese Liste in Kapitel 8.1.4 beschrieben, die Use-Cases aus dem Szenario ♠Konstruktionsbüro werden in Kapitel 8.1.5 beschrieben. In Kapitel 8.1.6 schließlich werden die abstrakten Use-Cases beschrieben. Abstrakte Use-Cases beschreiben Teilabläufe, die von mehreren konkreten Use-Cases benutzt und nicht alleine ausgeführt werden .

Auflistung der Use-Cases

♠Büro

✌Büromitarbeiter

- ✞Erfassung eines Dokumentes
- ✞Userdaten ändern
- ✞Eintrag löschen
- ✞Sitzungsbeginn
- ✞Sitzungsende

✌Sachbearbeiter

- ✞Suche eines Dokumentes
- ✞Betrachten eines Dokumentes
- ✞Reproduktion eines Dokumentes
- ✞Sitzungsbeginn
- ✞Sitzungsende

✌Archivar

- ✞Festlegen von Rechten

♠Konstruktionsbüro

✌Projektleiter

- ✞Anlegen eines Nummernkreises
- ✞Sitzungsbeginn
- ✞Sitzungsende

✌Konstrukteur

- ✞Bearbeitung einer Zeichnung
- ✞Überarbeitung einer Zeichnung
- ✞Übernahme einer Zeichnung
- ✞Suchen einer Zeichnung
- ✞Sitzungsbeginn
- ✞Sitzungsende

✌Qualitätssicherungsmitarbeiter

- ✞Überprüfung einer Zeichnung
- ✞Verfassen einer Anmerkung
- ✞Weiterleitung einer Zeichnung

✞Freigabe einer Zeichnung
✞Sitzungsbeginn
✞Sitzungsende

✌**Archivar**

✞ Anlegen eines neuen Benutzers

♠*Abstrakte Use-Cases*

✞Eintrag auswählen
✞Userdaten erfassen
✞Suchen durch komplexe Abfrage
✞Suchen durch einfache Abfrage
✞Suchen durch Volltextsuche

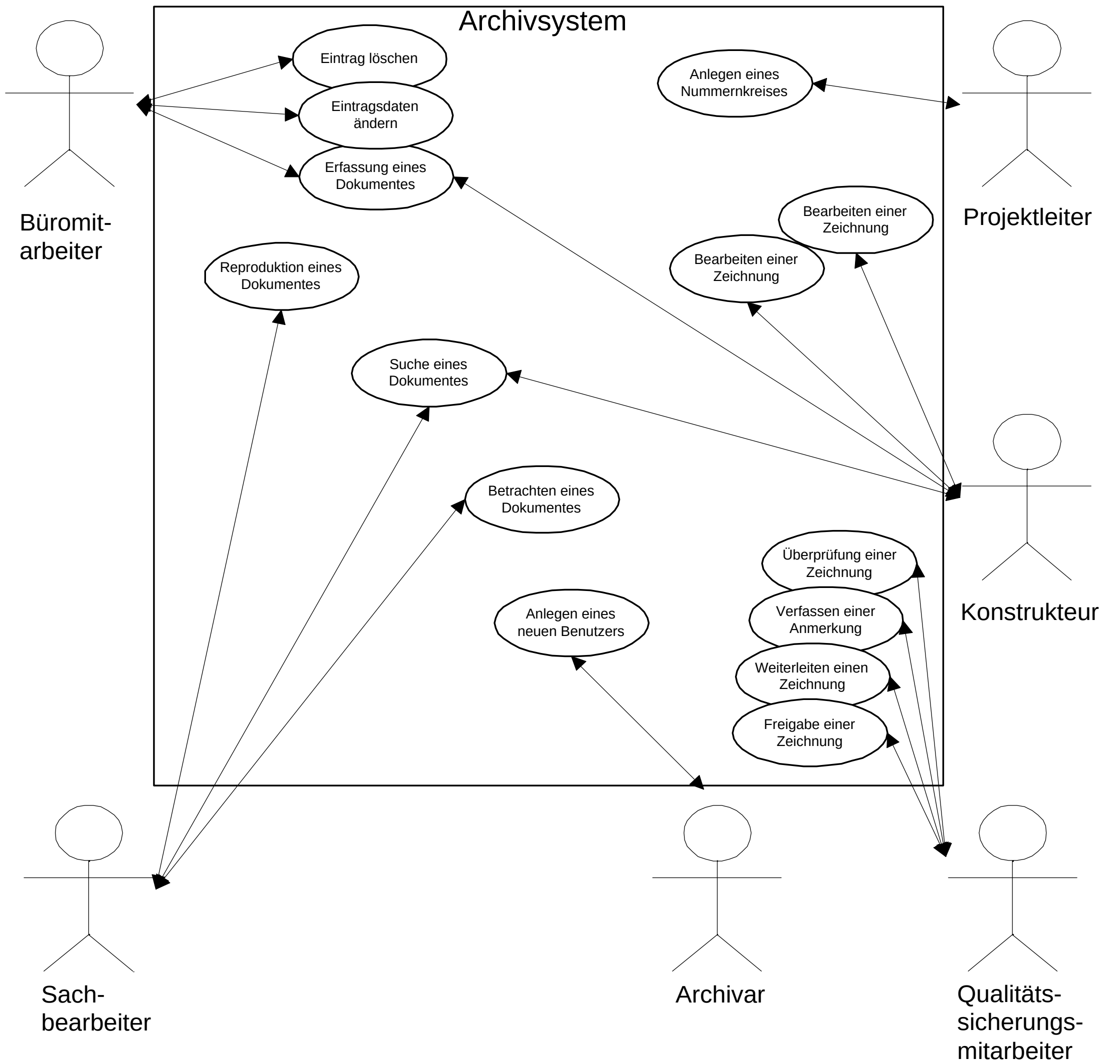

Abb. 11: Akteure und Use-Cases

8.1.4 Use-Cases „🌢Büro“

Beschreibung des Use-Cases „✞Erfassung eines Dokumentes“

Dieser Use-Case beschreibt die Erfassung eines in computergerechter Form vorliegenden **Dokumentes** mit den dazu gehörigen **Eintragsdaten** und die anschließende Übernahme in das **Archivsystem**.

Vorbedingung:

(1) Das **Dokument** liegt für die Erfassung in einer direkt computerverarbeitbaren Form vor. Ein nicht in einer direkt computerverarbeitbaren Form vorliegendes Dokument wurde mit geeigneten Mitteln in eine direkt computerverarbeitbare Form gebracht.
(2) Der Benutzer ist ein ✌**Büromitarbeiter**.

Ablauf:

(1) Dem Benutzer wird eine **Dateiauswahlliste** zur Auswahl von Dateien angeboten.
(2) Die **Datei**, die zu archivieren ist, wird ausgewählt.
(3) Das Archivsystem überprüft die Berechtigung zum lesenden Zugriff auf diese Datei. Existiert diese Berechtigung nicht, wird der **Benutzer** darauf hingewiesen, und er muß eine neue Auswahl treffen.
(4) Zur Überprüfung der Auswahl kann man die gewählte Datei mit einem passenden **Betrachter** ansehen. (siehe Use-Case „✞Betrachten eines Dokumentes“).
(5) Die Auswahl der Datei kann geändert werden. Bei Änderung der Auswahl wird der Betrachter, mit dem die vorher gewählte Datei angesehen wurde, geschlossen und ein neuer passender Betrachter gestartet.
(6) Dem Benutzer wird eine Auswahlliste zur Auswahl eines **Eintragstyps** (**Rechnung**, **Bestellung**, **Produktinformation**) angeboten.
(7) Entsprechend des Eintragstyps kann unter den bereits bestehenden **Einträgen** ein Eintrag ausgewählt werden (siehe Use-Case „✞Einträge auswählen“), dessen **Userdaten** als Vorbelegung in den neuen Eintrag eingetragen werden. Wird kein bereits bestehender Eintrag als Vorlage ausgewählt, bleiben die Eintragsfelder leer.
(8) Die **Userdaten** werden in der **Userdaten-Eingabemaske** eingegeben (siehe Use-Case „✞Userdaten erfassen“). Die Userdaten können eventuell aus der parallel angezeigten Datei abgelesen werden, wenn vorher der Betrachter gestartet wurde.
(9) Die **Systemdaten** werden vom Archivsystem festgestellt und gespeichert.
(10) Der **Eintrag** wird in das Archivsystem übernommen. Dazu werden die **Eintragsdaten** gespeichert und eine Kopie der ausgewählten Datei erzeugt, die von dem Archivsystem verwaltet wird.
(11) Nach Rückfrage wird die soeben archivierte Datei, die noch als Originaldatei vorliegt, gelöscht. Dazu überprüft das System die Berechtigung zum schreibenden Zugriff auf diese Datei. Existiert diese Berechtigung nicht, kann die Originaldatei nicht gelöscht werden und der Benutzer wird darauf hingewiesen.

Nachbedingung

(1) Der **Eintrag** wurde in das Archivsystem übernommen und steht allen anderen Benutzern damit zur Verfügung.

Beschreibung des Use-Cases „✞Betrachten eines Dokumentes“

Dieser Use-Case beschreibt das Betrachten eines Dokumentes, welches zu dem aktuell ausgewählten Eintrag gehört.

Vorbedingung:

(1) Der darzustellende **Eintrag** wurde ausgewählt (siehe Use-Case „✞Eintrag auswählen“).
(2) Der Benutzer ist ✌**Büromitarbeiter** oder ✌**Sachbearbeiter**.

Ablauf:

(1) Für das gewählte **Dokument** wird ein geeigneter **Betrachter** gestartet.
(2) Das Dokument kann auf einem geeigneten Ausgabegerät (z.B. Drucker, Videorecorder) ausgegeben werden.

Nachbedingung:

(1) Das betrachtete Dokument hat sich nicht verändert.

Beschreibung des Use-Cases „ ✞Userdaten ändern“

Dieser Use-Case beschreibt, wie die **Userdaten** eines **Eintrags** geändert werden.

Vorbedingung:

(1) Es ist nur ein **Eintrag** ausgewählt (siehe Use-Case „✞Eintrag auswählen“).
(2) Der Benutzer ist ✌**Büromitarbeiter**.

Ablauf:

(1) Der abstrakte Use-Case „✞Userdaten erfassen“ wird durchgeführt, wobei die aktuellen **Userdaten** als Vorbelegung für die **Userdaten-Eingabemaske** dienen.

Nachbedingung

(1) Die gültigen Änderungen werden in das **Archivsystem** übernommen.

Beschreibung des Use-Cases „ ✞Eintrag löschen“

Dieser Use-Case beschreibt, wie ein **Eintrag** gelöscht wird.

Vorbedingung:

(1) Es ist ein Eintrag ausgewählt (siehe Use-Case „ ✞Eintrag auswählen“).
(2) Der Benutzer ist ✌**Büromitarbeiter**.
(3) Der Eintrag darf gelöscht werden, das heißt, einer Löschung des Dokumentes aus dem Archivsystem stehen keine rechtlichen Probleme entgegen.

Ablauf:

(1) Es findet eine Rückfrage statt, ob der **Benutzer** sich sicher ist, daß der ausgewählte **Eintrag** gelöscht werden soll.
(2) Bestätigt der Benutzer das Löschen, wird der **Eintrag** gelöscht. Bestätigt der Benutzer das Löschen nicht, wird der **Eintrag** nicht gelöscht.

Nachbedingung:

(1) Hat der Benutzer das Löschen bestätigt, ist der ausgewählte **Eintrag** gelöscht worden. Hat der Benutzer das Löschen nicht bestätigt, ist der ausgewählte **Eintrag** nicht gelöscht worden.

Beschreibung des Use-Cases „ ✞Sitzungsbeginn“

Möchte man das **Archivsystem** nutzen, muß es zunächst gestartet werden. Dieser Use-Case beschreibt den Ablauf eines Sitzungsbeginns. Jeder **Benutzer** kann diesen Use-Case ausführen.

Vorbedingung:

(1) Das Archivsystem ist noch nicht gestartet.

Ablauf:

(1) Der **Benutzer** startet das **Archivsystem**.
(2) Es erscheint der **Paßwort-Dialog** (siehe Kapitel 8.1.7 Schnittstellenbeschreibung Paßwort-Dialog und Abb. 12: Paßwort-Dialog).
(3) Es erscheint ein Fenster, in dem der **Benutzer** seinen Benutzernamen und sein Paßwort eingibt.
(4) Das Archivsystem überprüft, ob der Benutzer bekannt ist und ob das Paßwort korrekt ist.
(5) Ist der Benutzername nicht bekannt, wird der Benutzer darauf hingewiesen und kann den Benutzernamen korrigieren.
(6) Ist das Paßwort nicht korrekt, wird der Benutzer darauf hingewiesen und kann das Paßwort korrigieren.
(7) Waren die Eingaben korrekt, erscheint das **Hauptfenster** des Archivsystems (siehe Kapitel 8.1.7 Schnittstellenbeschreibung Hauptfenster und Abb. 13: Hauptfenster).

Nachbedingung:

(1) Der Benutzer wurde korrekt identifiziert, sieht das **Hauptfenster** und kann weitere Programmfunktionen ausführen.

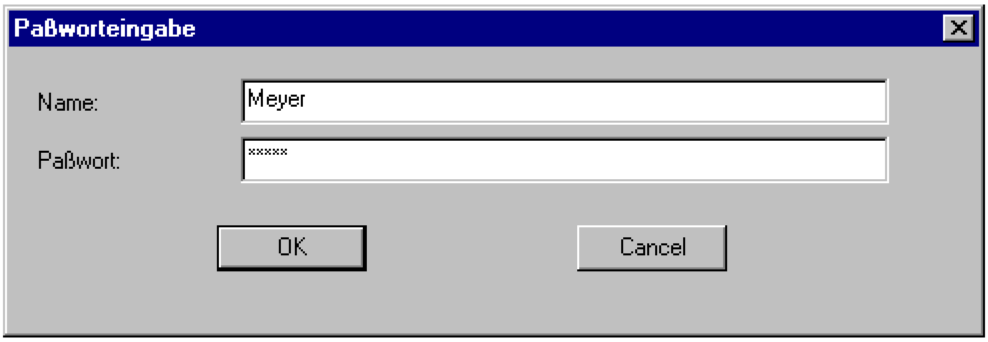

Abb. 12: Paßwort-Dialog

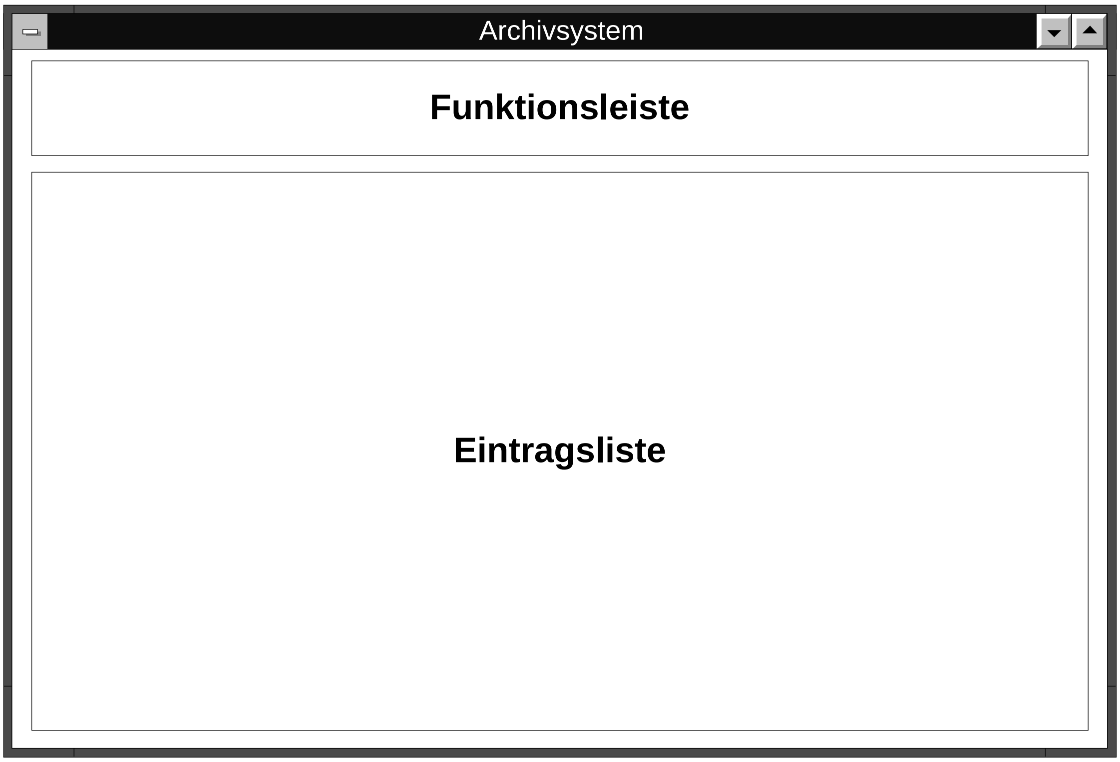

Abb. 13: Hauptfenster

Beschreibung des Use-Cases „✞Sitzungsende“

Hat der **Benutzer** die Arbeit mit dem **Archivsystem** beendet, wird dieser Use-Case ausgeführt. Jeder **Benutzer** kann diesen Use-Case ausführen.

Vorbedingung:

(1) Der **Benutzer** möchte das Programm beenden.
(2) Alle laufenden Aktionen müssen beendet sein.

Ablauf:

(1) Das **Hauptfenster** wird geschlossen.

Nachbedingung:

(1) Der **Benutzer** ist im System abgemeldet.

Beschreibung des Use-Cases „✞Suchen eines Dokumentes“

Dieser Use-Case beschreibt, wie mit Hilfe des **Archivsystems** ein bestimmtes **Dokument** gesucht wird. Jeder **Benutzer** kann diesen Use-Case ausführen.

Vorbedingung:

<Keine>

Ablauf:

(1) Es erscheint eine **Auswahlliste** mit einer Liste der drei Möglichkeiten zur Suche eines Eintrags. Diese Möglichkeiten sind „Komplexe Suche“, „Einfache Suche“ und „**Volltextsuche**“.
(2) Der **Benutzer** wählt zwischen „Komplexe Suche“ , „Einfache Suche“ und „**Volltextsuche**“.
(3) Hat der Benutzer die „Komplexe Suche“ gewählt, wird der abstrakte Use-Case „✞Suchen durch komplexe Abfrage“ ausgeführt.
(4) Hat der Benutzer die „Einfache Suche“ gewählt, wird der abstrakte Use-Case „✞Suchen durch einfache Abfrage“ ausgeführt.
(5) Hat der Benutzer die „**Volltextsuche**“ gewählt, wird der abstrakte Use-Case „✞Suchen durch **Volltextsuche**“ ausgeführt.

Nachbedingung:

(1) Die zu der Suchbeschreibung gefundenen Einträge werden angezeigt. Wurden zu der Suchbeschreibung keine Einträge gefunden, werden keine Einträge angezeigt.

Beschreibung des Use-Cases „ ✞Reproduktion eines Dokumentes“

Dieser Use-Case beschreibt die **Reproduktion** eines **Dokumentes**.

Vorbedingung:

(1) Der zu reproduzierende **Eintrag** wurde ausgewählt (siehe Use-Case „✞Eintrag auswählen“).
(2) Der **Benutzer** ist ✌**Büromitarbeiter** oder ✌**Sachbearbeiter**.

Ablauf:

(1) Das **Dokument** zu dem ausgewählten **Eintrag** wird auf einem geeigneten Ausgabegerät (z.B. Drucker, Videorecorder) ausgegeben.

Nachbedingung:

(1) Das **Dokument** hat sich nicht verändert.

Beschreibung des Use-Cases „ ✞Anlegen eines neuen Benutzers“

Dieser Use-Case beschreibt das Anlegen eines neuen Benutzers.

Vorbedingung:

(1) Der **Benutzer** ist ✌**Archivar**.

Ablauf:

(1) Es erscheint der **Neuer-Benutzer-Dialog** (siehe Kapitel 8.1.7. Schnittstellenbeschreibung Neuer-Benutzer-Dialog und Abb. 14: Neuer-Benutzer-Dialog).
(2) Das Eingabefeld „Benutzername“ wird mit dem Namen des **Benutzers**, der neu anzulegen ist, ausgefüllt.
(3) In dem Eingabefeld „Paßwort“ kann ein Paßwort voreingestellt werden, welches dem neuen Benutzer mitzuteilen ist.
(4) Aus einer Liste von Rollen kann schließlich eine ausgewählt werden, die dem neuen Benutzer zugeordnet wird.

Nachbedingung:

(1) Der neue Benutzer ist dem Archivsystem bekannt, und der neue Benutzer kann das System benutzen (siehe Use-Case „✞Sitzungsbeginn“).

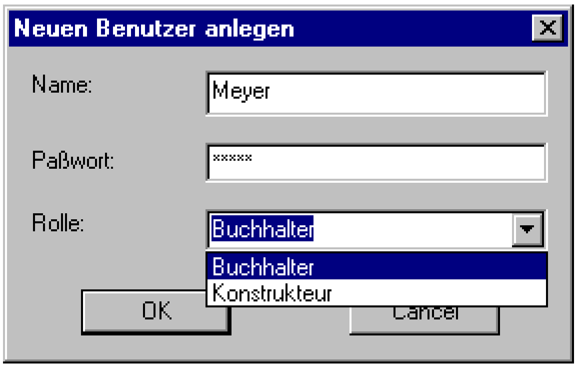

Abb. 14: Neuer-Benutzer-Dialog

8.1.5 Use-Cases „♠Konstruktionsbüro“

Beschreibung des Use-Cases „✞Anlegen eines Nummernkreises“

Der ✌**Projektleiter** legt zu Beginn eines Projektes einen **Nummernkreis** für die **Zeichnungen** an.

Vorbedingung:

(1) Ein ✌**Projektleiter** benötigt für ein Projekt einen neuen **Nummernkreis**.
(2) Der **Benutzer** ist ✌**Projektleiter**.

