Diplomarbeit

Objektorientierte Systementwicklung und Prototyping mit persistenten Daten

Axel Schlösser

Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

30. Januar 1997

Gutachter:

Prof. Dr. E.-E. Doberkat
Dr. S. Dißmann

Schriftliche Ausarbeitung zur Diplomarbeit

Objektorientierte Systementwicklung und Prototyping mit persistenten Daten

Zusammenfassung. In dieser Diplomarbeit wird die Konstruktion von Prototypen auf der Basis von Modellen, die im Rahmen eines objektorientierten Entwicklungsprozesses erstellt werden, anhand dreier Fallbeispiele untersucht. Diese Fallbeispiele spiegeln unterschiedliche Eigenheiten von Datenbankanwendungen wider. Für die Implementierung der Prototypen wird die Prototyping-Sprache ProSet verwendet. Die Prototypen nutzen den ProSet-Persistenzmechanismus. Bei den Fallbeispielen werden unterschiedliche Arten von Prototypen implementiert. Die Art des implementierten Prototyps ist abhängig von dem Ziel, das mit dem Prototyping verfolgt wird. Bei dem betrachteten Entwicklungsprozeß handelt es sich um den Objectory-Entwicklungsprozeß. Ergänzend zu Objectory werden Teile des WAM-Ansatzes betrachtet.

Schlüsselwörter: Objectory, Objektorientierte Software-Entwicklung, OOSE, Persistenz, ProSet, Prototyping, WAM

Summary. In this master's thesis the construction of prototypes based on models that are developed in an object-oriented development process is investigated for three different case studies. The case studies highlight different aspects of database applications. The prototyping language ProSet is used for the implementation of prototypes. The prototypes use the persistence mechanism of ProSet. The consideration of the development of prototypes is related to different goals of prototyping. The object-oriented development process is based on Objectory. In addition to Objectory parts of the WAM approach are used.

Key words: Object-oriented software development, Objectory, OOSE, Persistence, ProSet, Prototyping, WAM

Danksagungen. Insbesondere bedanke ich mich bei meinem Betreuer W. Franke für die Unterstützung bei der Erstellung dieser Diplomarbeit.

Des weiteren sei Dr. S. Dißmann, B. Jelinek, Dr. C. Pahl und K. Waltenberg gedankt, die mir durch Anregungen und Verbesserungsvorschläge geholfen oder mich bei der Einarbeitung in Software-Systeme oder Anwendungsbereiche unterstützt haben.

Für die Übernahme der Gutachten bedanke ich mich bei Prof. Dr. E.-E. Doberkat und Dr. S. Dißmann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

In diesem Kapitel werden zunächst im Abschnitt 1.1 die Motivation und der Kontext dieser Diplomarbeit vorgestellt. Anschließend wird im Abschnitt 1.2 die Aufgabenstellung der Arbeit beschrieben. Abschnitt 1.3 gibt einen kurzen allgemeinen Überblick über diese Arbeit. Im Abschnitt 1.4 werden die Rahmenbedingungen, die für diese Diplomarbeit wichtig sind, aufgeführt.

1.1 Motivation und Kontext

Bei der Entwicklung von Software-Systemen haben sich in den letzten Jahren sowohl die objektorientierte Systementwicklung als auch das Software-*Prototyping* als Alternativen oder Ergänzungen zu traditionellen Formen der Software-Entwicklung etabliert. Diese Ansätze sind bisher jedoch noch nicht auf die Eigenheiten von Datenbankanwendungen hin angepaßt worden. Der Begriff Prototyping wird in dieser Arbeit gemäß der Beschreibung von Floyd [Flo84] verwendet. Demnach bezieht sich das Prototyping auf eine wohldefinierte Phase im Produktionsprozeß von Software, in der ein Modell erzeugt wird, das bereits alle wesentlichen Eigenschaften des zu entwickelnden Software-Systems besitzt. Anhand des Modells werden diese Eigenschaften getestet und der weitere Software-Produktionsprozeß bestimmt. Ein ausführbares Modell, das im Rahmen des Prototyping erstellt wird, wird im folgenden als *Prototyp* bezeichnet.

In der vorliegenden Diplomarbeit wird die Verbindung der objektorientierten Vorgehensweise bei der Systementwicklung im Zusammenhang mit dem Prototyping untersucht, wobei die Entwicklung von Anwendungen mit persistenten Daten im Mittelpunkt steht.

Das in dieser Diplomarbeit betrachtete Prototyping bezieht sich speziell auf das Prototyping mit der mengenorientierten Prototyping-Sprache ProSet [DFG+92, DFKS93, DFH+95]. ProSet ermöglicht die Modellierung von Programmen und Daten. Hierzu stellt ProSet neben den üblichen primitiven Datentypen, wie z.B. Zahlen oder Zeichenketten, noch Mengen, Tupel, Funktionen, Moduln und Modulinstanzen als grundlegende Datentypen zur Verfügung. Des weiteren bietet ProSet die Möglichkeit, Daten persistent zu machen, d.h. Daten oder Programmeinheiten über eine Programmausführung hinweg zu erhalten und sie anderen Programmen zugänglich zu machen.

Ausgangspunkt für die objektorientierte Systementwicklung ist die Software-Entwicklungsmethodik *Objectory* [Ory95], die auf dem *OOSE*-Konzept nach Jacobson et al. [JCJÖ93] basiert. Die Prototypen, die im Rahmen der Diplomarbeit erstellt werden, sollen den Objectory-Entwicklungsprozeß ergänzen.