Ablauf:

(1) Der ✌**Projektleiter** gibt die Anzahl der benötigten **Zeichnungsnummern** und die Benennung des **Nummernkreises** ein.
(2) Das System reserviert die Zeichnungsnummern für die **Zeichnungen** und informiert den ✌**Projektleiter** über den Zahlenbereich der reservierten Zeichnungsnummern.

Nachbedingung:

(1) Es ist ein **Nummernkreis** reserviert.

Beschreibung des Use-Cases „✞Bearbeitung eines Dokumentes“

Dieser Use-Case beschreibt die Bearbeitung eines archivierten **Dokumentes** durch den ✌**Konstrukteur**.

Vorbedingung:

(1) Aus den Einträgen wurde genau ein **Eintrag** ausgewählt (siehe Use-Case „✞Eintrag auswählen“). Der Eintrag ist freigegeben.
(2) Der **Benutzer** ist ✌**Konstrukteur**.

Ablauf:

(1) Es wird eine Kopie des **Eintrags** erstellt.
(2) Die **Userdaten** des kopierten **Eintrags** müssen in der **Userdaten-Eingabemaske** geändert werden, um eine Unterscheidung von dem Originaleintrag zu ermöglichen.
(3) Es werden **Kopien** der Dateien, die das **Dokument** enthalten, angelegt. Das **Originaldokument** darf nicht verändert werden.

(4) Je nach Dokumententyp der gewählten Dokumente wird ein geeigneter Editor (Betrachter mit Änderungsmöglichkeiten des Dokuments) gestartet.
(5) Nach der Bearbeitung durch den Benutzer wird das geänderte **Dokument** wieder in das Archivsystem übernommen.

Nachbedingung:

(1) Der ursprüngliche Originaleintrag mit dem dazugehörigen Originaldokument ist unverändert.
(2) Der geänderte **Eintrag** mit dem bearbeiteten **Dokument** befindet sich im Archivsystem.

Beschreibung des Use-Cases „✞Überarbeitung eines Dokumentes“

Dieser Use-Case beschreibt die Überarbeitung eines noch nicht freigegeben **Dokumentes** durch den ✌**Konstrukteur**. Ist ein Dokument noch nicht freigeben worden, ist es nicht nötig, eine Kopie des Dokumentes zu erzeugen, um das Originaldokument vor Änderungen zu schützen.

Vorbedingung:

(1) Aus den Einträgen wurde genau ein **Eintrag** ausgewählt (siehe Use-Case „✞Eintrag auswählen“). Der Eintrag ist nicht freigegeben.
(2) Der Benutzer ist ✌**Konstrukteur**.

Ablauf:

(1) Es werden **Kopien** der Dateien, die das **Dokument** enthalten, angelegt. Die **Originaldokument** bleibt unverändert im Archivsystem und kann durch keinen weiteren Benutzer verändert werden.
(2) Je nach Dokumententyp der gewählten Dokumente wird ein geeigneter Editor gestartet.
(3) Nach der Bearbeitung durch den Benutzer wird das geänderte **Dokument** wieder in das Archivsystem übernommen und überschreibt das **Originaldokument**.

Nachbedingung:

(1) Der geänderte **Eintrag** mit dem bearbeiteten **Dokument** befindet sich im Archivsystem.

Beschreibung des Use-Cases „✞Erfassung eines Dokuments“

Dieser Use-Case wurde bereits im Szenario „♠**Büro**“ beschrieben. Dabei wurden die notwendigen Informationen, die zur Berücksichtigung der Anforderungen aus dem Szenario „♠**Konstruktionsbüro**“ nötig sind, noch nicht berücksichtigt. Da diese Fälle eine große Ähnlichkeit aufweisen und der Use-Case nur geringfügig zu ändern ist, werden hier nur die im ursprünglichen Use-Case zu ändernden Teile angegeben.

Änderungen gegenüber dem ursprünglichen Use-Case aus dem Szenario „♠Büro“:

Beschreibung:

Dieser Use-Case beschreibt die Erfassung eines in computergerechter Form vorliegenden **Dokumentes** mit den dazu gehörigen **Userdaten** in das **Archivsystem** durch einen ✌**Büromitarbeiter** oder einen ✌**Konstrukteur**.

Vorbedingung:

(1) Der Benutzer ist ✌**Büromitarbeiter** oder ✌**Konstrukteur**.

Ablauf:

(1) Dem Benutzer wird eine Auswahlliste zur Auswahl eines Eintragstyps (**Rechnung**, **Bestellung**, **Produktinformation, Zeichnung**) angeboten.

Beschreibung des Use-Cases „ ✞Überprüfung einer Zeichnung“

Dieser Use-Case beschreibt den Ablauf der Überprüfung einer **Zeichnung** durch den ✌**Qualitätssicherungsmitarbeiter**.

Vorbedingung:

(1) Der ✌**Qualitätssicherungsmitarbeiter** hat eine noch nicht freigegebene **Zeichnung** ausgewählt (siehe Use-Case „✞Eintrag auswählen“).
(2) Das System hat sichergestellt, daß der aktuelle Benutzer ein ✌**Qualitätssicherungsmitarbeiter** ist.

Ablauf:

(1) Der ✌**Qualitätssicherungsmitarbeiter** wählt über eine **Funktionsleiste** die Funktion „Betrachten“ aus.
(2) Die **Zeichnung** wird angezeigt (siehe Use-Case „✞Betrachten eines Dokumentes“).
(3) Entspricht die **Zeichnung** den Freigabekriterien, kann der ✌**Qualitätssicherungsmitarbeiter** die Freigabe starten (siehe Use-Case „✞Freigabe einer Zeichnung“).
(4) Entspricht die Zeichnung nicht den Freigabekriterien, kann der ✌**Qualitätssicherungsmitarbeiter** eine Anmerkung verfassen (siehe Use-Case „✞Verfassen einer Anmerkung“) und die Zeichnung dem ✌**Konstrukteur** zur Nachbearbeitung vorlegen (siehe Use-Case „✞Weiterleitung einer Zeichnung“).

Nachbedingung:

(1) Einer der beiden Use-Cases „✞Verfassen einer Anmerkung“ oder „✞Freigabe einer Zeichnung“ wird ausgeführt.

Beschreibung des Use-Cases „ ✞Freigabe einer Zeichnung“

Dieser Use-Case beschreibt den Ablauf der **Freigabe** einer **Zeichnung** durch den ✌**Qualitätssicherungsmitarbeiter**.

Vorbedingung:

(1) Der ✌**Qualitätssicherungsmitarbeiter** hat eine noch nicht freigegebene **Zeichnung** überprüft und stimmt der **Freigabe** zu, das heißt, der Use-Case „✞Überprüfung einer Zeichnung“ wurde ausgeführt.
(2) Die entsprechende Zeichnung ist ausgewählt (siehe Use-Case „✞Eintrag auswählen“).
(3) Der Benutzer ist ✌**Qualitätssicherungsmitarbeiter**.

Ablauf:

(1) Der ✌**Qualitätssicherungsmitarbeiter** wählt über die **Funktionsleiste** die Funktion „Freigabe“ aus.
(2) Nach Rückfrage wird die **Zeichnung** im Archivsystem als freigegeben markiert.

Nachbedingung:

(1) Die **Zeichnung** ist im Archivsystem als freigegeben markiert.

Beschreibung des Use-Cases „ ✞Verfassen einer Anmerkung“

Will der ✌**Qualitätssicherungsmitarbeiter** eine fehlerhafte **Zeichnung** dem ✌**Konstrukteur** wieder vorlegen, verfaßt er einen **Notizzettel** mit einer Fehlerbeschreibung.

Vorbedingung:

(1) Der ✌**Qualitätssicherungsmitarbeiter** hat den **Eintrag** ausgewählt, der die fehlerhafte **Zeichnung** enthält (siehe Use-Case „✞Eintrag auswählen“), das heißt, der Use-Case „✞Überprüfung einer Zeichnung“ wurde ausgeführt.

Ablauf:

(1) Der ✌**Qualitätssicherungsmitarbeiter** wählt über die **Funktionsleiste** die Funktion „**Notizzettel**“ aus.
(2) Es erscheint ein Fenster mit einem Bereich zur Texteingabe.
(3) Der ✌**Qualitätssicherungsmitarbeiter** verfaßt die Fehlerbeschreibung.
(4) Die Beendigung der Eingabe wird dem Archivsystem mitgeteilt und das Fenster wird geschlossen.

Nachbedingung:

(1) Der **Notizzettel** wird zusammen mit dem **Eintrag** vom Archivsystem verwaltet.

Beschreibung des Use-Cases „✞Weiterleitung einer Zeichnung“

Hat der ✌**Qualitätssicherungsmitarbeiter** bei der Überprüfung einer Zeichnung (siehe Use-Case „✞Überprüfung einer Zeichnung“) einen Mangel an einer **Zeichnung** festgestellt, kann er diese zur Bearbeitung (siehe Use-Case „✞Bearbeitung einer Zeichnung“) an den ✌**Konstrukteur** zurückleiten, damit dieser eine Fehlerbeseitigung vornehmen kann.

Vorbedingung:

(1) Der ✌**Qualitätssicherungsmitarbeiter** hat den Eintrag ausgewählt, der die fehlerhafte **Zeichnung** enthält (siehe Use-Case „✞Eintrag auswählen“).

Ablauf:

(1) Der ✌**Qualitätssicherungsmitarbeiter** wählt über die **Funktionsleiste** die Funktion „Weiterleitung“ aus.
(2) Es erscheint eine Bestätigung, daß die **Zeichnung** an den ✌**Konstrukteur** zurückgeleitet wird.

Nachbedingung:

(1) Die **Zeichnung** wird an den ✌**Konstrukteur** zurückgeleitet.

8.1.6 Use-Cases „💧Abstrakte Use-Cases“

Abstrakte Use-Cases im Sinne von OOSE beschreiben Teilabläufe, die von mehreren konkreten Use-Cases benutzt werden können. Der durch einen abstrakten Use-Case beschriebene Teilablauf wird in den Ablauf eines konkreten Use-Cases eingebunden. Die abstrakten Use-Cases sind nur in Verbindung mit den konkreten Use-Cases sinnvoll und werden nicht alleine ausgeführt.

Beschreibung des Use-Cases „✞Eintrag auswählen“

Dieser Use-Case beschreibt, wie in Eintrag zur weiteren Bearbeitung ausgewählt wird. Der Use-Case wird von den folgenden Use-Cases benutzt:

„✞Erfassung eines Dokumentes“
„✞Betrachtung eines Dokumentes“
„✞Userdaten ändern“
„✞Eintrag löschen“
„✞Reproduktion eines Dokumentes“
„✞Bearbeitung eines Dokumentes“
„✞Überprüfung einer Zeichnung“
„✞Freigabe einer Zeichnung“
„✞Verfassen einer Anmerkung“
„✞Weiterleitung einer Zeichnung“

Vorbedingung:

<Keine>

Ablauf:

(1) In dem **Fensterbereich** „**Eintragsliste**“ des **Hauptfensters** (siehe Abb. 13: Hauptfenster) wird eine die **Eintragsliste** mit den Einträgen angezeigt.
(2) Die **Eintragsdaten** werden als Informationszeile dargestellt. Der **Benutzer** kann einen Eintrag durch Auswahl mit einem Markierbalken markieren (siehe Kapitel 8.1.7 Schnittstellen-beschreibung Eintragsliste und Abb. 15: Eintrag auswählen).

Nachbedingung:

(1) Es gibt einen ausgewählten Eintrag.

```
-Nummernkreis 1000-1999, Projekt Kraftwerk
 -Eintrag 1: Zeichnung 1234 "Schraube"
  -Eintrag 2: Blatt 1 "Seitenansicht"
    -Eintrag 3: Version 1 "Erstzeichnung"
    -Eintrag 4: Version 2 "Jetzt mit Gewinde"
  -Eintrag 5: Blatt 2 "Sicht von oben"
    -Eintrag 6: Version 1 "Erstzeichnung"
    -Eintrag 7: Version 2 "Jetzt mit Schlitz"
 +Eintrag 8: Zeichnung 1235 "Schraubenzieher"
+Nummernkreis 2000-2999, Projekt Chemiefabrik
-Nummernkreis 3000-3999, Projekt Raffinerie
 -Eintrag 9: Zeichnung 3456 "Öltank"
  -Eintrag 10: Blatt 1 "Querschnitt"
   -Eintrag 11: Version 1 "Erstellung"
   -Eintrag 12: Version 2 "Neue Wandstärke"
 +Eintrag 13: Zeichnung 3457 "Pipeline"
```

Abb. 15: Eintrag auswählen

Beschreibung des Use-Cases „ ✞Userdaten erfassen“

Dieser Use-Case beschreibt, wie die **Userdaten** zu einem **Eintrag** erfaßt werden. Der Use-Case wird von den folgenden Use-Cases benutzt:

„✞Erfassung eines Dokumentes“
„✞Userdaten ändern“
„✞Bearbeitung eines Dokumentes“

Vorbedingung:

<Keine>

Ablauf:

(1) Die **Userdaten** werden in der **Userdaten-Eingabemaske** (siehe Kapitel 8.1.7 Schnittstellenbeschreibung der Userdaten-Eingabemaske und Abb. 16: Userdaten-Eingabemaske) dargestellt und der **Benutzer** kann die **Userdaten** in den Eingabefeldern eingeben.
(2) Die Eingaben werden auf Gültigkeit geprüft. Sind die Daten ungültig, wird der **Benutzer** darüber informiert und kann die Daten ändern.

Nachbedingung:

(1) Die **Userdaten** sind gültig und werden in das Archivsystem übernommen.

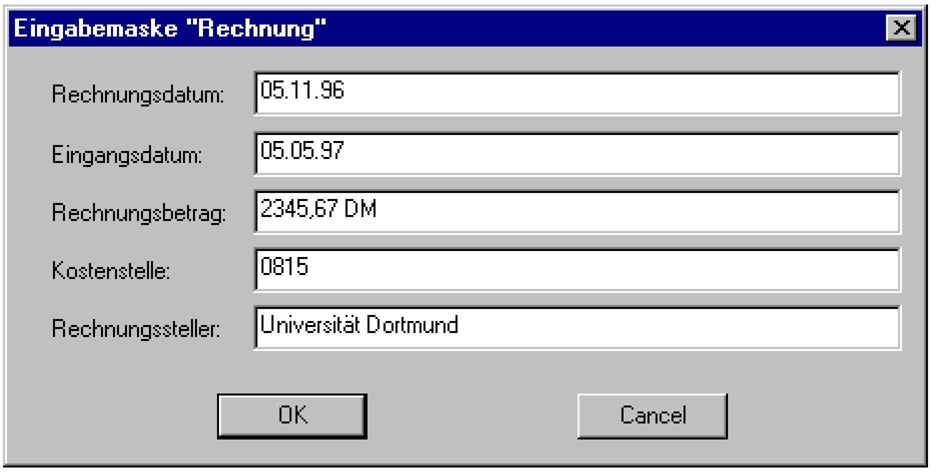

Abb. 16: Userdaten-Eingabemaske

Beschreibung des Use-Cases „✞Suchen durch komplexe Abfrage“

Dieser Use-Case beschreibt die Suche nach Einträgen durch eine komplexe Abfrage. Eine komplexe Abfrage besteht aus einem oder mehreren Suchausdrücken, die der **Benutzer** eingibt.

Vorbedingung:

<Keine>

Ablauf:

(1) Der **Benutzer** wählt den Eintragstyp des **Eintrags**, den der sucht.
(2) Über eine **Auswahlliste** kann der **Benutzer** ein Eintragsattribut zur Suche festlegen.
(3) Über eine **Auswahlliste** kann der Benutzer einen Suchoperator (Gleich, Ungleich, Größer, Kleiner, Enthält) auswählen.
(4) Der Benutzer gibt einen Vergleichswert ein.
(5) Die Suchausdrücke können durch „Und“, „Oder“, „Nicht“ verbunden werden und durch Klammerung in ihrer Auswertereihenfolge geändert werden.
(6) Der Benutzer kann die Abfrage erst starten, wenn mindestens ein Suchausdruck mit Attribut, Operator und Wert eingegeben worden ist.

Nachbedingung:

(1) Die zu der Suchbeschreibung gefundenen Einträge werden angezeigt. Wurden zu der Suchbeschreibung keine Einträge gefunden, werden keine Einträge angezeigt.

Beschreibung des Use-Cases „✞Suchen durch einfache Abfrage“

Dieser Use-Case beschreibt die Suche nach Einträgen durch eine einfache Abfrage. Im Unterschied zu der komplexen Abfrage kann bei der einfachen Abfrage nur über den Suchoperator „Gleich“ nach Einträgen gesucht werden. Durch den Wegfall der komplizierten Suchausdrücke kann der **Benutzer** so schneller die Eingabe beim Suchen erledigen.

Vorbedingung:

<Keine>

Ablauf:

(1) Der **Benutzer** wählt den Eintragstyp des **Eintrags**, den der sucht.
(2) Der Benutzer erstellt einen Beispieleintrag mit Beispieleintragsdaten (siehe Use-Case „✞Userdaten erfassen“).
(3) Alle Einträge werden gesucht, in denen die **Eintragsdaten** mit den Beispieleintragsdaten übereinstimmen. Leer gelassene Felder in dem Beispieleintrag werden dabei nicht berücksichtigt.

Nachbedingung:

(1) Die zu der Suchbeschreibung gefundenen Einträge werden angezeigt. Wurden zu der Suchbeschreibung keine Einträge gefunden, werden keine Einträge angezeigt.

Beschreibung des Use-Cases „✞Suchen durch Volltextsuche“

Dieser Use-Case beschreibt die Suche nach Einträgen durch **Volltextsuche**. Im Unterschied zu der komplexen und einfachen Abfrage wird ein Eintrag nicht anhand der **Eintragsdaten** gesucht, sondern anhand des **Dokumententextes**. So können auch Einträge gesucht werden, wenn die **Eintragsdaten** für eine Suche nicht umfangreich genug sind.

Vorbedingung:

<Keine>

Ablauf:

(1) Der **Benutzer** wählt den Eintragstyp des **Eintrags**, den der sucht.
(2) Es wird eine Eingabemaske zur Eingabe des Suchbegriffs angezeigt.
(3) Der Benutzer gibt den Suchbegriff ein, der gesucht werden soll.

Nachbedingung:

(1) Die zu dem Suchbegriff gefundenen Einträge werden angezeigt. Wurden zu dem Suchbegriff keine Einträge gefunden, werden keine Einträge angezeigt.

8.1.7 Schnittstellenbeschreibungen

Bei der Erstellung des Use-Case-Modells können außer der textuellen Beschreibung auch noch Schnittstellen, zum Beispiel Bildschirmmasken, verwendet werden. Die Schnittstellenbeschreibungen sind damit Teil des Use-Case-Modells. Die Schnittstellenbeschreibungen verdeutlichen einem möglichen Endbenutzer die Benutzung des Systems auf einfache Art und Weise.

Die Schnittstellen wurden bereits bei der Beschreibung der Use-Cases verwendet. Hier folgt der nun die Übersicht aller Schnittstellen mit der Schnittstellenbeschreibung.

Betrachter

Der Betrachter dient dazu, sich am Bildschirm ein **Dokument** zu betrachten. Der zu betrachtende Teil des **Dokumentes** kann näher bestimmt werden. Je nach Art des Dokumentes gibt es dazu verschiedene sinnvolle Möglichkeiten, die vom Benutzer ausgewählt werden.

Bei Textdokumenten kann man:

- eine Seite vorblättern,
- eine Seite zurückblättern,
- zur ersten Seite zurückspringen.

Bei graphischen Dokumenten kann man:

- die Ansicht vergrößern,
- die Ansicht verkleinern,
- einen Ausschnitt bestimmen, der dann vergrößert dargestellt wird,
- die Ansicht auf Vollansicht anpassen,
- die maximale Vergrößerung einstellen.

Bei Audio- und Videodokumenten kann man:

- die Wiedergabe starten,
- die Wiedergabe anhalten,
- die Wiedergabe fortsetzen,
- Vor- oder Zurückspulen.

Hauptfenster

Das Hauptfenster (siehe Abb. 13: Hauptfenster) ist die wichtigste Schnittstelle des Systems zum Benutzer.

Das **Hauptfenster** enthält einen **Fensterbereich** mit der **Funktionsleiste**, mit deren Hilfe der Benutzer Programmfunktionen auslösen kann und einen Fensterbereich mit der **Eintragsliste**, welche die **Einträge** darstellt (siehe Abb. 15: Eintrag auswählen).

Eintragsliste

Die Eintragsliste (siehe Abb. 15: Eintrag auswählen) stellt eine Liste von Einträgen dar und ist Bestandteil des Hauptfensters. Die **Eintragsdaten** eines Eintrags werden als Informationszeile dargestellt. Der **Benutzer** kann einen Eintrag durch Anwahl mit einem Markierbalken markieren (siehe Abb. 15: Eintrag auswählen).

Einem Eintrag können noch weitere Einträge zugeordnet sein. Zeichnungen werden ihrem Nummernkreise zugeordnet, Blätter ihrer Zeichnung und die Versionen ihren Blättern. Dadurch ergibt sich eine hierarchische Struktur der Einträge, die sich in der Darstellung der Eintragsliste widerspiegelt.

Bei der Auswahl eines Eintrags aus der Eintragsliste gilt folgendes:

- Ein Markierbalken kann auf jeden Eintrag der Eintragsliste positioniert werden.
- Durch eine geeignete Benutzerangabe wird der Eintrag, der sich unterhalb des Markierbalkens befindet, ausgewählt.
- Zu jedem Zeitpunkt ist entweder kein oder ein Eintrag ausgewählt.
- Der ausgewählte Eintrag wird vom System invers hinterlegt, um die Auswahl zu verdeutlichen (vgl. Abb. 15: Eintrag auswählen, Eintrag 6).
- Ein bereits ausgewählter Eintrag wird durch erneute Auswahl demarkiert.

- Gehören zu einem Eintrag weitere Einträge, wird dieses durch ein kleines Plus-Symbol an dem Eintrag gekennzeichnet. Man kann durch Anwahl des Plus-Symbols die weiteren Einträge anzeigen lassen (vgl. Abb. 15: Eintrag auswählen, Eintrag 8), dadurch ändert sich das Plus- in ein Minus-Symbol.
- Sind die zu einem Eintrag gehörenden weiteren Einträge ausgeklappt, wird dieses durch ein kleines Minus-Symbol an dem Eintrag gekennzeichnet. Man kann durch Anwahl des Minus-Symbols die weiteren Einträge wieder einklappen lassen (vgl. Abb. 15: Eintrag auswählen, Eintrag 1), dadurch ändert sich das Minus- in ein Plus-Symbol.

Funktionsleiste

Über die **Funktionsleiste** kann der **Benutzer** die Programmfunktionen auslösen.

Neuer-Benutzer-Dialog

Mit Hilfe des **Neuer-Benutzer-Dialogs** (siehe Abb. 14: Neuer-Benutzer-Dialog) wird ein neuer Benutzer im System angelegt. Einem neuen Benutzer wird ein im Archivsystem eindeutiger Name gegeben, der die Person identifiziert. Es wird außerdem ein Paßwort vergeben, das der Benutzer vor Benutzung des Archivsystems korrekt eingeben muß. Dieses Paßwort wird vom Archivar auf einem sicheren Weg dem Benutzer mitgeteilt, um Mißbrauch auszuschließen. Schlie0lich werden dem Benutzer noch Rollen zugeordnet, die er in dem Archivsystem annimmt.

Userdaten-Eingabemaske

Mit Hilfe der **Userdaten-Eingabemaske** (siehe Abb. 16: Userdaten-Eingabemaske) können Userdaten eingegeben werden.

Paßwort-Dialog

Der **Paßwort-Dialog** (siehe Abb. 12: Paßwort-Dialog) dient dazu, bei Sitzungsbeginn den Benutzer anhand seines Namens zu identifizieren und die Berechtigung zur Benutzung des Archivsystems anhand seines Paßwortes zu verifizieren.

8.2 Analysemodell

In dem Anforderungsmodell wurde spezifiziert, welche Funktionalität das System haben soll. Mit Hilfe des Analysemodells soll jetzt eine Ausgangsbasis für die Systementwicklung entstehen.

Das Analysemodell dient dazu, eine logische und wartbare Struktur in das System bringen. Logische Struktur bedeutet hier, daß die tatsächliche Implementationsumgebung nicht berücksichtigt wird und nur die Systemfunktionalität betrachtet wird. Zur Strukturierung des Systems werden drei Objekttypen benutzt:

- Schnittstellenobjekte,
- Entity-Objekte und
- Kontrollobjekte.

Die Schnittstellenobjekte modellieren die Funktionalität der Systemschnittstellen und beschreiben die Objekte, durch die ein Akteur mit dem System kommuniziert. Die Entity-Objekte modellieren die Funktionalität der Behandlung der Information, die längere Zeit im System vorhanden ist. Vereinfacht gesagt sind Entity-Objekte Datenobjekte. Kontrollobjekte decken die Funktionalität ab, die nicht durch die beiden ersten Objekttypen abgedeckt ist, also beispielsweise Verhaltensweisen von Objekten. Kontrollobjekte kontrollieren damit andere Objekte.

Die Schnittstellenobjekte, Entity-Objekte und Kontrollobjekte werden bei der Analyse eines Use-Cases identifiziert und sollen die komplette Funktionalität des Use-Cases unterstützen.

8.2.1 Schnittstellenobjekte

Mit Hilfe der Schnittstellenobjekte kommunizieren die Akteure mit dem System. Ein Schnittstellenobjekt übersetzt zum einen die Eingaben eines Akteurs in Systemereignisse, zum anderen werden Systemereignisse, die einem Akteur präsentiert werden sollen, in eine dafür geeignete Ausgabeform gebracht.

Die Schnittstellenobjekte werden in diesem Abschnitt aufgezählt. Dazu wird zuerst der Name des Schnittstellenobjekts genannt, und dann eine kurze Beschreibung der Funktionalität gegeben. In den graphischen Übersichten werden Schnittstellenobjekte durch dieses Symbol gekennzeichnet:

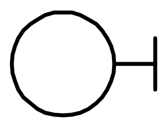

Abb. 17: Symbol für ein Schnittstellenobjekt

Die Use-Cases werden nun in der durch die in Kapitel 8 gegebene Reihenfolge betrachtet und die in den einzelnen Use-Cases enthaltenen Schnittstellenobjekte sukzessiv beschrieben. Diese Vorgehensweise entspricht der Vorgehensweise, die Jacobson [JCJÖ92, S. 176] zur Findung der Schnittstellenobjekte vorschlägt.

Die bei der Betrachtung aller Use-Cases aus Kapitel 8.1.3 gefundenen Schnittstellenobjekte sind in alphabetischer Reihenfolge:

Auswahlliste

In einer Auswahlliste kann aus einer Auswahl von Optionen eine Option ausgewählt werden.

Betrachter

Der Betrachter dient dazu, sich am Bildschirm ein Dokument zu betrachten.

Dateiauswahlliste

Die Dateiauswahlliste stellt eine Liste dar, aus der eine Datei ausgewählt werden kann.

Eintragsliste

Die Eintragsliste stellt eine Liste von Einträgen des Archivsystems dar.

Funktionsleiste

Über die Funktionsleiste kann der Benutzer Funktionen auslösen.

Hauptfenster

Das Hauptfenster besteht aus der Eintragsliste und der Funktionsleiste

Neuer-Benutzer-Dialog

In dem Neuer-Benutzer-Dialog werden die Daten zu einem neuen Benutzer eingetragen.

Paßwort-Dialog

In diesem Dialogfenster gibt der Benutzer seinen Namen und sein Paßwort ein.

Userdaten-Eingabemaske

Mit Hilfe der Userdaten-Eingabemaske können Userdaten eingegeben werden.

Die Beziehungen zwischen den einzelnen Schnittstellenobjekten veranschaulicht Abb. 18: Beziehungen zwischen den Schnittstellenobjekten.

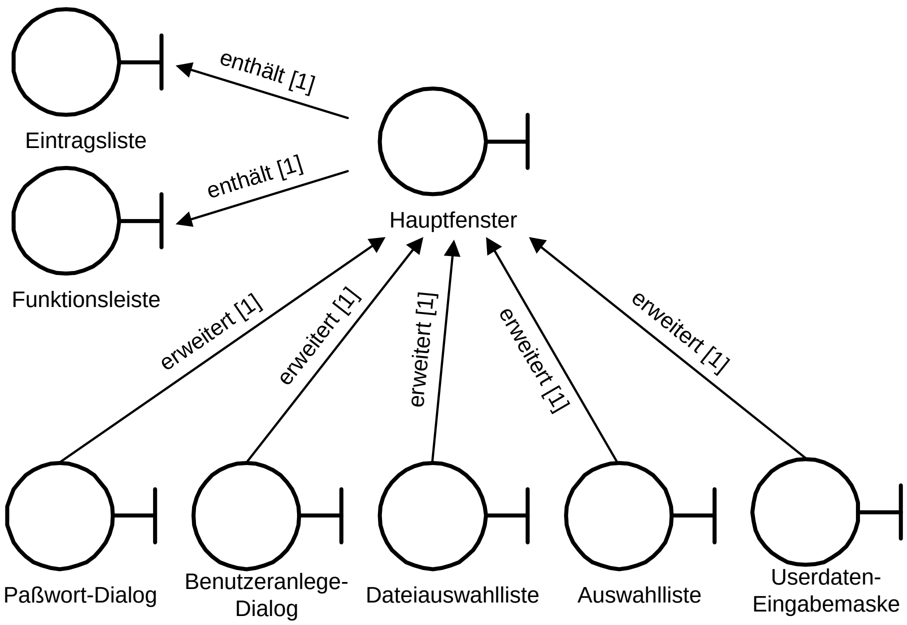

Abb. 18: Beziehungen zwischen den Schnittstellenobjekten

8.2.2 Entity-Objekte

Entity-Objekte modellieren die Information, die das System über einen längeren Zeitraum verwaltet. Um die Information zu speichern, benutzen Objekte Attribute. Zu einem Entity-Objekt gehören also ein oder mehrere Attribute. Jedes Attribut hat einen Typ. Dieser Typ kann entweder ein primitiver Datentyp wie eine Zahl oder eine Zeichenkette oder ein zusammengesetzter Datentyp sein.

Struktur der Beschreibung

Ein Attribut eines Entity-Objekts wird mit Hilfe einer Assoziation dargestellt. Diese Assoziation hat einen Namen und eine Kardinalität und verweist auf den Typ des Attributes. Die Kardinalität gibt an, welche Anzahl des durch die Assoziation beschriebenen Attributes für das Entity-Objekt möglich ist. In dem Beispiel (Abb. 19: Objekt mit Attribut) hat das Entity-Objekt „Element“ genau ein Attribut mit Namen „Attribut“, welches den Typ „Typ“ besitzt. Die Kardinalität wird auch dann angegeben, wenn nur ein Attribut möglich ist, um Fehlinterpretationen zu vermeiden. Diese Notation wurde von [JCJÖ92] übernommen.

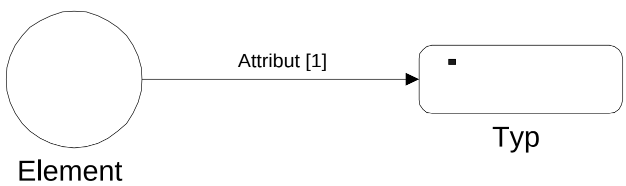

Abb. 19: Objekt mit Attribut

Die bei der Beschreibung der Entity-Objekte verwendeten primitiven Datentypen sind:

Zahl

Eine beliebige Fließkommazahl, z.B. 3.141592 oder 4711.0

Ganzzahl

Eine beliebige ganze Zahl, z.B. 4711 oder 0

Zeichenkette

Eine Folge von alphanumerischen Zeichen, z.B. „Mayer“ oder „DOKID12345“

Boolean

Eine Wahrheitsaussage. Der Wert kann entweder Ja/ Richtig/ Wahr oder Nein/ Falsch/ Unwahr annehmen.

Datum

Eine Datumsangabe, z.B. „5.5.97“

Beschreibung der Entity-Objekte

Die Use-Cases werden nun in der durch die in Kapitel 8 gegebene Reihenfolge betrachtet und die in den einzelnen Use-Cases enthaltenen Entity-Objekte sukzessiv beschrieben.

♠Büro

✐Büromitarbeiter

Beteiligte Objekte im Use-Case „✞Erfassung von Dokumenten“

Eintrag	(Abb. 20: Attribute von „Eintrag“)
Eintragsdaten	(Abb. 21: Attribute von „Eintragsdaten“)
Systemdaten	(Abb. 23: Attribute von „Systemdaten)
Userdaten	(Abb. 27: Vererbungsbeziehungen von „Userdaten“)
Userdaten-Eingabemaske	(Abb. 16: Userdaten-Eingabemaske)
Dokument	(Abb. 24: Attribute von „Dokument“)
Datei	(Abb. 22: Attribute von „Datei)
Archiv	(Abb. 26: Attribute von „Archiv“)
Archivsystem	(Abb. 25: Attribute von „Archivsystem“)
Rechnungsdaten	(Abb. 28: Attribute von „Rechnungsdaten“)
Bestellungsdaten	(Abb. 29: Attribute von „Bestellungsdaten“)
Produktinformationsdaten	(Abb. 30: Attribute von „Produktinformationsdaten“)

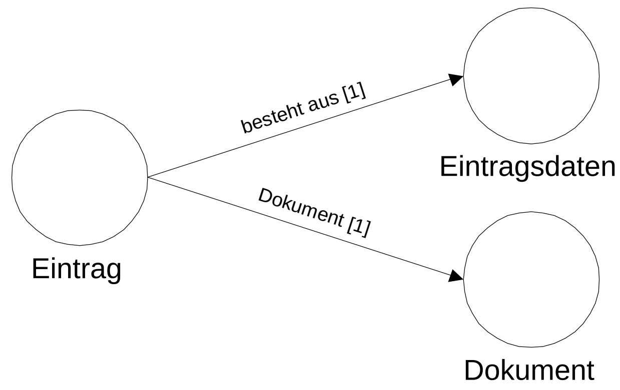

Abb. 20: Attribute von „Eintrag“

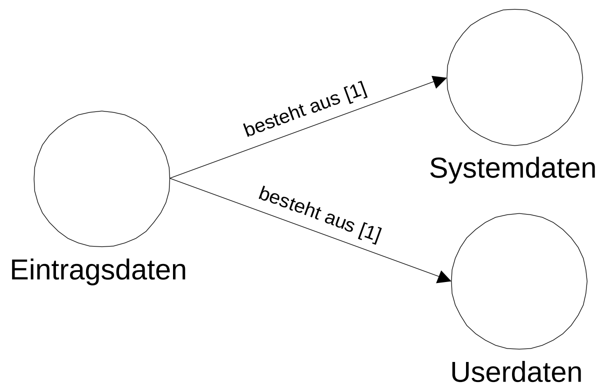

Abb. 21: Attribute von „Eintragsdaten“

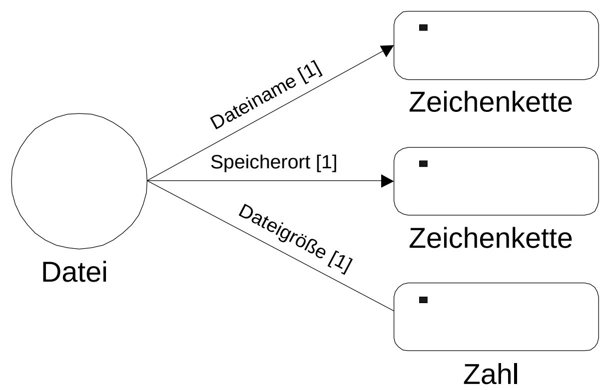

Abb. 22: Attribute von „Datei“

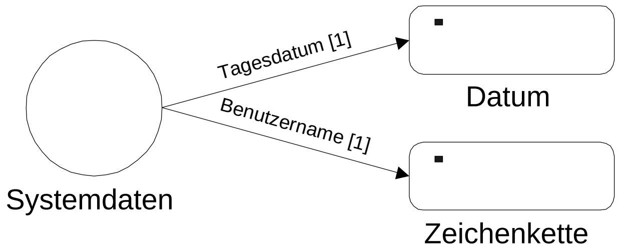

Abb. 23: Attribute von „Systemdaten“

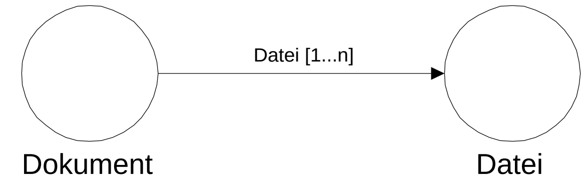

Abb. 24: Attribute von „Dokument“

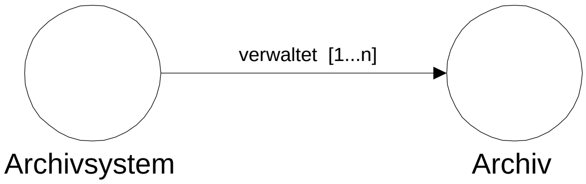

Abb. 25: Attribute von „Archivsystem“

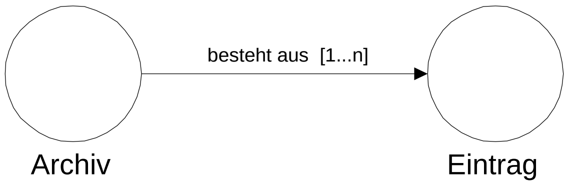

Abb. 26: Attribute von „Archiv“

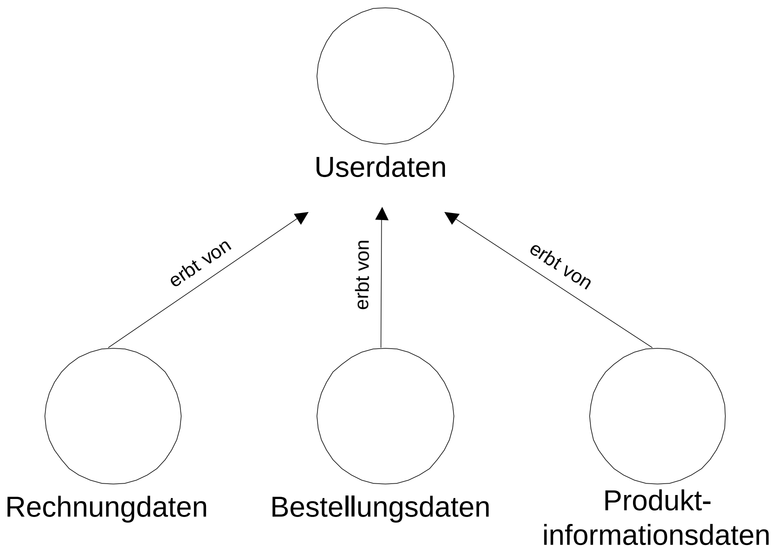

Abb. 27: Vererbungsbeziehungen von „Userdaten“

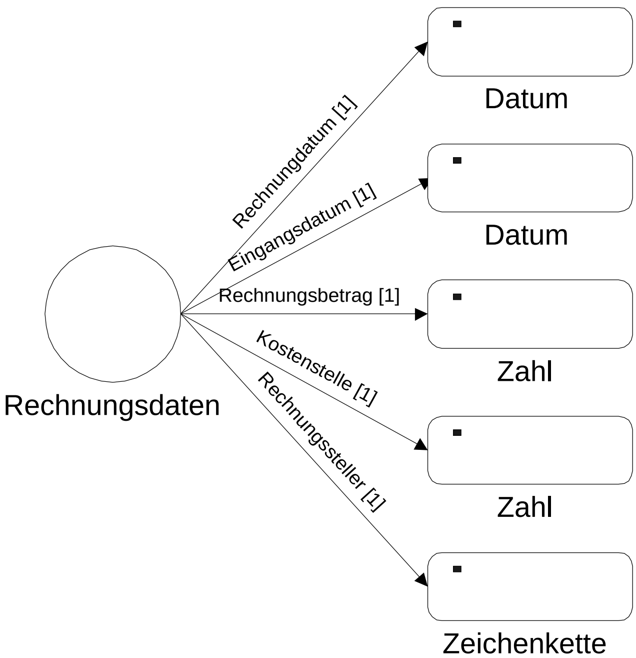

Abb. 28: Attribute von „Rechnungsdaten“

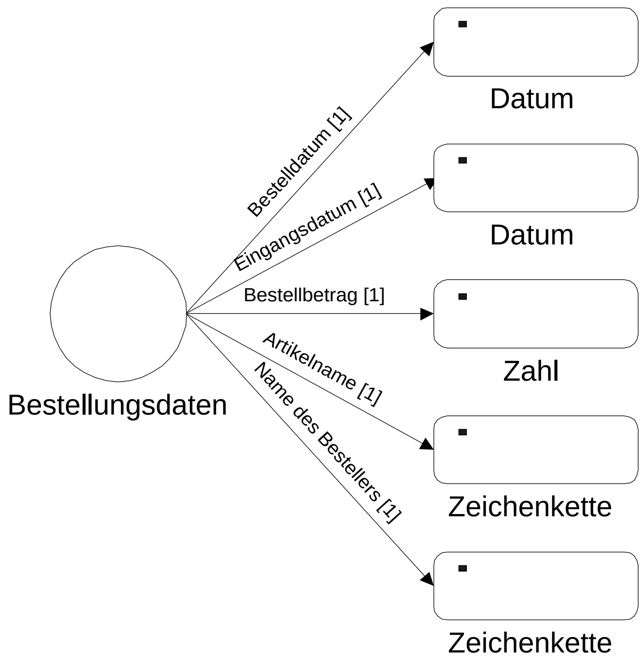

Abb. 29: Attribute von „Bestellungsdaten“

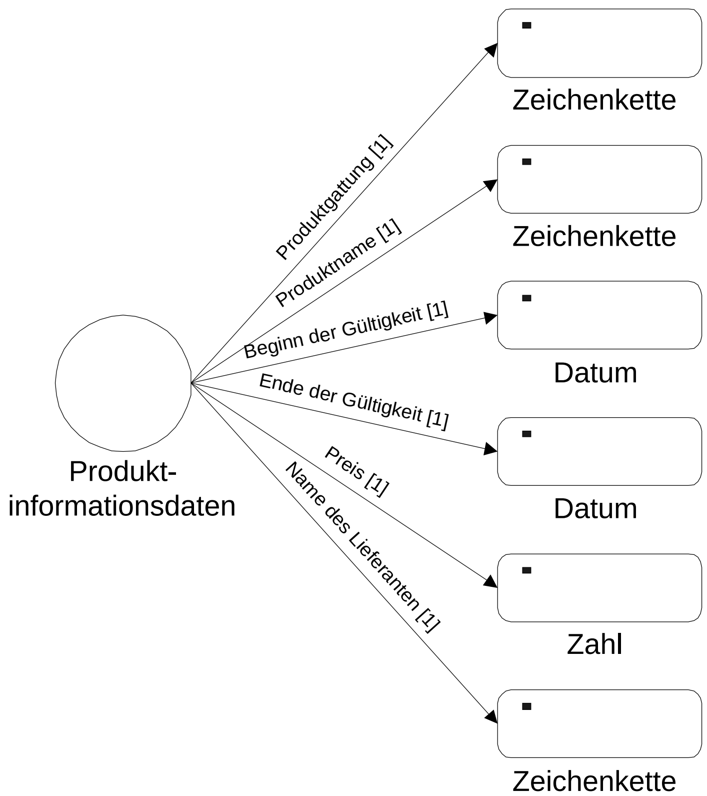

Abb. 30: Attribute von „Produktinformationsdaten“

Beteiligte Objekte im Use-Case „✞Betrachtung eines Dokumentes"

Betrachter	
Dokument	(Abb. 24: Attribute von „Dokument")
Eintrag	(Abb. 20: Attribute von „Eintrag")

Beteiligte Objekte im Use-Case „✞Userdaten ändern"

Eintrag	(Abb. 20: Attribute von „Eintrag")
Userdaten	(Abb. 27: Vererbungsbeziehungen von „Userdaten")
Userdaten-Eingabemaske	(Abb. 16: Userdaten-Eingabemaske)

Beteiligte Objekte im Use-Case „✞Eintrag löschen"

Benutzer	(Abb. 31: Attribute von „Benutzer")
Eintrag	(Abb. 20: Attribute von „Eintrag")

Beteiligte Objekte im Use-Cases „✞Sitzungsbeginn"

Benutzer	(Abb. 31: Attribute von „Benutzer")
Eintragsliste	(Abb. 32: Attribute von „Eintragsliste")
Paßwort-Dialog	(Abb. 12: Paßwort-Dialog)
Hauptfenster	(Abb. 13: Hauptfenster)

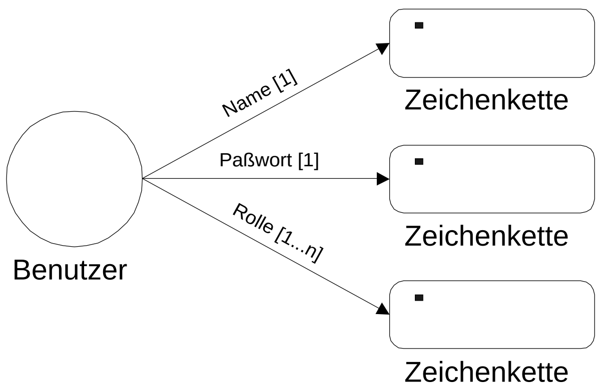

Abb. 31: Attribute von „Benutzer"

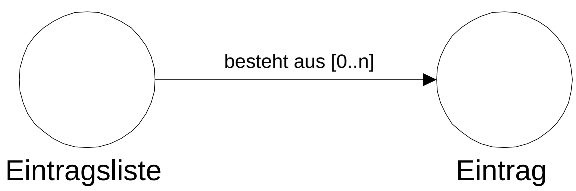

Abb. 32: Attribute von „Eintragsliste"

Beteiligte Objekte im Use-Case „✞Sitzungsende“

Benutzer	(Abb. 31: Attribute von „Benutzer“)
Hauptfenster	(Abb. 13: Hauptfenster)

✌Sachbearbeiter

Beteiligte Objekte im Use-Case „✞Suchen eines Dokumentes“

Benutzer	(Abb. 31: Attribute von „Benutzer“)
Dokument	(Abb. 24: Attribute von „Dokument“)

Beteiligte Objekte im Use-Case „✞Reproduktion eines Dokumentes“

Dokument	(Abb. 24: Attribute von „Dokument“)
Eintrag	(Abb. 20: Attribute von „Eintrag“)

✌Archivar

Beteiligte Objekte im Use-Case „✞Anlegen eines neuen Benutzers“

Benutzer	(Abb. 31: Attribute von „Benutzer“)
Neuer-Benutzer-Dialog	(Abb. 14: Neuer-Benutzer-Dialog)

♠Konstruktionsbüro

✌Projektleiter

Beteiligte Objekte im Use-Case „✞Anlegen eines Nummernkreises“

Nummernkreis	(Abb. 33: Attribute von „Nummernkreis“)
Zeichnung	(Abb. 34: Attribute von „Zeichnungdaten“)

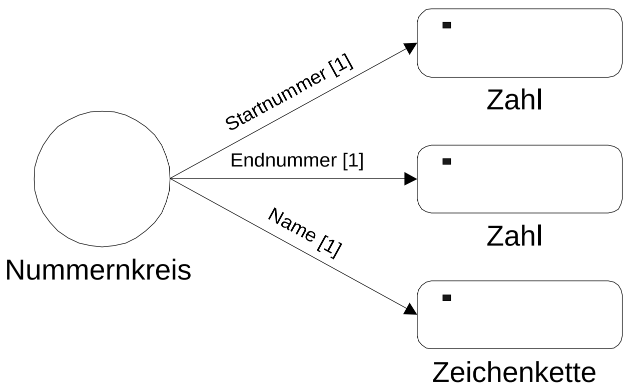

Abb. 33: Attribute von „Nummernkreis“

Beteiligte Objekte im Use-Case „✞Bearbeitung eines Dokumentes“

Dokument	(Abb. 24: Attribute von „Dokument“)
Eintrag	(Abb. 20: Attribute von „Eintrag“)
Eintragsdaten	(Abb. 21: Attribute von „Eintragsdaten“)
Userdaten	(Abb. 27: Vererbungsbeziehungen von „Userdaten“)
Userdaten-Eingabemaske	(Abb. 16: Userdaten-Eingabemaske)

Beteiligte Objekte im Use-Case „✞Überarbeitung eines Dokumentes“

Dokument	(Abb. 24: Attribute von „Dokument“)
Eintrag	(Abb. 20: Attribute von „Eintrag“)

Beteiligte Objekte im Use-Case „✞Erfassung eines Dokumentes“

Zeichnungsdaten	(Abb. 34: Attribute von „Zeichnungdaten“)
Blatt	(Abb. 35: Attribute von „Blatt“)
Version	(Abb. 36: Attribute von „Version“)

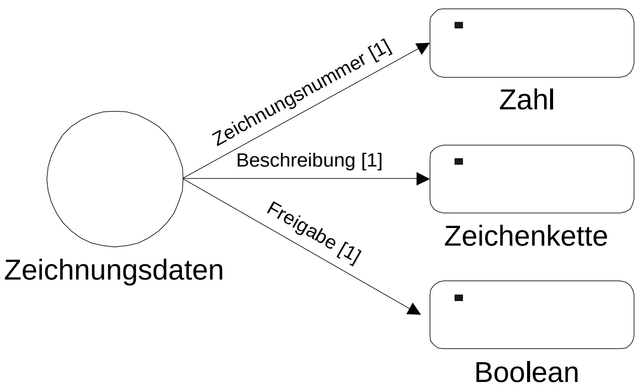

Abb. 34: Attribute von „Zeichnungdaten“

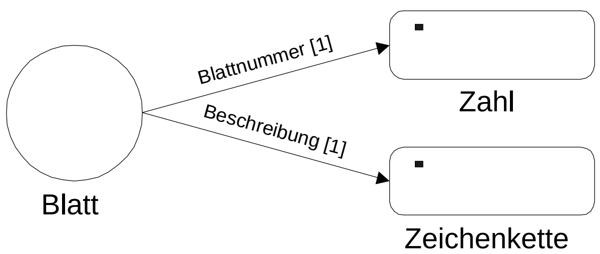

Abb. 35: Attribute von „Blatt“

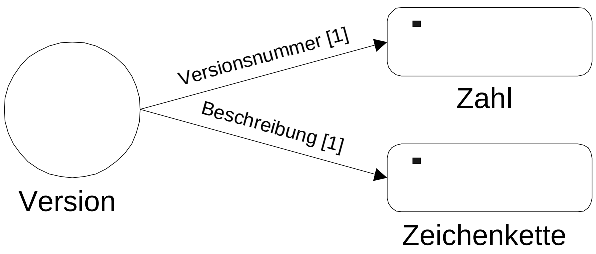

Abb. 36: Attribute von „Version“

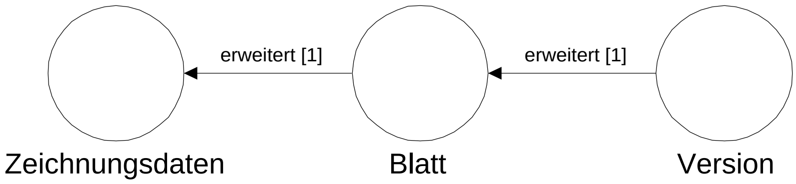

Abb. 37: Erweiterung von „Zeichnungsdaten“

Durch das in diesem Use-Case neu gefundene Entity-Objekt „**Zeichnungsdaten**“ werden die **Userdaten** (Abb. 27: Vererbungsbeziehungen von „Userdaten“), wie in Abb. 38: Erweiterte Vererbungsbeziehungen von „Userdaten“ gezeigt, erweitert.

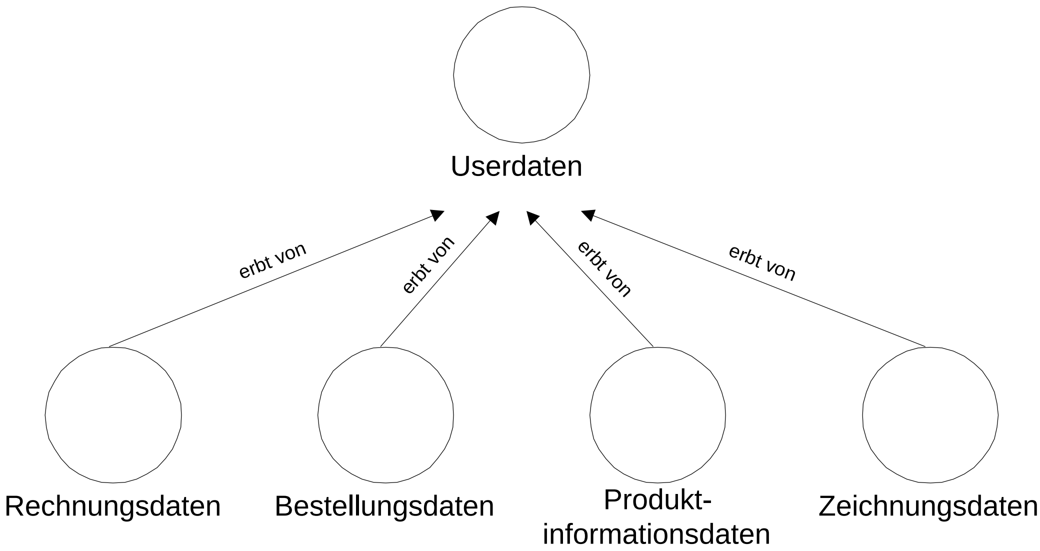

Abb. 38: Erweiterte Vererbungsbeziehungen von „Userdaten“

Beteiligte Objekte im Use-Case „✞Überprüfung einer Zeichnung“

Zeichnung (Abb. 34: Attribute von „Zeichnungdaten“)

Beteiligte Objekte im Use-Case „✞Freigabe einer Zeichnung“

Zeichnung (Abb. 34: Attribute von „Zeichnungdaten“)

Beteiligte Objekte im Use-Case „✞Verfassen einer Anmerkung“

Notizzettel (Abb. 39: Attribute von „Notizzettel“)

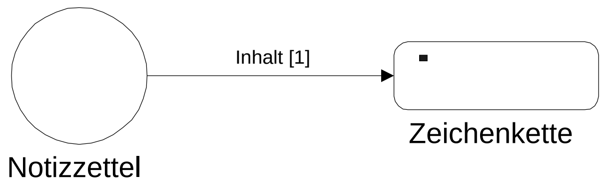

Abb. 39: Attribute von „Notizzettel“

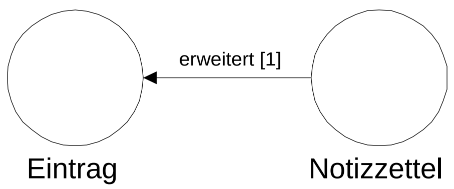

Abb. 40: Erweiterung von „Eintrag“

Beteiligte Objekte im Use-Case „✞Weiterleitung einer Zeichnung“

Zeichnung (Abb. 34: Attribute von „Zeichnungdaten“)

♦*Abstrakte Use-Cases*

Beteiligte Objekte im Use-Case „✞Eintrag auswählen“

Eintragsdaten (Abb. 21: Attribute von „Eintragsdaten“)

Eintragsliste (Abb. 32: Attribute von „Eintragsliste“)

Hauptfenster (Abb. 13: Hauptfenster)

Beteiligte Objekte im Use-Case „✞Userdaten erfassen“

Eintragsdaten (Abb. 21: Attribute von „Eintragsdaten“)

Userdaten (Abb. 27: Vererbungsbeziehungen von „Userdaten“)

Userdaten-Eingabemaske (Abb. 16: Userdaten-Eingabemaske)

Beteiligte Objekte im Use-Case „✞Suchen durch komplexe Abfrage“

Benutzer (Abb. 31: Attribute von „Benutzer“)

Eintrag (Abb. 20: Attribute von „Eintrag“)

Beteiligte Objekte im Use-Case „✞Suchen durch einfache Abfrage“

Benutzer (Abb. 31: Attribute von „Benutzer“)

Eintrag (Abb. 20: Attribute von „Eintrag“)

Eintragsdaten (Abb. 21: Attribute von „Eintragsdaten“)

Beteiligte Objekte im Use-Case „✞Suchen durch Volltextsuche“

Benutzer	(Abb. 31: Attribute von „Benutzer“)
Eintrag	(Abb. 20: Attribute von „Eintrag“)
Eintragsdaten	(Abb. 21: Attribute von „Eintragsdaten“)
Dokumententext	(Abb. 41: Attribute von „Dokumententext“)

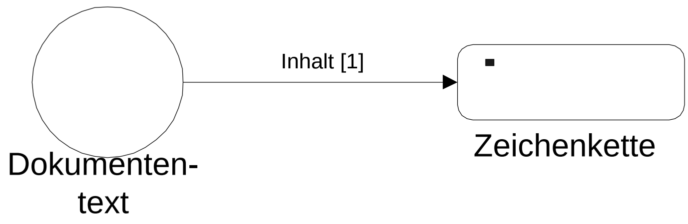

Abb. 41: Attribute von „Dokumententext“

8.2.3 Kontrollobjekte

Kontrollobjekte decken die Funktionalität ab, die nicht durch die beiden ersten Objekttypen Schnittstellenobjekte und Entity-Objekte abgedeckt ist, also beispielsweise Verhaltensweisen von Objekten. Kontrollobjekte kontrollieren damit andere Objekte und können eine Verbindung zwischen Schnittstellenobjekten und Entity-Objekten herstellen. Eine einfache Methode, Kontrollobjekte zu identifizieren, ist, das Verhalten eines Use-Cases durch jeweils ein Kontrollobjekt zu modellieren.

Die Kontrollobjekte werden in diesem Abschnitt aufgezählt. Dazu wird zuerst der Name des Kontrollobjekts genannt, und dann eine kurze Beschreibung der Funktionalität gegeben. In den graphischen Übersichten werden Kontrollobjekte durch dieses Symbol gekennzeichnet:

Abb. 42: Kontrollobjekt

Die Use-Cases werden nun in der durch die in Kapitel 8 gegebene Reihenfolge betrachtet und die in den einzelnen Use-Cases enthaltenen Kontrollobjekte sukzessiv beschrieben. Diese Vorgehensweise entspricht der Vorgehensweise, die Jacobson [JCJÖ92, S. 176] zur Findung der Kontrollobjekte vorschlägt.

Die bei der Betrachtung aller Use-Cases aus Kapitel 8.1.3 gefundenen Kontrollobjekte sind in alphabetischer Reihenfolge:

Fenstersteuerung:

Die **Fenstersteuerung** verarbeitet die Befehle, die der **Benutzer** über die **Funktionsleiste** an das Archivsystem übermittelt, und stellt auf Grundlage der **Einträge** des Archivsystems die Ausgabe für die **Eintragsliste** zusammen.

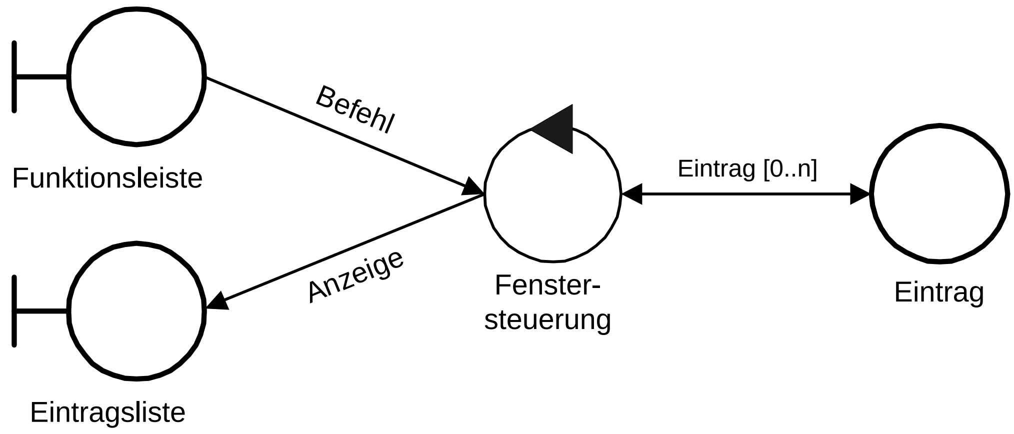

Abb. 43: Fenstersteuerung

Betrachtersteuerung:

Die **Betrachtersteuerung** dient dazu, ein **Dokument** aus dem Archivsystem mit Hilfe des **Betrachters** anzuzeigen.

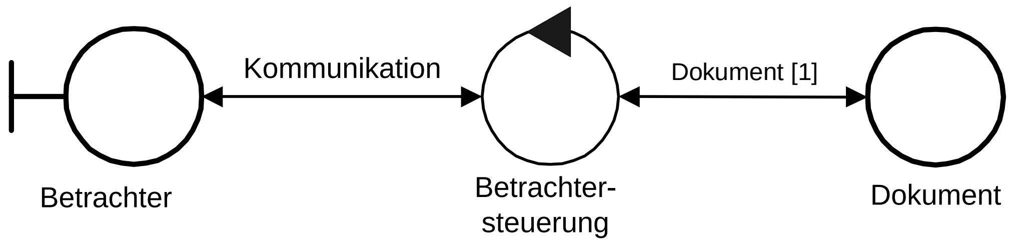

Abb. 44: Betrachtersteuerung

KAPITEL 9

9 Konstruktion des Archivsystems

Unter Verwendung des fertiggestellten Analysemodell folgt der Konstruktionsprozeß. Der Konstruktionsprozeß liefert das Entwurfsmodell und das Implementationsmodell. Das Entwurfsmodell verfeinert die Objektstruktur des Analysemodells unter Berücksichtigung der konkret vorgegebenen Implementationsumgebung. Bei Erstellung des Entwurfsmodells werden erste die Implementation betreffende Entscheidungen getroffen. Wenn das Entwurfsmodell weit genug verfeinert wurde, wird daraus das Implementationsmodell entwickelt, welches letztendlich den kommentierten Programmcode enthält. Mit diesem Programmcode wird das System implementiert.

Der Konstruktionsprozeß wird im Rahmen dieser Arbeit nicht durchgeführt. Statt dessen sollen Anforderungsmodell und Analysemodell anhand einer prototypischen Implementation validiert und die Umsetzbarkeit der Systemarchitektur überprüft werden. Eine prototypische Implementation ist nicht Bestandteil der Softwareentwicklungsmethode OOSE, kann aber nach [JCJÖ92] bei Bedarf verwendet werden.

9.1 Entwurf

Am Beginn der Entwurfsphase steht die Festlegung der tatsächlichen Implementationsumgebung. Alle maßgeblichen Umstände, die die Realisierung des Systems betreffen, müssen untersucht werden. Dies betrifft auch die Integration eines Datenbanksystems, die Handhabung der Prozesse und der Verteilung, die Eigenschaften der Programmiersprache, die Auswahl der Komponenten, aber auch den Werkzeugen zur Oberflächengestaltung. Unter Berücksichtigung dieser Punkte ist aus dem Analysemodell ein erstes Entwurfsmodell zu entwickeln. Dieses zeigt einen ersten Annäherung an die tatsächliche Systemarchitektur. Um diese Architektur stabil und robust gegenüber Änderungen der Implementationsumgebung zu halten, sollten so wenig wie möglich Abweichungen von dem Analysemodell gemacht werden.

Dieses erste Entwurfsmodell wird dann anhand der während der Analysephase erstellten Use-Cases verfeinert. Dabei entstehen unter Zuhilfenahme von Interaktionsdiagrammen die Schnittstellenspezifikationen der Objekte. Das Objektverhalten kann durch Zustandsübergangsgraphen gezeigt werden.

Die Objekte werden dann durch Objektmodule aufgeteilt und implementiert. Bei einer objektorientierten Programmiersprache entsprechen diese Objektmodule den Klassen, allerdings ist die Verwendung einer objektorientierten Programmiersprache keine Voraussetzung für die Implementation.

Abstraktionsmechanismen auf verschiedenen Ebenen vereinfachen die Konstruktion des System. Logisch zusammenhängende Klassen können in Blöcke strukturiert werden. Subsysteme unterteilen das Entwurfsmodell .

9.2 Implementation

Die Implementierung erfolgt dann in der gewählten Programmiersprache und orientiert sich an dem Entwurfsmodell. Diese ist vorzugsweise objektorientiert, um die Konzepte von OOSE leicht umsetzen zu können.

9.3 Prototypische Implementation

Der Konstruktionsprozeß wird im Rahmen dieser Arbeit nicht durchgeführt. Statt dessen sollen Anforderungsmodell und Analysemodell anhand einer prototypischen Implementation validiert und die Umsetzbarkeit der Systemarchitektur überprüft werden.

Eine prototypische Implementation soll ermöglichen, die Grundideen und Konzepte des Systementwurfs praxisnah nachvollziehen und bewerten zu können, wie es schon in Kapitel 4.4 beschrieben wurde. Dazu soll eine prototypische Implementierung elementarer Funktionen durchgeführt werden. Elementare Funktionen sind hier die Funktionen, welche die Grundideen der Systemvisionen umsetzen. Außerdem soll die prototypische Implementierung die Architektur des Archivsystems veranschaulichen. Die prototypische Implementation hilft, die Qualität und Genauigkeit des Anforderungsmodells zu verbessern.

Bei der Implementierung des System löst man sich von der im Laufe des Systementwicklungsprozesses entstandenen logischen Struktur des Systems und legt eine konkrete Implementationsumgebung fest. Bei diesem Archivsystem fällt darunter die Auswahl einer Programmiersprache zur prototypischen Implementation des Systems, eine Architektur, welche die Zusammenarbeit der einzelnen Programmodule veranschaulicht, die Auswahl eines Datenbanksystem zur Datenhaltung und die Festlegung einer Kommunikationsmethode zwischen den Klassen des Systems.

9.3.1 Programmiersprache

Der objektorientierte Entwurf legt es nahe, die prototypische Implementation des Archivsystems in einer objektorientierten Programmiersprache durchzuführen. Java ist eine objektorientierte Programmiersprache und hat darüber hinaus noch den Vorteil, weitgehend plattformunabhängig zu sein und für eine Reihe von Plattformen kostenlos zur Verfügung zu stehen. Die kürzlich freigegebene Version Java 1.1 [Sun97] ermöglicht es außerdem, Oberflächenprogrammierung und Kommunikationsprogrammierung auf einfache Weise durchzuführen. Daher fiel die Entscheidung, Java als Programmiersprache für die prototypische Implementierung zu nehmen.

9.3.2 Architektur

Es ist naheliegend, als grundlegende Systemarchitektur die Client-Server-Architektur zu verwenden. Dabei können die Benutzer mit Hilfe eines Client-Programms die Funktionalität des Archivsystems nutzen. Das Client-Programm baut eine Verbindung zu dem Server-Programm auf, welches für die Datenhaltung und Koordination der Client-Programme zuständig ist. Auf diese Weise können die Client-Programme gering im Umfang gehalten werden. Die Datenhaltung durch ein zentrales Server-Programm ermöglicht es, die Daten auf einfache Weise konsistent zu halten. Die folgende Abbildung (Abb. 45: Client-Server-Architektur) zeigt eine Client-Server-Architektur als Übersicht.

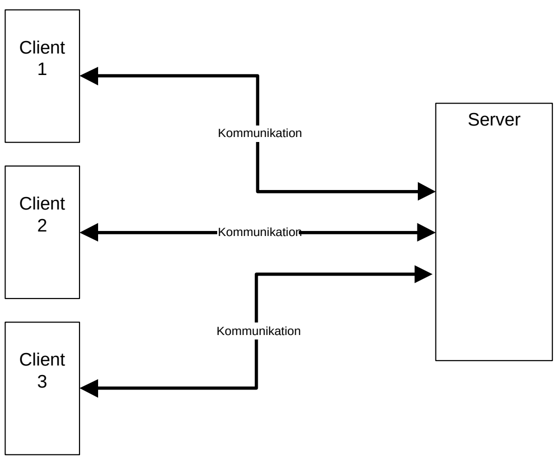

Abb. 45: Client-Server-Architektur

9.3.3 Datenhaltung

Die Entscheidung für Java ist unter anderem aufgrund der Eigenschaften als objektorientierte Programmiersprache gefallen. Bei der Entscheidung zur Auswahl eines Datenbanksystems kann man die gleichen Überlegungen anstellen. Ideal wäre ein objektorientiertes Datenbanksystem, welches sich einfach an Java-Programme anbinden läßt und in einer Client-Server-Architektur einsetzbar ist. Als notwendige Eigenschaften des Datenbanksystems ergeben sich daraus also Objektorientierung, Anbindung an Java sowie die Unterstützung von Transaktionen. Nach Betrachtung verschiedener Datenbanksysteme fiel die Entscheidung auf das Produkt „ObjectStore PSE" der Firma Object Design, da dieses Datenbanksystem die drei Eigenschaften erfüllt. Darüber hinaus hat ObjectStore PSE den Vorteil, daß es kostenlos verfügbar ist und komplett in Java geschrieben wurde, so daß eine Portierung auf verschiedene Betriebssystem möglich ist. Eine spätere Erweiterung auf das umfangreichere, aber nicht mehr kostenlose Produkt „ObjectStore" (nicht zu verwechseln mit „ObjectStore PSE") der gleichen Firma ist denkbar.

Die Firma Object Design ist Mitglied des ODMG, einer Vereinigung von Herstellern von Objektdatenbanken [ODI97]. Das Ziel dieser Organisation ist es, gemeinsame und für die Mitglieder verbindliche Standards für Objektdatenbanken festzulegen [Catt93]. Die ODMG-Organisation hat für die Verwendung von Objektdatenbanken mit Java einen ODMG-Java-Standard festgelegt. Das Persistenz-Modell des ODMG-Java-Standards wird durch die drei Konzepte Transaktionen, benannte Wurzelobjekte und Persistenz durch Erreichbarkeit bestimmt [Deb97]. Eine Transaktion führt die Datenbank von einem gültigen Zustand in einer anderen gültigen Zustand über. Die Vergabe eines Namens an ein Objekt macht dieses zu einem benannten Wurzelobjekt, welches persistent gespeichert wird. Das Objekt kann durch die Namensvergabe später unter diesem Namen in der Datenbank wieder lokalisiert werden. Persistenz durch Erreichbarkeit bedeutet, daß beim Abschluß einer Transaktion nicht nur das Wurzelobjekt selbst persistent gemacht wird, sondern darüber hinaus alle von diesem Wurzelobjekt referenzierten Objekte.

Die Einbindung des Datenbanksystems folgt der Client-Server-Architektur. Der Datenzugriff von einem Client-Programm aus findet über ein Server-Programm statt, der auf die Datenbank zugreift. Diese besondere Architektur der Datenbanknutzung wird auch Multi-Tier-Architektur [Deb97] genannt, da die Datenbankzugriffe über mehrere Schichten (Tier) erfolgen. Die folgende Abbildung (Abb. 46: Multi-Tier-Architektur) verdeutlicht den Zugriff der Client-Programme auf die zuständigen Server-Threads, die dann wiederum die Zugriffe auf die Datenbank tätigen.

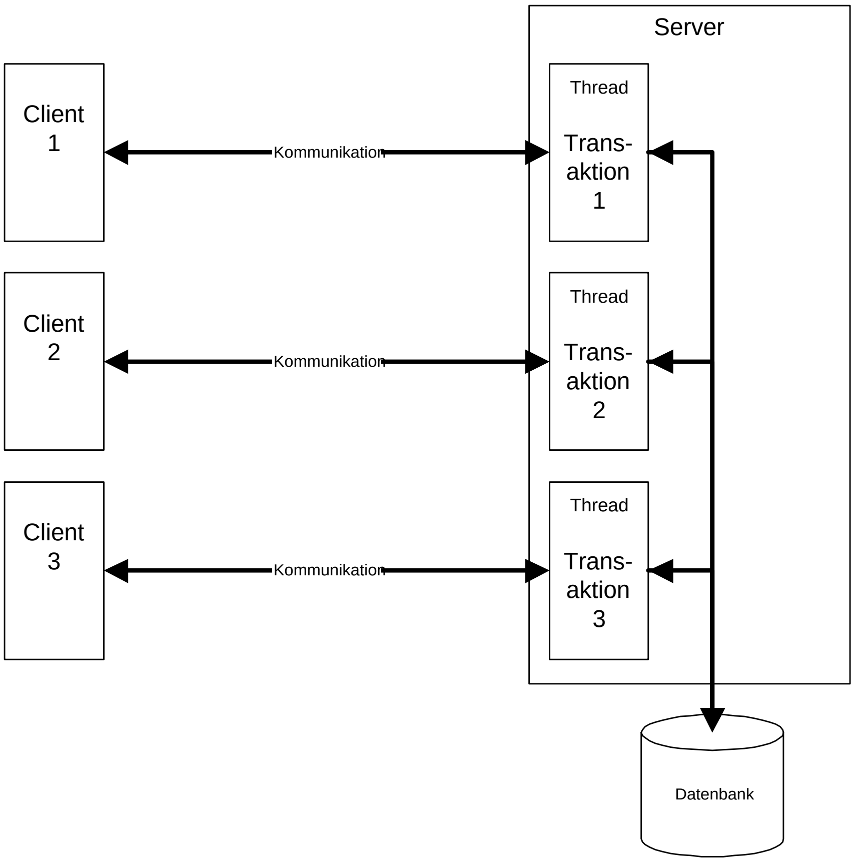

Abb. 46: Multi-Tier-Architektur

9.3.4 Kommunikation

Da Client-Programm und Server-Programm in Java implementiert werden, wird zur Kommunikation zwischen den beiden Programmen ein Mechanismus benutzt, der von Java unterstützt wird. Java wurde in der Version 1.1 um RMI (Remote Method Invocation) erweitert. RMI unterstützt die Entwicklung von verteilten Java-Anwendungen dadurch, daß Methoden eines Javaobjektes auf einem Computersystem durch eine Java-Anwendung auf einem anderen Computersystem aufgerufen werden können. RMI nutzt zur Übertragung der Objektinformationen die Objektserialisierung [Sun97].

KAPITEL 10

10 Betrachtung eines ausgewählten Use-Cases

In diesem Kapitel wird ein ausgewählter Use-Case exemplarisch betrachtet. Dazu wurde der Use-Case „✞Anlegen eines neuen Benutzers“ ausgewählt. Der Use-Case beschreibt, wie ein neuer Benutzer durch den Archivar angelegt wird. Zur Erinnerung wird der Ablauf des Use-Cases hier noch einmal angegeben:

Beschreibung des Use-Cases „ ✞Anlegen eines neuen Benutzers“

Dieser Use-Case beschreibt das Anlegen eines neuen Benutzers.

Vorbedingung:

(1) Der **Benutzer** ist ✌**Archivar**.

Ablauf:

(1) Es erscheint der **Neuer-Benutzer-Dialog** (siehe Kapitel 8.1.7. Schnittstellenbeschreibung Neuer-Benutzer-Dialog und Abb. 14: Neuer-Benutzer-Dialog).
(2) Das Eingabefeld „Benutzername“ wird mit dem Namen des **Benutzers**, der neu anzulegen ist, ausgefüllt.
(3) In dem Eingabefeld „Paßwort“ kann ein Paßwort voreingestellt werden, welches dem neuen Benutzer mitzuteilen ist.
(4) Aus einer Liste von Rollen kann schließlich eine ausgewählt werden, die dem neuen Benutzer zugeordnet wird.

Nachbedingung:

(1) Der neue Benutzer ist dem Archivsystem bekannt, und der neue Benutzer kann das System benutzen (siehe Use-Case „✞Sitzungsbeginn“).

Dieser Use-Case wird jetzt mit Hilfe der prototypischen Implementation ausgeführt:

Der Benutzer hat über die Funktionsleiste (siehe Abb. 47: Funktionsauswahl) die Funktion „Neuen

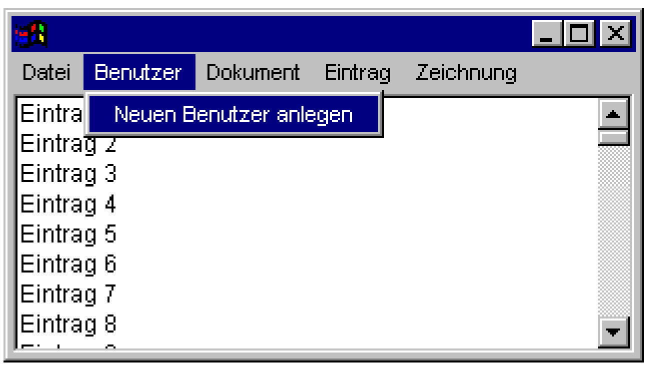

Abb. 47: Funktionsauswahl

Benutzer anlegen“ gewählt. Das System überprüft nun, ob der Benutzer wirklich die Rolle Archivar besitzt (Vorbedingung 1). Ist das der Fall, erscheint der Neuer-Benutzer-Dialog (Ablauf 1, siehe Abb. 48: Dateneingabe). Die Felder werden entsprechend der Beschreibung des Use-Cases (Ablauf 2,3 und 4)

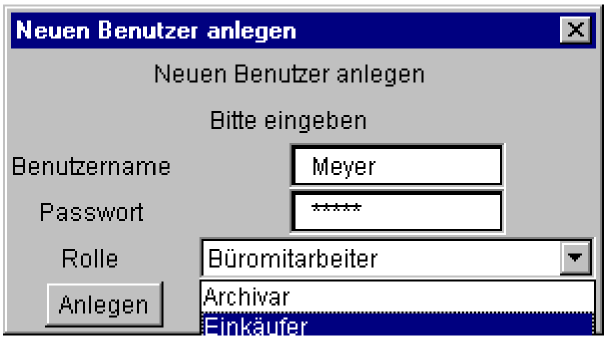

Abb. 48: Dateneingabe

ausgefüllt. Durch Betätigen der Schaltfläche „Anlegen“ schließlich werden die Daten in das Archivsystem übernommen (Nachbedingung 1) und der neu angelegte Benutzer kann das Archivsystem nutzen.

KAPITEL 11

11 Zusammenfassung und Ausblick

11.1 Zusammenfassung

Diese Arbeit beschreibt die Entwicklung eines Archivsystems. Hierzu wurde im Kapitel 1 zunächst die Motivation für die Arbeit gegeben. In Kapitel 2 wurde eine Einführung in die Anwendungsgebiete für Archivsystem durch sogenannte Szenarien durchgeführt. Die dabei erkannten Verbesserungsmöglichkeiten durch eine Computerunterstützung wurden anschließend in Kapitel 3 detailliert betrachtet. Unter Berücksichtigung der Verbesserungsmöglichkeiten wurden in Kapitel 4 dann Systemvisionen entworfen, die erste Ansätze zur Lösung der Archivproblematik mit Hilfe eines Archivsystems zeigen sollen. Auch dabei wurden wieder die eingeführten Szenarien unterschieden und für jedes Szenario eine eigene Systemvision entworfen.

Im Kapitel 0 wurden Dokumenten-Management-Systeme näher betrachtet und die verschiedenen konzeptionellen Schwerpunkte diskutiert, um Komponenten der verschiedenen Dokumenten-Management-Systeme dahingehend bewerten zu können, ob eine Berücksichtigung bestimmter Komponenten bei der Erstellung des zu entwickelnden Archivsystems hilfreich sein kann. Dazu wurden die drei Gruppen Archivierungsysteme, Retrievalsysteme und Workflowsysteme, in die sich Dokumenten-Management-Systeme unterteilen lassen, untersucht.

Im Kapitel 0 wurde auf die Vor- und Nachteile von elektronischen Archivierungssystemen eingegangen. Außerdem wurden die Voraussetzungen geklärt, unter denen Archivsysteme eingesetzt werden können und welche Bedingungen dabei für die Dokumente gelten.

Die Auswahl der Entwicklungsmethode wurde in Kapitel 7 dargestellt und die gewählte Entwicklungsmethode in einer Übersicht beschrieben. Diese Entwicklungsmethode ist Grundlage für die weiteren Kapitel dieser Arbeit.

Die Analyse des Archivsystems stand in Kapitel 8 im Mittelpunkt. Als erstes wurden die Anforderungen an das System modelliert. Dazu werden zunächst die Personengruppen aufgezählt, die das System benutzen. Danach wird ein Glossar erstellt, welches die wichtigen Begriffe definiert, die bei der Analyse benutzt werden. Die Benutzung des Systems veranschaulichen dann die Use-Cases, die einen Ablauf einer einzelnen Handlung beschreiben. Dabei wurde dann auch die Retrievalkomponente „Volltextsuche“ und die Workflowkomponente „Weiterleiten“ benutzt. Auf Grundlage des entstandenen Modells wurde dann das Analysemodell entwickelt, welches beteiligte Objekte näher beschreibt.

Die Auswahl einer konkreten Implementationsumgebung wurde in Kapitel 9 dargestellt. Abschließend wurde in Kapitel 10 ein ausgewählter Use-Case exemplarisch betrachtet.

Die Programmdokumentation zur prototypischen Implementierung findet sich in Anhang A wieder, die Programmquelltexte finden sich in Anhang B wieder.

11.2 Ausblick

Prinzipiell ließ sich zeigen, daß Komponenten aus den Bereichen Retrieval- und Workflowsystemen auch in einem Archivsystem sinnvoll einzusetzen sind. Um jedoch ein in der Praxis einsetzbares Archivsystem zu entwickeln, ist noch Feinarbeit in vielen Bereichen notwendig.

Die prototypische Implementation erfolgte in Java mit dem Datenbanksystem ObjectStore PSE und zeigte keinen wesentlichen Nachteil. In der Praxis sollte jedoch überprüft werden, ob das Datenbanksystem ObjectStore PSE nicht durch das Datenbanksystem ObjectStore, welches allerdings nicht kostenlos verfügbar ist, ersetzt werden kann.

Leider lassen sich die Client-Programme noch nicht in einem WWW-Browser ausführen, da die verfügbaren WWW-Browser noch nicht die Java-Version 1.1 unterstützen. Mit der Zeit werden solche WWW-Browser jedoch verfügbar sein.

Die Tatsache, daß das Client-Programm noch nicht in einem WWW-Browser ausführbar ist, führte zu der Überlegung, ob eine Realisierung des Client-Programms nur über Verwendung des HTTP-Protokolls möglich wäre. Die Darstellung der Einträge würde dann in Textform geschehen, und der Zugriff auf weitere Informationen zu einem Eintrag könnte über Links auf Dokumente, Notizzettel und so weiter erfolgen. Das Server-Programm würde auch dann die Datenbankzugriffe steuern und die Ausgaben mittels HTTP aufbereiten.

Die Verwaltung der Dokumente wurde für die prototypische Implementation auf beispielhaft ausgewählte Dokumententypen wie Rechnung oder Zeichnung mit den dazugehörigen fest vorgegebenen Userdaten eingeschränkt. Wünschenswert ist selbstverständlich, daß der Benutzer selbst solche Dokumententypen erzeugen kann und die Userdaten individuell anpassen kann. Auch eine Möglichkeit, über Nummernkreise hinaus weitere Hierarchiestufen anlegen zu können, wäre dann vorzusehen.

Allgemein wäre jetzt die Frage zu diskutieren, ob ein komplett neuer Entwurf durchgeführt wird oder evolutionäres Prototyping angewendet werden soll. Ein komplett neuer Entwurf ist nötig, wenn die prototypische Implementation zeigt , daß . Das läßt sich allerdings anhand der vorliegenden prototypischen Implementation nicht erkennen. Evolutionäres Prototyping wird angewendet, um den Entwurf anhand der Erfahrungen zu verbessern. In diesem Fall kann die Auswertung der prototypischen Implementation helfen, die Qualität und Genauigkeit des Anforderungsmodells zu verbessern. Dieser Schritt wird allerdings im Rahmen dieser Arbeit nicht mehr ausgeführt.

Inzwischen ist die objektorientierte Entwicklungsmethode „Unified Method“, die anfangs nur als Draft in der Version 0.8 [BoRu95] zur Verfügung stand, weiterentwickelt worden. Die Methode wurde umbenannt und nennt sich jetzt „Unified Modelling Language“, kurz UML. UML ist die Vereinigung der Methoden von Booch (OOAD, vgl. [Boo94]), Rumbaugh (OMT, vgl. [Rumb91]) und Jacobson (OOSE, vgl. [JCJÖ92]). Eine mögliche Verwendung der UML-Methode zur Erweiterung der Modelle des Use-Case-Driven-Approach unter Verwendung der Ergebnisse einer Auswertung der prototypischen Implementation wäre eine interessante Fortsetzung der Arbeit.

KAPITEL 12

12 Anhang A

Im Anhang A werden die Programme, die im Rahmen der prototypischen Implementierung erstellt worden, dokumentiert.

Zur Dokumentation wird das Dienstprogramm „javadoc“ [Sun97] verwendet, welches Bestandteil des Java SDKs ist. Javadoc generiert automatische Programmdokumentation aus den Programmquelltexten und besonders markierten Kommentaren. Die erzeugte Klassendokumentation beschreibt eine Klasse und ihre Vererbungshierarchie und erstellt Verzeichnisse und Beschreibungen der Klassenmethoden und Klassenvariablen. Da im Anhang B der Programmquelltext abgedruckt ist, wird hier auf die detaillierte Angabe der Methodenbeschreibung und Parameterübergabe verzichtet.

Die Dokumentation ist aufgeteilt in die Dokumentation des Client-Programms und in die Dokumentation des Server-Programms. Davor steht noch die Klassenhierarchie, die die Hierarchie der Klassen untereinander und in der Hierarchie der Systemklassen veranschaulicht.

Klassenhierarchie

- class java.lang.Object
  - class COM.odi.demo.archiv.AS_Objekt
    - class COM.odi.demo.archiv.AS_Benutzer
    - class COM.odi.demo.archiv.AS_Blatt
    - class COM.odi.demo.archiv.AS_Datei
    - class COM.odi.demo.archiv.AS_Dokumententext
    - class COM.odi.demo.archiv.AS_Eintrag
    - class COM.odi.demo.archiv.AS_Eintragsdaten
    - class COM.odi.demo.archiv.AS_Notizzettel
    - class COM.odi.demo.archiv.AS_Nummernkreis
    - class COM.odi.demo.archiv.AS_Systemdaten
    - class COM.odi.demo.archiv.AS_Userdaten
      - class COM.odi.demo.archiv.AS_Bestellungsdaten
      - class COM.odi.demo.archiv.AS_Rechnungsdaten
      - class COM.odi.demo.archiv.AS_Zeichnungsdaten
    - class COM.odi.demo.archiv.AS_Vektor
      - class COM.odi.demo.archiv.AS_Archiv
      - class COM.odi.demo.archiv.AS_Dokument
      - class COM.odi.demo.archiv.AS_Eintragsliste
    - class COM.odi.demo.archiv.AS_Version
  - class COM.odi.demo.archiv.Client
  - class java.awt.Component (implements java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable)
    - class java.awt.Container
      - class java.awt.Window
        - class java.awt.Dialog
          - class COM.odi.demo.archiv.NeuerBenutzerDialog
          - class COM.odi.demo.archiv.PasswortDialog
        - class java.awt.Frame (implements java.awt.MenuContainer)
          - class COM.odi.demo.archiv.Hauptfenster
  - interface COM.odi.demo.archiv.Interface (extends java.rmi.Remote)
  - class java.awt.MenuComponent (implements java.io.Serializable)

        - class java.awt.MenuBar (implements java.awt.MenuContainer)
            - class COM.odi.demo.archiv.Funktionsleiste
    - class java.rmi.server.RemoteObject (implements java.rmi.Remote, java.io.Serializable)
        - class java.rmi.server.RemoteServer
            - class java.rmi.server.UnicastRemoteObject
                - class COM.odi.demo.archiv.Impl (implements COM.odi.demo.archiv.Interface)
    - class COM.odi.demo.archiv.Server

12.2 Programmdokumentation Client

12.2.1 Class COM.odi.demo.archiv.AS_Archiv

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Vektor
                   |
                   +----COM.odi.demo.archiv.AS_Archiv
```

public class **AS_Archiv**

extends AS_Vektor

Die Klasse AS_Archiv ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Archiv stellt die Entsprechung des Entity-Objektes "Archiv" dar.

Index der Kontruktoren

AS_Archiv()

AS_Archiv(String)

Index der Methoden

getEinträge()
Methode zum indirekten Zugriff auf das private Attribute "Einträge"

getObjektName()
Liefert einen String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

setEinträge(OSVector)
Methode zum indirekten Setzen des privaten Attributes "Einträge"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.2 Class COM.odi.demo.archiv.AS_Benutzer

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Benutzer
```

public class **AS_Benutzer**

extends AS_Objekt

Die Klasse AS_Benutzer ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Benutzer stellt die Entsprechung des Entity-Objektes "Benutzer" dar.

Index der Kontruktoren

AS_Benutzer()

Index der Methoden

getBenutzername()
Methode zum indirekten Zugriff auf das private Attribute "Benutzername"

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

getPasswort()
Methode zum indirekten Zugriff auf das private Attribute "Passwort"

getRolle()
Methode zum indirekten Zugriff auf das private Attribute "Rolle"

setBenutzername(String)
Methode zum indirekten Setzen des privaten Attributes "Benutzername"

setPasswort(String)
Methode zum indirekten Setzen des privaten Attributes "Passwort"

setRolle(String)
Methode zum indirekten Setzen des privaten Attributes "Rolle"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.3 Class COM.odi.demo.archiv.AS_Bestellungsdaten

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Userdaten
                   |
                   +----COM.odi.demo.archiv.AS_Bestellungsdaten
```

public class **AS_Bestellungsdaten**

extends AS_Userdaten

Die Klasse AS_Bestellungsdaten ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Bestellungsdaten stellt die Entsprechung des Entity-Objektes "Bestellungsdaten" dar.

Index der Kontruktoren

AS_Bestellungsdaten()

Index der Methoden

getArtikelname()
Methode zum indirekten Zugriff auf das private Attribute "Artikelname"

getBestellbetrag()
Methode zum indirekten Zugriff auf das private Attribute "Bestellbetrag"

getBestelldatum()
Methode zum indirekten Zugriff auf das private Attribute "Bestelldatum"

getBesteller()
Methode zum indirekten Zugriff auf das private Attribute "Besteller"

getEingangsdatum()
Methode zum indirekten Zugriff auf das private Attribute "Eingangsdatum"

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

setArtikelname(String)
Methode zum indirekten Setzen des privaten Attributes "Artikelname"

setBestellbetrag(Integer)
Methode zum indirekten Setzen des privaten Attributes "Bestellbetrag"

setBestelldatum(String)
Methode zum indirekten Setzen des privaten Attributes "Bestelldatum"

setBesteller(String)
Methode zum indirekten Setzen des privaten Attributes "Besteller"

setEingangsdatum(String)
Methode zum indirekten Setzen des privaten Attributes "Eingangsdatum"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.4 Class COM.odi.demo.archiv.AS_Blatt

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Blatt
```

public class **AS_Blatt**

extends AS_Objekt

Die Klasse AS_Blatt ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Blatt stellt die Entsprechung des Entity-Objektes "Blatt" dar.

Index der Kontruktoren

AS_Blatt()

Index der Methoden

getBeschreibung()
Methode zum indirekten Zugriff auf das private Attribute "Beschreibung"

getBlattnummer()
Methode zum indirekten Zugriff auf das private Attribute "Blattnummer"

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

setBeschreibung(String)
Methode zum indirekten Setzen des privaten Attributes "Beschreibung"

setBlattnummer(Integer)
Methode zum indirekten Setzen des privaten Attributes "Blattnummer"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.5 Class COM.odi.demo.archiv.AS_Datei

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Datei
```

public class **AS_Datei**

extends AS_Objekt

Die Klasse AS_Datei ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Datei stellt die Entsprechung des Entity-Objektes "Datei" dar.

Index der Kontruktoren

AS_Datei()

Index der Methoden

getDateigröße()
Methode zum indirekten Zugriff auf das private Attribute "Dateigröße"

getDateiname()
Methode zum indirekten Zugriff auf das private Attribute "Dateiname"

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

getSpeicherort()
Methode zum indirekten Zugriff auf das private Attribute "Speicherort"

setDateigröße(Integer)
Methode zum indirekten Setzen des privaten Attributes "Dateigröße"

setDateiname(String)
Methode zum indirekten Setzen des privaten Attributes "Dateiname"

setSpeicherort(String)
Methode zum indirekten Setzen des privaten Attributes "Speicherort"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.6 Class COM.odi.demo.archiv.AS_Dokument

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Vektor
                   |
                   +----COM.odi.demo.archiv.AS_Dokument
```

public class **AS_Dokument**

extends AS_Vektor

Die Klasse AS_Dokument ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Dokument stellt die Entsprechung des Entity-Objektes "Dokument" dar.

Index der Kontruktoren

AS_Dokument()

Index der Methoden

getDateien()
Methode zum indirekten Zugriff auf das private Attribute "Dateien"

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

setDateien(OSVector)
Methode zum indirekten Setzen des privaten Attributes "Dateien"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.7 Class COM.odi.demo.archiv.AS_Dokumententext

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Dokumententext
```

public class **AS_Dokumententext**

extends AS_Objekt

Die Klasse AS_Dokumententext ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Dokumententext stellt die Entsprechung des Entity-Objektes "Dokumententext" dar.

Index der Kontruktoren

- **AS_Dokumententext**()

Index der Methoden

- **getInhalt**()
  Methode zum indirekten Zugriff auf das private Attribute "Inhalt"
- **getObjektName**()
  Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
- **setInhalt**(String)
  Methode zum indirekten Setzen des privaten Attributes "Inhalt"
- **toString**()
  Erzeugt eine lesbare Stringausgabe des Objekts

12.2.8 Class COM.odi.demo.archiv.AS_Eintrag

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Eintrag
```

public class **AS_Eintrag**

extends AS_Objekt

```
Die Klasse AS_Eintrag ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.

Die Klasse AS_Eintrag stellt die Entsprechung des Entity-Objektes "Eintrag" dar.
```

Index der Kontruktoren

- **AS_Eintrag**()

Index der Methoden

getDokument()
Methode zum indirekten Zugriff auf das private Attribute "Dokument"

getEintragsdaten()
Methode zum indirekten Zugriff auf das private Attribute "Eintragsdaten"

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

setDokument(AS_Dokument)
Methode zum indirekten Setzen des privaten Attributes "Dokument"

setEintragsdaten(AS_Eintragsdaten)
Methode zum indirekten Setzen des privaten Attributes "Eintragsdaten"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.9 Class COM.odi.demo.archiv.AS_Eintragsdaten

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Eintragsdaten
```

public class **AS_Eintragsdaten**

extends AS_Objekt

Die Klasse AS_Eintragsdaten ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Eintragsdaten stellt die Entsprechung des Entity-Objektes "Eintragsdaten" dar.

Index der Kontruktoren

AS_Eintragsdaten()

Index der Methoden

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

getSystemdaten()
Methode zum indirekten Zugriff auf das private Attribute "Systemdaten"

getUserdaten()
Methode zum indirekten Zugriff auf das private Attribute "Userdaten"

setSystemdaten(AS_Systemdaten)
Methode zum indirekten Setzen des privaten Attributes "Systemdaten"

setUserdaten(AS_Userdaten)
Methode zum indirekten Setzen des privaten Attributes "Userdaten"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.10 Class COM.odi.demo.archiv.AS_Eintragsliste

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Vektor
                   |
                   +----COM.odi.demo.archiv.AS_Eintragsliste
```

public class **AS_Eintragsliste**

extends AS_Vektor

Die Klasse AS_Eintragsliste ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Eintragsliste stellt die Entsprechung des Entity-Objektes "Eintragsliste" dar.

Index der Kontruktoren

AS_Eintragsliste()

AS_Eintragsliste(String)

Index der Methoden

getEinträge()
Methode zum indirekten Zugriff auf das private Attribute "Einträge"

getObjektName()
Liefert einen String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

setEinträge(OSVector)
Methode zum indirekten Setzen des privaten Attributes "Einträge"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.11 Class COM.odi.demo.archiv.AS_Notizzettel

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Notizzettel
```

public class **AS_Notizzettel**

extends AS_Objekt

Die Klasse AS_Notizzettel ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Notizzettel stellt die Entsprechung des Entity-Objektes "Notizzettel" dar.

Index der Kontruktoren

- **AS_Notizzettel**()

Index der Methoden

- **getInhalt**()
  Methode zum indirekten Zugriff auf das private Attribute "Inhalt"
- **getObjektName**()
  Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
- **setInhalt**(String)
  Methode zum indirekten Setzen des privaten Attributes "Inhalt"
- **toString**()
  Erzeugt eine lesbare Stringausgabe des Objekts

12.2.12 Class COM.odi.demo.archiv.AS_Nummernkreis

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Nummernkreis
```

public class **AS_Nummernkreis**

extends AS_Objekt

Die Klasse AS_Nummernkreis ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Nummernkreis stellt die Entsprechung des Entity-Objektes "Datei" dar.

Index der Kontruktoren

- **AS_Nummernkreis**()

Index der Methoden

- **getEndnummer**()
  Methode zum indirekten Zugriff auf das private Attribute "Endnummer"
- **getName**()
  Methode zum indirekten Zugriff auf das private Attribute "Name"
- **getObjektName**()
  Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
- **getStartnummer**()
  Methode zum indirekten Zugriff auf das private Attribute "Startnummer"
- **setEndnummer**(Integer)
  Methode zum indirekten Setzen des privaten Attributes "Endnummer"
- **setName**(String)
  Methode zum indirekten Setzen des privaten Attributes "Name"
- **setStartnummer**(Integer)
  Methode zum indirekten Setzen des privaten Attributes "Startnummer"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.13 Class COM.odi.demo.archiv.AS_Objekt

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
```

public class **AS_Objekt**

extends Object

Die Klasse AS_Objekt ist eine "persistence-capable"-Klasse.
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Sie ist "abstract" und kann nicht instantiiert werden.
Die Klasse AS_Objekt ist die Klasse, von der alle anderen AS_ Klassen erben.

Index der Kontruktoren

AS_Objekt()

Index der Methoden

getInstanzName()
Methode zum indirekten Zugriff auf das private Attribute "InstanzName"

getObjektName()
Liefert einen String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

hashCode()
Diese Methode ist für die Datenbank nötig, wird aber in dieser Implementation nicht weiter verwendet.

setInstanzName(String)
Methode zum indirekten Setzen des privaten Attributes "InstanzName"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.14 Class COM.odi.demo.archiv.AS_Rechnungsdaten

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Userdaten
                   |
                   +----COM.odi.demo.archiv.AS_Rechnungsdaten
```

public class **AS_Rechnungsdaten**

extends AS_Userdaten

Die Klasse AS_Rechnungsdaten ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Datei stellt die Entsprechung des Entity-Objektes "Rechnungsdaten" dar.

Index der Kontruktoren

AS_Rechnungsdaten()

Index der Methoden

getEingangsdatum()
Methode zum indirekten Zugriff auf das private Attribute "Eingangsdatum"

getKostenstelle()
Methode zum indirekten Zugriff auf das private Attribute "Kostenstelle"

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

getRechnungsbetrag()
Methode zum indirekten Zugriff auf das private Attribute "Rechnungsbetrag"

getRechnungsdatum()
Methode zum indirekten Zugriff auf das private Attribute "Rechnungsdatum"

getRechnungssteller()
Methode zum indirekten Zugriff auf das private Attribute "Rechnungssteller"

setEingangsdatum(String)
Methode zum indirekten Setzen des privaten Attributes "Eingangsdatum"

setKostenstelle(Integer)
Methode zum indirekten Setzen des privaten Attributes "Kostenstelle"

setRechnungsbetrag(Integer)
Methode zum indirekten Setzen des privaten Attributes "Rechnungsbetrag"

setRechnungsdatum(String)
Methode zum indirekten Setzen des privaten Attributes "Rechnungsdatum"

setRechnungssteller(String)
Methode zum indirekten Setzen des privaten Attributes "Rechnungssteller"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.15 Class COM.odi.demo.archiv.AS_Systemdaten

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Systemdaten
```

public class **AS_Systemdaten**

extends AS_Objekt

Die Klasse AS_Systemdaten ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Systemdaten stellt die Entsprechung des Entity-Objektes "Systemdaten" dar.

Index der Kontruktoren

AS_Systemdaten()

Index der Methoden

getBenutzername()
Methode zum indirekten Zugriff auf das private Attribute "Benutzername"

getDatum()
Methode zum indirekten Zugriff auf das private Attribute "Datum"

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

setBenutzername(String)
Methode zum indirekten Setzen des privaten Attributes "Benutzername"

setDatum(String)
Methode zum indirekten Setzen des privaten Attributes "Datum"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.16 Class COM.odi.demo.archiv.AS_Userdaten

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Userdaten
```

public class **AS_Userdaten**

extends AS_Objekt

Die Klasse AS_Userdaten ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Im System ist sie allerdings abstract und kann daher nicht instantiiert werden.
Die Klasse AS_Userdaten stellt die Entsprechung des Entity-Objektes "Userdaten" dar.

Index der Kontruktoren

AS_Userdaten()

Index der Methoden

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.17 Class COM.odi.demo.archiv.AS_Vektor

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Vektor
```

public class **AS_Vektor**

extends AS_Objekt

Die Klasse AS_Vektor ist eine "persistence-capable"-Klasse.
Sie ist "abstract" und kann keine diskrete Instanz in der Datenbank sein.
Die Klasse AS_Vektor ist die Klasse, von der alle anderen AS_ Klassen erben,
die einen Vektor verwalten.

Index der Kontruktoren

AS_Vektor()

Index der Methoden

getObjektName()
Liefert einen String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

getVektor()
Methode zum indirekten Zugriff auf das private Attribute "Vektor"

setVektor(OSVector)
Methode zum indirekten Setzen des privaten Attributes "Vektor"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.18 Class COM.odi.demo.archiv.AS_Version

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Version
```

public class **AS_Version**

extends AS_Objekt

Die Klasse AS_Version ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
[illegible]tsprechung des Entity-Objektes "Version" dar.

Index der Kontruktoren

AS_Version()

Index der Methoden

getBeschreibung()
Methode zum indirekten Zugriff auf das private Attribute "Beschreibung"

getObjektName()
Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

getVersionnummer()
Methode zum indirekten Zugriff auf das private Attribute "Versionnummer"

setBeschreibung(String)
Methode zum indirekten Setzen des privaten Attributes "Beschreibung"

setVersionnummer(Integer)
Methode zum indirekten Setzen des privaten Attributes "Versionnummer"

toString()
Erzeugt eine lesbare Stringausgabe des Objekts

12.2.19 Class COM.odi.demo.archiv.AS_Zeichnungsdaten

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.AS_Objekt
           |
           +----COM.odi.demo.archiv.AS_Userdaten
                   |
                   +----COM.odi.demo.archiv.AS_Zeichnungsdaten
```

public class **AS_Zeichnungsdaten**

extends AS_Userdaten

Die Klasse AS_Zeichnungsdaten ist eine "persistence-capable"-Klasse,
d.h. sie kann in der Persistenzdatenbank gespeichert werden.
Die Klasse AS_Zeichnungdaten stellt die Entsprechung des Entity-Objektes "Zeichnungsdaten" dar.

Index der Kontruktoren

AS_Zeichnungsdaten()

Index der Methoden

getBeschreibung()
 Methode zum indirekten Zugriff auf das private Attribute "Beschreibung"

getFreigabe()
 Methode zum indirekten Zugriff auf das private Attribute "Freigabe"

getObjektName()
 Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.

getZeichnungsnummer()
 Methode zum indirekten Zugriff auf das private Attribute "Zeichnungsnummer"

setBeschreibung(String)
 Methode zum indirekten Setzen des privaten Attributes "Beschreibung"

setFreigabe(boolean)
 Methode zum indirekten Setzen des privaten Attributes "Freigabe"

setZeichnungsnummer(Integer)
 Methode zum indirekten Setzen des privaten Attributes "Zeichnungsnummer"

toString()
 Erzeugt eine lesbare Stringausgabe des Objekts

12.2.20 Class COM.odi.demo.archiv.Client

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.Client
```

public class **Client**

extends Object

Die Klasse Client ist das Hauptprogramm auf Clientseite.
Es stellt die Verbindung zum Server her und
instantiiert die Schnittstellenobjekte.

Client()

Index der Kontruktoren

Index der Methoden

main(String[])

Die Funktion main wird beim Start von der Komnmandozeile aufgerufen

12.2.21 Class COM.odi.demo.archiv.Funktionsleiste

```
ponent
          |
          +----java.awt.MenuBar
                  |
                  +----COM.odi.demo.archiv.Funktionsleiste
```

public class **Funktionsleiste**

extends MenuBar

Die Klasse Funktionsleiste ist die Entsprechung des Schnittstellenobjektes "Funktionsleiste"
Es empfängt die Benutzerkommandos und führt die Aktionen aus.

Index der Kontruktoren

Funktionsleiste(Frame)

Erzeugt die Funktionsleiste

12.2.22 Class COM.odi.demo.archiv.Hauptfenster

```
nt
          |
          +----java.awt.Container
                  |
                  +----java.awt.Window
                          |
                          +----java.awt.Frame
                                  |
                                  +----COM.odi.demo.archiv.Hauptfenster
```

public class **Hauptfenster**

extends Frame

Die Klasse Hauptfenster ist die Entsprechung des Schnittstellenobjektes "Hauptfenster"
Es enthält die Funktionsleiste und die Eintragsliste.

Index der Kontruktoren

Hauptfenster(Object)

Erzeugt das Hauptfenster

12.2.23 Class COM.odi.demo.archiv.NeuerBenutzerDialog

```
nt
          |
          +----java.awt.Container
                  |
                  +----java.awt.Window
                          |
                          +----java.awt.Dialog
                                  |
                                  +----COM.odi.demo.archiv.NeuerBenutzerDialog
```

public class **NeuerBenutzerDialog**

extends Dialog

Die Klasse NeuerBenutzerDialog ist die Entsprechung des Schnittstellenobjektes "NeuerBenutzerDialog".
Mit Hilfe dieses Dialogs wird ein neuer Benutzer angelegt.

Index der Kontruktoren

NeuerBenutzerDialog(Frame, String)

Erzeugt den NeuerBenutzerDialog

Index der Methoden

action(Event, Object)

getBenutzername()

Gibt den Benutzernamen zurück

getOK()

Zeigt an, ob der Dialog abgebrochen wurde

getPasswort()

Gibt das voreingestellte Passwort zurück

getRolle()

Gibt die zugewiesene Rolle zurück

12.2.24 Class COM.odi.demo.archiv.PasswortDialog

```
nt
          |
          +----java.awt.Container
                  |
                  +----java.awt.Window
                          |
                          +----java.awt.Dialog
                                  |
                                  +----COM.odi.demo.archiv.PasswortDialog
```

public class **PasswortDialog**

extends Dialog

Die Klasse PasswortDialog stellt die Entsprechung des
Schnittstellen-Objektes "PasswortDialog" dar.

Index der Kontruktoren

PasswortDialog(Frame, String)

Index der Methoden

action(Event, Object)

getBenutzername()

getBenutzername liefert den eingegebenen Benutzernamen

getOK()

getOK gibt an, ob der Dialog abgebrochen wurde

getPasswort()

getPasswort liefert das eingegebene Passwort

12.3 Programmdokumentation Server

12.3.1 Class COM.odi.demo.archiv.Impl

```
java.lang.Object
   |
   +----java.rmi.server.RemoteObject
           |
           +----java.rmi.server.RemoteServer
                   |
                   +----java.rmi.server.UnicastRemoteObject
                           |
                           +----COM.odi.demo.archiv.Impl
```

public class **Impl**

extends UnicastRemoteObject

implements Interface

Die Klasse Impl implementiert die Methoden auf der Serverseite,
die dann von den Clientprogrammen aufgerufen werden können.

Index der Kontruktoren

Impl(String, String)
Impl führt das Naming.rebind aus und initialisiert die Datenbank

Index der Methoden

checkLogin(String, String)
Überprüft das Passwort und gibt eine gültige Rolle zurück

getEinträge()
gibt die Einträge zur Darstellung im Hauptfenster zurück

getHello()
`Gibt einen String zur Begrüssung des Clients zurück.`

12.3.2 Interface COM.odi.demo.archiv.Interface

public interface **Interface**

extends Remote

Das Interface definiert die Methoden,
die von Clients auf dem Server aufgerufen werden können.
Aus dieser Interfacedefinition erzeugt ein Interface-Compiler (rmic) Stubs.

Index der Methoden

checkLogin(String, String)

getEinträge()

getHello()

12.3.3 Class COM.odi.demo.archiv.Server

```
java.lang.Object
   |
   +----COM.odi.demo.archiv.Server
```

public class **Server**

extends Object

Die Klasse Server ist das Hauptprogramm auf Serverseite.
Es instantiiert das Interface, über das die Clients
auf die Servermethode zugreifen.

Index der Kontruktoren

Server()

Index der Methoden

main(String[])

Die Funktion main wird beim Start von der Komnmandozeile aufgerufen

KAPITEL 13

13 Anhang B

Im Anhang B wird der Programmquelltext, der im Rahmen der prototypischen Implementierung erstellt wurde, dokumentiert.

Die Dokumentation beschränkt sich auf eine einfache Ausgabe der Programmquelltexte. Die Programmquelltexte werden Klassenweise angegeben.

Die Programmquelltexte sind aufgeteilt in die Programmquelltexte des Client-Programms und in die Programmquelltexte des Server-Programms.

13.1 Programmquelltext Client

13.1.1 AS_Archiv

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems
(AS) enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Archiv ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Archiv stellt die Entsprechung des Entity-Objektes "Archiv" dar.
*/

public class AS_Archiv extends AS_Vektor{

        // Private Variabeln

        public AS_Archiv (String InstanzName) {
                this.setInstanzName(InstanzName);
        }

        public AS_Archiv () {
                this("Ohne Name");
        }

        public String getObjektName() {
                return "AS_Archiv";
        }

        // Methoden zum Indirekten Zugriff auf private A ttribute*/

        /** Methode zum indirekten Zugriff auf das private Attribute "Einträge" */
        public OSVector getEinträge() {
                return getVektor();
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Einträge" */
        public void setEinträge(OSVector Einträge) {
                this.setVektor(Einträge);
        }

        public String toString() {
                return getObjektName()+" "+getInstanzName();
        }
```

```
}
```

13.1.2 AS_Benutzer

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems
(AS) enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Benutzer ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Benutzer stellt die Entsprechung des Entity-Objektes "Benutzer" dar.
*/

public class AS_Benutzer extends AS_Objekt{

	// Private Variabeln

	private String Benutzername;
	private String Passwort;
	private String Rolle;

	/** Liefert ein String-Objekt, welches einen literalen Namen des Objektes
beinhaltet. */
	public String getObjektName() {
		return "AS_Benutzer";
	}

	// Methoden zum indirekten Zugriff auf private Attribute

	/** Methode zum indirekten Zugriff auf das private Attribute "Benutzername" */
	public String getBenutzername() {
		return Benutzername;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Benutzername" */
	public void setBenutzername(String Benutzername) {
		this.Benutzername=Benutzername;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Passwort" */
	public String getPasswort() {
		return Passwort;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Passwort" */
	public void setPasswort(String Benutzer) {
		this.Passwort=Passwort;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Rolle" */
	public String getRolle() {
		return Rolle;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Rolle" */
	public void setRolle(String Rolle) {
		this.Rolle=Rolle;
	}

	/** Erzeugt eine lesbare Stringausgabe des Objekts */
	public String toString() {
		return this.getObjektName()+" "+Benutzername+" "+Passwort+" "+Rolle;
	}
}
```

13.1.3 AS_Bestellungsdaten

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems
(AS) enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Bestellungsdaten ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Bestellungsdaten stellt die Entsprechung des Entity-Objektes
"Bestellungsdaten" dar.
*/

public class AS_Bestellungsdaten extends AS_Userdaten{

        // Private Variabeln

        private String        Bestelldatum;
        private String        Eingangsdatum;
        private Integer Bestellbetrag;
        private String Artikelname;
        private String        Besteller;

        /** Liefert ein String-Objekt, welches einen literalen Namen des Objektes
beinhaltet. */
        public String getObjektName() {
               return "AS_Bestellungsdaten";
        }

        // Methoden zum indirekten Zugriff auf private Attribute

        /** Methode zum indirekten Zugriff auf das private Attribute "Bestelldatum" */
        public String getBestelldatum() {
               return Bestelldatum;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Bestelldatum" */
        public void setBestelldatum(String Bestelldatum) {
               this.Bestelldatum=Bestelldatum;
        }

        /** Methode zum indirekten Zugriff auf das private Attribute "Eingangsdatum" */
        public String getEingangsdatum() {
               return Eingangsdatum;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Eingangsdatum" */
        public void setEingangsdatum(String Bestellungsdaten) {
               this.Eingangsdatum=Eingangsdatum;
        }

        /** Methode zum indirekten Zugriff auf das private Attribute "Bestellbetrag" */
        public Integer getBestellbetrag() {
               return Bestellbetrag;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Bestellbetrag" */
        public void setBestellbetrag(Integer Bestellbetrag) {
               this.Bestellbetrag=Bestellbetrag;
        }
               /** Methode zum indirekten Zugriff auf das private Attribute
"Artikelname" */
        public String getArtikelname() {
               return Artikelname;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Artikelname" */
        public void setArtikelname(String Artikelname) {
               this.Artikelname=Artikelname;
        }

        /** Methode zum indirekten Zugriff auf das private Attribute "Besteller" */
```

```
        public String getBesteller() {
                return Besteller;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Besteller" */
        public void setBesteller(String Bestellungsdaten) {
                this.Besteller=Besteller;
        }

        /** Erzeugt eine lesbare Stringausgabe des Objekts */
        public String toString() {
                return this.getObjektName()+" "+Bestelldatum+" "+Eingangsdatum+"
"+Bestellbetrag+" "+Artikelname+" "+Besteller;
        }
}
```

13.1.4 AS_Blatt

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems
(AS) enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Blatt ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Blatt stellt die Entsprechung des Entity-Objektes "Blatt" dar.
*/

public class AS_Blatt extends AS_Objekt{

        // Private Variabeln

        private Integer Blattnummer;
        private String Beschreibung;

        /** Liefert ein String-Objekt, welches einen literalen Namen des Objektes
beinhaltet. */
        public String getObjektName() {
                return "AS_Blatt";
        }

        // Methoden zum indirekten Zugriff auf private Attribute

        /** Methode zum indirekten Zugriff auf das private Attribute "Blattnummer" */
        public Integer getBlattnummer() {
                return Blattnummer;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Blattnummer" */
        public void setBlattnummer(Integer Blattnummer) {
                this.Blattnummer=Blattnummer;
        }

        /** Methode zum indirekten Zugriff auf das private Attribute "Beschreibung" */
        public String getBeschreibung() {
                return Beschreibung;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Beschreibung" */
        public void setBeschreibung(String Blatt) {
                this.Beschreibung=Beschreibung;
        }

        /** Erzeugt eine lesbare Stringausgabe des Objekts */
        public String toString() {
                return this.getObjektName()+" "+Blattnummer+" "+Beschreibung;
        }
}
```

13.1.5 AS_Datei

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems
(AS) enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Datei ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Datei stellt die Entsprechung des Entity-Objektes "Datei" dar.
*/

public class AS_Datei extends AS_Objekt{

	// Private Variabeln

	private String Dateiname;
	private String Speicherort;
	private Integer Dateigröße;

	/** Liefert ein String-Objekt, welches einen literalen Namen des Objektes
beinhaltet. */
	public String getObjektName() {
		return "AS_Datei";
	}

	// Methoden zum indirekten Zugriff auf private Attribute

	/** Methode zum indirekten Zugriff auf das private A ttribute "Dateiname" */
	public String getDateiname() {
		return Dateiname;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Dateiname" */
	public void setDateiname(String Dateiname) {
		this.Dateiname=Dateiname;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Speicherort" */
	public String getSpeicherort() {
		return Speicherort;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Speicherort" */
	public void setSpeicherort(String Datei) {
		this.Speicherort=Speicherort;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Dateigröße" */
	public Integer getDateigröße() {
		return Dateigröße;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Dateigröße" */
	public void setDateigröße(Integer Dateigröße) {
		this.Dateigröße=Dateigröße;
	}

	/** Erzeugt eine lesbare Stringausgabe des Objekts */
	public String toString() {
		return this.getObjektName()+" "+Dateiname+" "+Speicherort+" "+Dateigröße;
	}
}
```

13.1.6 AS_Dokument

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems
(AS) enthalten */
```

```
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Dokument ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Dokument stellt die Entsprechung des Entity-Objektes "Dokument" dar.
*/

public class AS_Dokument extends AS_Vektor{

	// Private Variabeln

	/** Liefert ein String-Objekt, welches einen literalen Namen des Objektes 
beinhaltet. */
	public String getObjektName() {
		return "AS_Dokument";
	}

	// Methoden zum indirekten Zugriff auf private Attribute

	/** Methode zum indirekten Zugriff auf das private Attribute "Dateien" */
	public OSVector getDateien() {
		return getVektor();
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Dateien" */
	public void setDateien(OSVector Dateien) {
		this.setVektor(Dateien);
	}

	/** Erzeugt eine lesbare Stringausgabe des Objekts */
	public String toString() {
		return this.getObjektName()+" "+getVektor();
	}
}
```

13.1.7 AS_Dokumententext

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems 
(AS) enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Dokumententext ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Dokumententext stellt die Entsprechung des Entity-Objektes 
"Dokumententext" dar.
*/

public class AS_Dokumententext extends AS_Objekt{

	// Private Variabeln

	private String Inhalt;

	/** Liefert ein String-Objekt, welches einen literalen Namen des Objektes 
beinhaltet. */
	public String getObjektName() {
		return "AS_Dokumententext";
	}

	// Methoden zum indirekten Zugriff auf private Attribute
```

```
        /** Methode zum indirekten Zugriff auf das private Attribute "Inhalt" */
        public String getInhalt() {
                return Inhalt;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Inhalt" */
        public void setInhalt(String Dokumententext) {
                this.Inhalt=Inhalt;
        }

        /** Erzeugt eine lesbare Stringausgabe des Objekts */
        public String toString() {
                return this.getObjektName()+" "+Inhalt;
        }
}
```

13.1.8 AS_Eintrag

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems
(AS) enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Eintrag ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Eintrag stellt die Entsprechung des Entity-Objektes "Eintrag" dar.
*/

public class AS_Eintrag extends AS_Objekt{

        // Private Variabeln

        private AS_Eintragsdaten Eintragsdaten;
        private AS_Dokument Dokument;

        /** Liefert ein String-Objekt, welches einen literalen Namen des Objektes
beinhaltet. */
        public String getObjektName() {
                return "AS_Eintrag";
        }

        // Methoden zum indirekten Zugriff auf private Attribute

        /** Methode zum indirekten Zugriff auf das private Attribute "Dokument" */
        public AS_Dokument getDokument() {
                return Dokument;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Dokument" */
        public void setDokument(AS_Dokument Dokument) {
                this.Dokument=Dokument;
        }

        /** Methode zum indirekten Zugriff auf das private Attribute "Eintragsdaten" */
        public AS_Eintragsdaten getEintragsdaten() {
                return Eintragsdaten;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Eintragsdaten" */
        public void setEintragsdaten(AS_Eintragsdaten Eintragsdaten) {
                this.Eintragsdaten=Eintragsdaten;
        }

        /** Erzeugt eine lesbare Stringausgabe des Objekts */
        public String toString() {
                return this.getObjektName()+" "+Eintragsdaten+" "+Dokument;
        }
}
```

13.1.9 AS_Eintragsdaten

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Eintragsdaten ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Eintragsdaten stellt die Entsprechung des Entity-Objektes "Eintragsdaten" dar.
*/

public class AS_Eintragsdaten extends AS_Objekt{

	// Private Variabeln

	private AS_Systemdaten	Systemdaten;
	private AS_Userdaten		Userdaten;

	/** Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
*/
	public String getObjektName() {
		return "AS_Eintragsdaten";
	}

	// Methoden zum indirekten Zugriff auf private At tribute

	/** Methode zum indirekten Zugriff auf das private Attribute "Systemdaten" */
	public AS_Systemdaten getSystemdaten() {
		return Systemdaten;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Systemdaten" */
	public void setSystemdaten(AS_Systemdaten Systemdaten) {
		this.Systemdaten=Systemdaten;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Userdaten" */
	public AS_Userdaten getUserdaten() {
		return Userdaten;
	}

	/** Methode zum indirekten Setzen des priv aten Attributes "Userdaten" */
	public void setUserdaten(AS_Userdaten Eintragsdaten) {
		this.Userdaten=Userdaten;
	}

	/** Erzeugt eine lesbare Stringausgabe des Objekts */
	public String toString() {
		return this.getObjektName()+" "+Systemdaten+" "+Userdaten;
	}
}
```

13.1.10 AS_Eintragsliste

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des
Eintragslistesystems (AS) enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Eintragsliste ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
```

```
* <br>
* Die Klasse AS_Eintragsliste stellt die Entsprechung des Entity-Objektes "Eintragsliste" dar.
*/

public class AS_Eintragsliste extends AS_Vektor{

	// Private Variabeln

	public AS_Eintragsliste (String InstanzName) {
		this.setInstanzName(InstanzName);
	}

	public AS_Eintragsliste () {
		this("Ohne Name");
	}

	public String getObjektName() {
		return "AS_Eintragsliste";
	}

	// Methoden zum Indirekten Zugriff auf private Attribute*/

	/** Methode zum indirekten Zugriff auf das private Attribute "Einträge" */
	public OSVector getEinträge() {
		return getVektor();
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Einträge" */
	public void setEinträge(OSVector Einträge) {
		this.setVektor(Einträge);
	}

	public String toString() {
		return getObjektName()+" "+getInstanzName();
	}
}
```

13.1.11 AS_Notizzettel

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Notizzettel ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Notizzettel stellt die Entsprechung des Entity-Objektes "Notizzettel" dar.
*/

public class AS_Notizzettel extends AS_Objekt{

	// Private Variabeln

	private String Inhalt;

	/** Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
*/
	public String getObjektName() {
		return "AS_Notizzettel";
	}

	// Methoden zum indirekten Zugriff auf private Attribute

	/** Methode zum indirekten Zugriff auf das private Attribute "Inhalt" */
	public String getInhalt() {
		return Inhalt;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Inhalt" */
	public void setInhalt(String Notizzettel) {
		this.Inhalt=Inhalt;
```

```
	}

	/** Erzeugt eine lesbare Stringausgabe des Objekts */
	public String toString() {
		return this.getObjektName()+" "+Inhalt;
	}
}
```

13.1.12 AS_Nummernkreis

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Nummernkreis ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Nummernkreis stellt die Entsprechung des Entity-Objektes "Datei" dar.
*/

public class AS_Nummernkreis extends AS_Objekt{

	// Private Variabeln

	private Integer		Startnummer;
	private Integer		Endnummer;
	private String		Name;

	/** Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
*/
	public String getObjektName() {
		return "AS_Nummernkreis";
	}

	// Methoden zum indirekten Zugriff auf private Attribute

	/** Methode zum indirekten Zugriff auf das private Attribute "Startnummer" */
	public Integer getStartnummer() {
		return Startnummer;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Startnummer" */
	public void setStartnummer(Integer Startnummer) {
		this.Startnummer=Startnummer;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Endnummer" */
	public Integer getEndnummer() {
		return Endnummer;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Endnummer" */
	public void setEndnummer(Integer Endnummer) {
		this.Endnummer=Endnummer;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Name" */
	public String getName() {
		return Name;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Name" */
	public void setName(String Name) {
		this.Name=Name;
	}

	/** Erzeugt eine lesbare Stringausgabe des Objekts */
	public String toString() {
		return this.getObjektName()+" "+Startnummer+" "+Endnummer+" "+Name;
	}
}
```

13.1.13 AS_Objekt

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Objekt ist eine "persistence-capable"-Klasse.<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* Sie ist "abstract" und kann nicht instantiiert werden.<br>
* <br>
* Die Klasse AS_Objekt ist die Klasse, von der alle anderen AS_ Klassen erben.
*/

public abstract class AS_Objekt {

        // Database fields
        private String InstanzName;

        /** Liefert einen String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
*/
        public String getObjektName() {
                return "AS_Objekt";
        }

        // Methoden zum indirekten Zugriff auf private Attribute

        /** Erzeugt eine lesbare Stringausgabe des Objekts */
        public String toString() {
                return this.getInstanzName();
        }

        /** Methode zum indirekten Zugriff auf das private Attribute "InstanzName" */
        public String getInstanzName() {
                return InstanzName;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "InstanzName" */
        public void setInstanzName(String InstanzName) {
                this.InstanzName=InstanzName;
        }

        /** Diese Methode ist für die Datenbank nötig , wird aber in dieser Implementation nicht
weiter verwendet. */
        public int hashCode() {
                return super.hashCode();
        }
}
```

13.1.14 AS_Rechnungsdaten

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Datei stellt die Entsprechung des Entity-Objektes "Rechnungsdaten" dar.
*/

public class AS_Rechnungsdaten extends AS_Userdaten{

        // Private Variabeln
```

```
	private String 		Rechnungsdatum;
	private String 		Eingangsdatum;
	private Integer Rechnungsbetrag;
	private Integer Kostenstelle;
	private String 		Rechnungssteller;

	/** Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
*/
	public String getObjektName() {
		return "AS_Rechnungsdaten";
	}

	// Methoden zum indirekten Zugriff auf private Attribute

	/** Methode zum indirekten Zugriff auf das private Attribute "Rechnungsdatum" */
	public String getRechnungsdatum() {
		return Rechnungsdatum;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Rechnungsdatum" */
	public void setRechnungsdatum(String Rechnungsdatum) {
		this.Rechnungsdatum=Rechnungsdatum;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Eingangsdatu m" */
	public String getEingangsdatum() {
		return Eingangsdatum;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Eingangsdatum" */
	public void setEingangsdatum(String Rechnungsdaten) {
		this.Eingangsdatum=Eingangsdatum;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Rechnungsbetrag" */
	public Integer getRechnungsbetrag() {
		return Rechnungsbetrag;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Rechnungsbetrag" */
	public void setRechnungsbetrag(In teger Rechnungsbetrag) {
		this.Rechnungsbetrag=Rechnungsbetrag;
	}
		/** Methode zum indirekten Zugriff auf das private Attribute "Kostenstelle" */
	public Integer getKostenstelle() {
		return Kostenstelle;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Kostenstelle" */
	public void setKostenstelle(Integer Kostenstelle) {
		this.Kostenstelle=Kostenstelle;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Rechnungssteller" */
	public String getRechnungssteller() {
		return Rechnungssteller;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Rechnungssteller" */
	public void setRechnungssteller(String Rechnungsdaten) {
		this.Rechnungssteller=Rechnungssteller;
	}

	/** Erzeugt eine lesbare Stringausgabe des Objekts */
	public String toString() {
		return this.getObjektName()+" "+Rechnungsdatum+" "+Eingangsdatum+"
"+Rechnungsbetrag+" "+Kostenstelle+" "+Rechnungssteller;
	}
}
```

13.1.15 AS_Systemdaten

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
```

```
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Systemdaten ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Systemdaten stellt die Entsprechung des Entity-Objektes "Systemdaten" dar.
*/

public class AS_Systemdaten extends AS_Objekt{

	// Private Variabeln

	private String		Datum;
	private String		Benutzername;

	/** Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
*/
	public String getObjektName() {
		return "AS_Systemdaten";
	}

	// Methoden zum indirekten Zugriff auf private Attribute

	/** Methode zum indirekten Zugriff auf das private Attribute "Datum" */
	public String getDatum() {
		return Datum;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Datum" */
	public void setDatum(String Datum) {
		this.Datum=Datum;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Benutzername" */
	public String getBenutzername() {
		return Benutzername;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Benutzername" */
	public void setBenutzername(String Systemdaten) {
		this.Benutzername=Benutzername;
	}

	/** Erzeugt eine lesbare Stringausgabe des Objekts */
	public String toString() {
		return this.getObjektName()+" "+Datum+" "+Benutzername;
	}
}
```

13.1.16 AS_Userdaten

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Userdaten ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* Im System ist sie allerdings abstract und kann dahe nicht instantiiert werden<br>
* <br>
* Die Klasse AS_Userdaten stellt die Entsprechung des Entity-Objektes "Userdaten" dar.
*/

public abstract class AS_Userdaten extends AS_Objekt{

	// Private Variabeln

	/** Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
*/
	public String getObjektName() {
		return "AS_Userdaten";
```

```
        }

        // Methoden zum indirekten Zugriff auf private Attribute

        /** Erzeugt eine lesbare Stringausgabe des Objekts */
        public String toString() {
                return this.getObjektName();
        }
}
```

13.1.17 AS_Vektor

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Vektor ist eine "persistence-capable"-Klasse.<br>
* Sie ist "abstract" und kann keine diskrete Instanz in der Datenbank sein.<br>
* <br>
* Die Klasse AS_Vektor ist die Klasse, von der alle anderen AS_ Klassen erben,
* die einen Vektor verwenden.
*/

public abstract class AS_Vektor extends AS_Objekt{

        // Database fields
        private OSVector Vektor=new OSVector();

        /** Liefert einen String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
*/
        public String getObjektName() {
                return "AS_Vektor";
        }

        // Methoden zum indirekten Zugriff auf private Attribute

        /** Methode zum indirekten Zugriff auf das private Attribute "Vektor" */

        public OSVector getVektor() {
                return Vektor;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Vektor" */

        public void setVektor(OSVector Vektor) {
                this.Vektor=Vektor;
        }

        /** Erzeugt eine lesbare Stringausgabe des Objekts */

        public String toString() {
                return this.getObjektName();
        }
}
```

13.1.18 AS_Version

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Version ist eine "persistence-capable"-Klasse,<br>
```

```
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Version stellt die Entsprechung des Entity-Objektes "Version" dar.
*/

public class AS_Version extends AS_Objekt{

        // Private Variabeln

        private Integer Versionnummer;
        private String Beschreibung;

        /** Liefert ein String-Objekt, welches einen  literalen Namen des Objektes beinhaltet.
*/
        public String getObjektName() {
                return "AS_Version";
        }

        // Methoden zum indirekten Zugriff auf private Attribute

        /** Methode zum indirekten Zugriff auf das private Attribute "Versionnummer" */
        public Integer getVersionnummer() {
                return Versionnummer;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Versionnummer" */
        public void setVersionnummer(Integer Versionnummer) {
                this.Versionnummer=Versionnummer;
        }

        /** Methode zum indirekten Zugriff auf das private Attribute "Beschreibung" */
        public String getBeschreibung() {
                return Beschreibung;
        }

        /** Methode zum indirekten Setzen des privaten Attributes "Beschreibung" */
        public void setBeschreibung(String Version) {
                this.Beschreibung=Beschreibung;
        }

        /** Erzeugt eine lesbare Stringausgabe des Objekts */
        public String toString() {
                return this.getObjektName()+" "+Versionnummer+" "+Beschreibung;
        }
}
```

13.1.19 AS_Zeichnungsdaten

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;

/**
* Die Klasse AS_Zeichnungsdaten ist eine "persistence-capable"-Klasse,<br>
* d.h. sie kann in der Persistenzdatenbank gespeichert werden.<br>
* <br>
* Die Klasse AS_Zeichnungdaten stellt die Entsprechung des Entity-Objektes "Zeichnungsdaten"
dar.
*/

public class AS_Zeichnungsdaten extends AS_Userdaten{

        // Private Variabeln

        private Integer Zeichnungsnummer;
        private String         Beschreibung;
        private boolean        Freigabe;

        /** Liefert ein String-Objekt, welches einen literalen Namen des Objektes beinhaltet.
*/
        public String getObjektName() {
                return "AS_Zeichnungsdaten";
        }
```

```
	// Methoden zum indirekten Zugriff auf private Attribute

	/** Methode zum indirekten Zugriff auf das private Attribute "Beschreibung" */
	public String getBeschreibung() {
		return Beschreibung;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Beschreibung" */
	public void setBeschreibung(String Beschreibung) {
		this.Beschreibung=Beschreibung;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Zeichnungsnummer" */
	public Integer getZeichnungsnummer() {
		return Zeichnungsnummer;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Zeichnungsnummer" */
	public void setZeichnungsnummer(Integer Zeichnungsnummer) {
		this.Zeichnungsnummer=Zeichnungsnummer;
	}

	/** Methode zum indirekten Zugriff auf das private Attribute "Freigabe" */
	public boolean getFreigabe() {
		return Freigabe;
	}

	/** Methode zum indirekten Setzen des privaten Attributes "Freigabe" */
	public void setFreigabe(boolean Freigabe) {
		this.Freigabe=Freigabe;
	}

	/** Erzeugt eine lesbare Stringausgabe des Objekts */
	public String toString() {
		return this.getObjektName()+Zeichnungsnummer+" "+Beschreibung+" "+Freigabe;
	}
}
```

13.1.20 Client

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;
import java.awt.*;

/**
* Die Klasse Client ist das Hauptprogramm auf Clientseite.
* Es stellt die Verbindung zum Server her und
* instantiiert die Schnittstellenobjekte
*/

public class Client{

	/** Die Funktion main wird beim Start von der Komnmandozeile aufgerufen */
	public static void main(String argv[]) {
		Frame f=new Frame("Archiv");
		Interface AS=null;

		System.setSecurityManager(new RMISecurityManager());
		if (argv.length != 1)
		{
			System.out.println("usage: java Client <IP address of host running RMI
server>");
			System.exit(0);
		}
		String serverName = argv[0];
		try
		{
			//bind server object to object in client
```

```
                        AS = (Interface) Naming.lookup("rmi://"+s erverName+"/ASServer");

                        //invoke method on server object
                        String s = AS.getHello();
                        System.out.println(s);
                }
                catch(Exception e)
                {
                        System.out.println("Exception occured: " + e);
                        System.exit(0);
                }
                System.out.println("RMI connection successful");

                PasswortDialog pd=new PasswortDialog(f,"Login");
                pd.show();
                System.out.println("Ergebnis: "+pd.getBenutzername() +" " +pd.getPasswort());
                if (pd.getOK()) {
                        try
                        {
                                //invoke method on server object
                                String Rolle =
AS.checkLogin(pd.getBenutzername(),pd.getPasswort());
                                System.out.println(pd.getBenutzername()+" ist "+Rolle);
                        }
                        catch(Exception e)
                        {
                                System.out.println("Exception occured: " + e);
                                System.exit(0);
                        }

                        Hauptfenster H= new Hauptfenster(AS);
                        H.show();
                        while (1==1) {}
                }
                pd.dispose();
        }
}
```

13.1.21 Funktionsleiste

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

import java.util.*;
import java.awt.*;
import java.awt.event.*;

/**
* Die Klasse Funktionsleiste ist die Entsprechung des Schnittstellenobjektes "Funktionsleiste"
* Es empfängt die Benutzerkommandos und führt die Aktionen aus
*/

public class Funktionsleiste extends MenuBar{
        Frame parent;

        /** Erzeugt die Funktionsleiste */

        public Funktionsleiste(Frame parent){
                this.parent=parent;

                Menu DateiMenu = new Menu("Datei");
                MenuItem ProgEnde = new MenuItem("Programm beenden");
                DateiMenu.add(ProgEnde);
                ProgEnde.addActionListener(new FunktionsleisteListener(parent));

                Menu BenutzerMenu = new Menu("Benutzer");
                MenuItem BenutzerNeu = new MenuItem("Neuen Benutzer anlegen");
                BenutzerMenu.add(BenutzerNeu);
                BenutzerNeu.addActionListener(new FunktionsleisteListener(parent));

                Menu DokumentMenu = new Menu("Dokument");
                MenuItem Ansehen = new MenuItem("Betrachten");
                DokumentMenu.add(Ansehen);
                Ansehen.addActionListener(new FunktionsleisteListener(parent));
                MenuItem Erfassen= new MenuItem("Erfassen");
                DokumentMenu.add(Erfassen);
                Erfassen.addActionListener(new FunktionsleisteListener(parent));
```

```
		MenuItem Reproduzieren= new MenuItem("Reproduzieren");
		DokumentMenu.add(Reproduzieren);
		Reproduzieren.addActionListener(new FunktionsleisteListener(parent));

		Menu EintragMenu = new Menu("Eintrag");
		MenuItem Suchen= new MenuItem("Suchen");
		EintragMenu.add(Suchen);
		Suchen.addActionListener(new FunktionsleisteListener(parent));
		MenuItem Ändern = new MenuItem("Ändern");
		EintragMenu.add(Ändern);
		Ändern.addActionListener(new FunktionsleisteListener(parent));
		MenuItem Löschen= new MenuItem("Löschen");
		EintragMenu.add(Löschen);
		Löschen.addActionListener(new FunktionsleisteListener(parent));

		Menu ZeichnungMenu = new Menu("Zeichnung");
		MenuItem Bearbeiten = new MenuItem("Bearbeiten");
		ZeichnungMenu.add(Bearbeiten);
		Bearbeiten.addActionListener(new FunktionsleisteListener(parent));
		MenuItem Weiterleitung= new MenuItem("Weiterleitung");
		ZeichnungMenu.add(Weiterleitung);
		Weiterleitung.addActionListener(new FunktionsleisteListener(parent));
		MenuItem Freigabe= new MenuItem("Freigabe");
		ZeichnungMenu.add(Freigabe);
		Freigabe.addActionListener(new FunktionsleisteListener(parent));

		add(DateiMenu);
		add(BenutzerMenu);
		add(DokumentMenu);
		add(EintragMenu);
		add(ZeichnungMenu);
	}

	class FunktionsleisteListener implements ActionListener {
		Frame Fenster;
		FunktionsleisteListener(Frame Fenster){this.Fenster=Fenster;}

		public void actionPerformed(ActionEvent e){
			System.out.println(e);
			if (e.getActionCommand()=="Neuen Benutzer anlegen"){
				NeuerBenutzerDialog nbd=new NeuerBenutzerDialog(Fenster,"Neuen
Benutzer anlegen");
				nbd.show();
				System.out.println("Neuer Benutzer: "+nbd.getBenutzername()+"
als "+nbd.getRolle());
				nbd.dispose();
			}

			if (e.getActionCommand()=="Programm beenden"){
				Fenster.dispose();
				System.exit(0);
			}

			if (e.getActionCommand()=="Datei öffnen"){
				PasswortDialog pd=new PasswortDialog(Fenster,"Login");
				pd.show();
				System.out.println("Ergebnis: "+pd.getBenutzername());
				pd.dispose();
			}
		}
	}
}
```

13.1.22 Hauptfenster

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;

/**
* Die Klasse Hauptfenster ist die Entsprechung des Schnittstellenobjektes "Hauptfenster".
```

```
* Es enthält die Funktionsleiste und die Eintragsliste.
*/

public class Hauptfenster extends Frame{

	/** Erzeugt das Hauptfenster */

	public Hauptfenster(Interface AS){
		int i;
		Funktionsleiste Menuleiste=new Funktionsleiste(this);

		setMenuBar(Menuleiste);
		List Eintrag=new List(20);
		Vector V=new Vector();

		try
		{
			//invoke method on server object
			V=AS.getEinträge();
		}
		catch(Exception e)
		{
			System.out.println("Exception occured: " + e);
			System.exit(0);
		}
		for (i=0;i<V.size();i++)
			Eintrag.add(V.elementAt(i).toString());

		Eintrag.setSize(getPreferredSize());
		add(Eintrag);
		addWindowListener(new HauptfensterListener(this));

		setSize(getPreferredSize());
		pack();
		repaint();
	} // End public Hauptfenster

	class HauptfensterListener extends WindowAdapter {
		Frame Fenster;
		HauptfensterListener(Frame Fenster){this.Fenster=Fenster;}

		public void windowClosed(WindowEvent e){
			System.out.println(e);
			Fenster.dispose();
			System.exit(0);
		}
		public void windowClosing(WindowEvent e){
			System.out.println(e);
			Fenster.dispose();
			System.exit(0);
		}
	}

	class dol implements ActionListener {
		Frame Fenster;
		dol(Frame Fenster){this.Fenster=Fenster;}

		public void actionPerformed(ActionEvent e){
			System.out.println(e);
			if (e.getActionCommand()=="Neuen Benutzer anlegen"){
				NeuerBenutzerDialog nbd=new NeuerBenutzerDialog(Fenster,"Neuen
Benutzer anlegen");
				nbd.show();
				System.out.println("Ergebnis: "+nbd.g etBenutzername());
				nbd.dispose();
			}

			if (e.getActionCommand()=="Programm beenden"){
				dispose();
			}

			if (e.getActionCommand()=="Datei öffnen"){
				PasswortDialog pd=new PasswortDialog(Fenster,"Login");
				pd.show();
				System.out.println("Ergebnis: "+pd.getBenutzername());
				pd.dispose();
			}
		}
	}
}
```

13.1.23 NeuerBenutzerDialog

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

import java.util.*;
import java.awt.*;
import java.awt.event.*;

/**
* Die Klasse NeuerBenutzerDialog ist die Entsprechung des Schnittstellenobjektes
"NeuerBenutzerDialog".
* Mit Hilfe dieses Dialogs wird ein neuer Benutzer angelegt.
*/

public class NeuerBenutzerDialog extends Dialog{

        private TextField t1 = new TextField(12);
        private TextField t2 = new TextField(12);
        private Choice Rollen=new Choice();
        private Button loginButton = new Button("Anlegen");
        private Button cancelButton = new Button("Abbruch");
        private String userName;
        private String password;
        private String rolle;
        private boolean OK;

        /** Gibt den Benutzernamen zurück */
        public String getBenutzername(){
                return userName;
        }

        /** Gibt das voreingestellte Passwort zurück */
        public String getPasswort(){
                return password;
        }

        /** Gibt die zugewiesene Rolle zurück */
        public String getRolle(){
                return rolle;
        }

        /** Zeigt an, ob der Dialog abgebrochen wurde */
        public boolean getOK(){
                return OK;
        }

        /** Erzeugt den NeuerBenutzerDialog */
        public NeuerBenutzerDialog( Frame dw, String title )
        {
                super(dw,title,true);
                addNotify();
                GridBagLayout gb = new GridBagLayout();
                GridBagConstraints c = new GridBagConstraints();
                setLayout(gb);

                c.gridwidth = GridBagConstraints.REMAINDER;
                Label text1 = new Label ("Neuen Benutzer anlegen");
                gb.setConstraints(text1,c);
                add(text1);

                Label text2 = new Label ("Bitte eingeben");
                gb.setConstraints(text2,c);
                add(text2);

                Label l1 = new Label("Benutzername");
                add(l1);

                c.gridwidth = GridBagConstraints.REMAINDER;
                gb.setConstraints(t1,c);
                add(t1);

                Label l2 = new Label("Passwort");
                add(l2);

                c.gridwidth = GridBagConstraints.REMAINDER;
                gb.setConstraints(t2,c);
                t2.setEchoChar('*');
                add(t2);
```

```
            Label l3 = new Label("Rolle");
            add(l3);

            Rollen.add("Archivar");
            Rollen.add("Einkäufer");
            Rollen.add("Verkäufer");
            Rollen.add("Projektleiter");
            Rollen.add("Büromitarbeiter");
            Rollen.add("Qualitätssicherungsmitarbeiter") ;
            Rollen.add("Konstrukeur");
            Rollen.add("Buchhalter");
            c.gridwidth = GridBagConstraints.REMAINDER;
            gb.setConstraints(Rollen,c);

            add(Rollen);
            add(loginButton);
            add(cancelButton);
            pack();
    }

    public boolean action ( Event e, Object o )  {
            if ( e.target == loginButton ) {
                    System.out.println("NeuerBenutzerDialog: Login: pressed");
                    userName = t1.getText();
                    password = t2.getText();
                    rolle=Rollen.getSelectedItem();
                    OK=true;
                    dispose();
            }
            if ( e.target == cancelButton )      {
                    System.out.println("NeuerBenutzerDialog: Cancel: pressed");
                    userName = "";
                    password = "";
                    rolle = "";
                    OK=false;
                    dispose();
            }
            return true;
    }
}
```

13.1.24 PasswortDialog

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */
import COM.odi.util.*;
import COM.odi.*;
import java.util.*;
import java.awt.*;

/**
* Die Klasse PasswortDialog stellt die Entsprechung des
* Schnittstellen-Objektes "PasswortDialog" dar.
*/

public class PasswortDialog extends Dialog{

    private TextField t1 = new TextField(12);
    private TextField t2 = new TextField(12);
    private Button loginButton = new Button("OK");
    private Button cancelButton = new Button("Abbruch");
    private String userName;
    private String password;
    private boolean OK;

    // Methoden zum indirekten Zugriff auf private Attribute

    /** getBenutzername liefert den eingeg ebenen Benutzernamen */

    public String getBenutzername(){
            return userName;
    }

    /** getPasswort liefert das eingegebene Passwort */
```

```
        public String getPasswort(){
                return password;
        }

        /** getOK gibt an, ob der Dialog abgebrochen wurde */

        public boolean getOK(){
                return OK;
        }

        public PasswortDialog( Frame dw, String title )
        {
                super(dw,title,true);
                addNotify();
                GridBagLayout gb = new GridBagLayout();
                GridBagConstraints c = new GridBagConstraints();
                setLayout(gb);

                Label text2 = new Label ("Bitte anmelden");
                gb.setConstraints(text2,c);
                add(text2);

                Label l1 = new Label("Benutzername");
                add(l1);

                c.gridwidth = GridBagConstraints.REMAINDER;
                gb.setConstraints(t1,c);
                add(t1);

                Label l2 = new Label("Passwort");
                add(l2);

                c.gridwidth = GridBagConstraints.REMAINDER;
                gb.setConstraints(t2,c);
                t2.setEchoChar('*');
                add(t2);

                add(loginButton);
                add(cancelButton);
                pack();
        }

        public boolean action ( Event e, Object o )  {
                if ( e.target == loginButton ) {
                        System.out.println("PasswortDialog: Login: pressed");
                        userName = t1.getText();
                        password = t2.getText();
                        OK=true;
                        dispose();
                }
                if ( e.target == cancelButton ) {
                        System.out.println("PasswortDialog: Cancel: pressed");
                        userName = "";
                        password = "";
                        OK=false;
                        dispose();
                }

                return true;
        }
}
```

13.2 Programmquelltext Server

13.2.1 Impl

```
package COM.odi.demo.archiv;
import COM.odi.*;
import COM.odi.util.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.*;

/** Die Klasse Impl implementiert die Methoden auf der Serverseite,
* die dann von den Clientprogrammen aufgerufen werden können */

public class Impl extends UnicastRemoteObject implements Interface
```

```
{
        private Database db;
        private Transaction tr;

        /** Impl führt das Naming.rebind aus und initialisiert die Datenbank */
        public Impl(String name, String dbName) throws RemoteException
        {
                super();
                try
                {
                        Naming.rebind(name, this);
                }
                catch(Exception e)
                {
                        System.out.println("Exception occurred: " + e);
                }

                System.out.println("Archivdatei: "+dbName);

                // Die folgende Zeile initialisiert die ObjectStore Java software.
                ObjectStore.initialize(null, null);

                try {
                        Database.open(dbName, Database.openUpdate).destroy();
                } catch (DatabaseNotFoundException e) {
                }

                // Call the Database.create() method to create and open the
                // database that is specified on the command line
                // when the application is invoked:

                try {
                        db = Database.open(dbName, Database.openUpdate);
                        db.destroy();
                } catch (DatabaseNotFoundException e) {
                }

                db = Database.create(dbName, Database.allRead | Database.allWrite);
        }

        /** Gibt einen String zur Begrüssung des Clients zurück. Damit wird
        die Kommunikation getestet */
        public String getHello(){
                return "Willkommen zum Archivsystem";
        }

        /** Überprüft das Passwort und gibt eine gültige Rolle zurück */
        public String checkLogin(String Benutzername, String Passwort){
                return "Konstrukteur";
        }

        /** gibt die Einträge zur Darstellung im Hauptfenster zurück */
        public Vector getEinträge(){
                Vector V=new Vector();
                for (int i=1;i<120;i++)
                        V.addElement("Eintrag "+i);
                return V;
        }
}
```

13.2.2 Interface

```
/** Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

import java.util.*;

/** Das Interface definiert die Methoden,
* die von Clients auf dem Server aufgerufen werden können.
* Aus dieser Interfacedefinition erzeugt ein Interface-Compiler (rmic) Stubs. */

public interface Interface extends java.rmi.Remote
{
        public String getHello() throws java.rmi.RemoteException;
        public String checkLogin(String Benutzername, String Passwort) throws
java.rmi.RemoteException;
        public Vector getEinträge() throws java.rmi.RemoteException;
}
```

13.2.3 Server

```
/* Copyright (C) Stephan Winking, 1997 */

/* Im Package COM.odi.demo.archiv sind alle Module zur Implementierung des Archivsystems (AS)
enthalten */
package COM.odi.demo.archiv;

/** Das COM.odi.util package enthält die Klasse OSVector  */

import COM.odi.*;
import COM.odi.util.*;
import java.util.*;
import java.awt.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

/**
* Die Klasse Server ist das Hauptprogramm auf Serverseite.
* Es instantiiert das Interface, über das die Clients
* auf die Servermethode zugreifen.
*/

public class Server{

	/** Die Funktion main wird beim Start von der Komnmandozeile aufgerufen */

	public static void main(String argv[]) {

		String dbName = argv[0];

		System.setSecurityManager(new RMISecurityManager());
		try {
			Impl implementation = new Impl("ASServer",dbName);
			System.out.println("ASServer bound");
		}
		catch (Exception e){
			System.out.println("Exception occurred: " + e);
		}

		Frame f=new Frame("Archiv");
		while (1==1);
	}
}
```

14 Literatur

[Booch95] Booch G. *Objektorientiert Analyse und Design*, Addison-Wesley, Bonn, 1. korr. Nachdruck, 1995

[BoRu95] Booch G., Rumbaugh J. *Unified Method for Object-Oriented Development*, Version 0.8, Rational Software Corporation, USA, 1995

[Brock68] *Brockhaus Enzyklopädie*, 4. Band, 17. Aufl., F.A. Brockhaus, Wiesbaden, 1968

[Brock78] *Der Große Brockhaus*, 3. Band, 18. Aufl., F.A. Brockhaus, Wiesbaden, 1978

[Bull93] Bull (Veranstalter). *Die IMAGEWorks Produktfamilie*, Folienvortrag, Bull Information Systems, Langen, 1993.

[BuMa94] Bullinger, Mager. *Integriertes Dokumenten-Management*, Fraunhofer-Gesellschaft, Institut für Arbeit und Organisation, FBO-Verlag, Baden-Baden, 1994

[Catt93] Catell, R.G.G. *The Object Database Standard: ODMG 93*, Morgan Kaufmann Publishers, USA, 1993

[Deb97] Debatin, F. *Java und Datenbanken - Der ODMG-Standard zur objektorientierten Datenbankanbindung*, Java Spektrum 2/97: S.25ff.

[DudI88] *Duden Informatik*, 1. Aufl., Dudenverlag, Mannheim, 1988

[Flan96] Flanagan, D. *Java in a nutshell*, O´Reilly & Ass., Bonn, 1996

[Grei88] Greif, I. (ed.). *Computer Supported Cooperative Work: A book of readings*, Morgan Kaufman Publ., 1988

[GSZ93] Gulbins J., Seyfried M., Zimmermann H. *Elektronische Archivsysteme*, Springer-Verlag, Heidelberg, 1993

[Jabl95] Jablonski, S. *Workflow-Management-System: Modellierung und Architektur*, Thomson Publishing, 1995

[Jabl95b] Jablonski, S. *Workflow-Management-Systeme: Motivation, Modellierung, Architektur*, Informatik Spektrum 18: 13-24, Springer-Verlag, 1995

[Jaco95] Jacobson, I. *Formalizing use-case modeling*, Journal of Object-Oriented Programming 6:10-14, 1995.

[JCJÖ92] Jacobson I., Christerson M., Jonsson P., Övergaard G. *Object Oriented Software Engineering*, Addison-Wesley, 1992.

[Lit95] Litke, Prof. Dr. H. *Workflow: Gute Planung, hoher Nutzen*, Business Computing 7/95: S. 24ff.

[Mack85] Mackensen, L. *Etymologisches Wörterbuch der deutschen Sprache*, VMA- Verlag, Wiesbaden, 1985

[Meyer73] *Meyers Enzyklopädisches Lexikon*, 7. Band, 9. Aufl, Bibliographisches Institut, Mannheim, 1973

[MüSc96] Müsken V., Schult Dr. T. J. *Vom Problem zum objektorientierten Programm*, c´t 4/96: S.330ff., Heise Verlag, Hannover

[NiPe96] Niemeyer P., Peck, J. *Exploring Java*, O´Reilly & Ass., Bonn, 1996

[ODI97] *Technischer Überblick Objectstore für Java*, Object Design Software GmbH, Wiesbaden, 1997

[PG231] Projektgruppe 231, *Vorgangsverwaltung in Office-Systemen*, Endbericht, Fachbereich Informatik, Universität Dortmund, Mai 1994

[PG96g] Projektgruppe 263, *Prometheus-Spezifikation: Objectory-Dokumente,* Interner Bericht, Fachbereich Informatik, Universität Dortmund, Mai 1996

[Rumb91] Rumbaugh J. et al. *Object Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, USA, 1991

[Stor95] Storp Dr. H. *Leitstand im Büro*, Business Computing 7/95: S. 32ff.

[Sun97] Sun Microsystems Inc. *Java Development Kit 1.1 Documentation*, Sun Microsystems Inc, Mountain View, USA, 1997

[Teu95] Teufel S. *Computerunterstützung für die Gruppenarbeit*, Addison-Wesley, 1995

[Waln88] Walne P. *Dictionary of archival terminology*, Saur Verlag, München, 1988

[WfMC94] Workflow Management Coalition. *A Workflow Management Coalition Specification*, Brüssel, Belgien, 1994