Prototyping kann in verschiedenen Phasen eines Software-Entwicklungsprozesses eingesetzt werden. Umfassende Darstellungen hierzu geben Kieback et al. [KLSZ92] und Budde et al. [BKKZ92]. Abhängig davon, welche Aktivitäten im Software-Entwicklungsprozeß durch das Prototyipng unterstützt werden sollen, schlagen Kieback et al. beispielsweise eine Unterscheidung zwischen Demonstrationsprototypen, Prototypen im engeren Sinne, Labormustern und Pilotsystemen vor.

In dieser Diplomarbeit wird das Prototyping in verschiedenen Phasen des Software-Entwicklungsprozesses betrachtet. Dabei werden Konzepte des bei Gryczan und Züllighoven [GZ92] sowie bei Bäumer et al. [BGZ95] vorgestellten *WAM(Werkzeug, Aspekt, Material)*-Ansatzes berücksichtigt.

1.2 Aufgabenstellung

Zum Prototyping im Rahmen einer objektorientierten Systementwicklung werden in dieser Arbeit Vorgehensweisen entwickelt, die auf verschiedene Fallbeispiele angewendet werden. Als Fallbeispiele werden Anwendungen betrachtet, die unterschiedliche Eigenheiten von Datenbankanwendungen widerspiegeln. Im einzelnen handelt es sich dabei um die folgenden Fallstudien:

Bibliotheksverwaltung: Ein System zur Bibliotheksverwaltung repräsentiert eine traditionelle Datenbankanwendung. Das System ist charakterisiert durch die Verwaltung großer Mengen von Daten, zwischen denen vielfältige Beziehungen bestehen, durch die Unterstützung verschiedener Benutzergruppen und durch eine große Anzahl von Benutzern. Die meisten Daten haben eine lange *Lebensdauer* und werden nur selten modifiziert. Unter der Lebensdauer eines Datums wird hier der Zeitraum von dem Erzeugen des Datums bis zum Löschen verstanden. Die Definition der Anforderungen an das zu modellierende System orientiert sich an den Anforderungen der Bereichsbibliothek Informatik der Universität Dortmund.

E-Mail-System: Bei einem E-Mail-System werden Daten zwischen verschiedenen Benutzern ausgetauscht. Das System ist durch eine hohe Anzahl von Modifikationen des Datenbestandes gekennzeichnet (z.B. Erzeugen und Löschen von Nachrichten). Es werden wesentlich weniger Daten benötigt als beim System zur Bibliotheksverwaltung.

Pflegedienst- und Terminplanung im stationären Krankenhausbereich: Das System zur Pflegedienst- und Terminplanung im stationären Krankenhausbereich besteht aus mehreren miteinander kooperierenden Komponenten. Ein erster Prototyp soll ein System zur Unterstützung der Pflegedienstplanung modellieren. Als Erweiterung dieses Systems soll anschließend ein Terminplanungssystem des Krankenhauspersonals entwickelt werden.

Jedes der drei Fallbeispiele wird im Zusammenhang mit einer anderen der bei Floyd [Flo84] beschriebenen Prototyping-Arten

- exploratives Prototyping,
- experimentelles Prototyping und
- evolutionäres Prototyping

untersucht.

Das System zur Bibliotheksverwaltung wird im Zusammenhang mit explorativem Prototyping betrachtet. Im Rahmen eines „realen“ Projektes sollte der zu erstellende Prototyp dazu dienen können, die Anforderungen an ein zu entwickelndes System zwischen Anwendern und Software-Entwicklern zu klären. Ein solcher Prototyp wird bei Kieback et al. [KLSZ92] als Prototyp im engeren Sinne bezeichnet. Der Ausdruck „reales“ Projekt wird in dieser Arbeit zur Bezeichnung für ein Projekt verwendet, in dem ein Software-System für den tatsächlichen Einsatz in der Praxis entwickelt werden soll.

Das E-Mail-System wird als Beispiel zum experimentellen Prototyping untersucht. Anhand des zu erstellenden Prototyps sollen technische Fragen geklärt werden können, die sich für Software-Entwickler im Rahmen eines „realen“ Projektes ergeben könnten. Einen solchen Prototyp bezeichnen Kieback et al. als Labormuster.

Das System zur Pflegedienst- und Terminplanung im Krankenhausbereich wird im Zusammenhang mit evolutionärem Prototyping betrachtet. Hierbei wird zunächst ein Prototyp als Pilotsystem [KLSZ92, BKKZ92] entwickelt, das bereits grundlegende Funktionen für die Arbeit der Endanwender bereitstellt. Das Pilotsystem wird anschließend erweitert.

Auf der Basis von Modellen, die im Rahmen des Objectory-Entwicklungsprozesses erstellt werden, werden die geforderten ProSet-Prototypen implementiert. Ergänzend zu Objectory werden grundlegende Ideen des bei Gryczan und Züllighoven sowie bei Bäumer et al. vorgestellten WAM-Ansatzes für die Systementwicklung übernommen und in den Objectory-Entwicklungsprozeß integriert.

Die Modellierung der Benutzerschnittstellen spielt in der Arbeit eine untergeordnete Rolle. Die Benutzerschnittstellen werden jedoch gekapselt, um spätere Erweiterungen der Oberfläche zu vereinfachen.

In dieser Diplomarbeit sollen die entwickelten Vorgehensweisen zum Prototyping, die Sprachmittel von ProSet für das Prototyping mit persistenten Daten und die verfügbaren Werkzeuge abschließend bewertet werden. Die Ergebnisse der Arbeit sollen Aufschlüsse für verbesserte Methoden, Techniken und Werkzeuge für das Protoyping mit persistenten Daten liefern.

1.3 Überblick

Im Mittelpunkt dieser Arbeit steht die Erstellung von ProSet-Prototypen auf der Grundlage von Objectory-Modellen unter Verwendung von Konzepten des WAM-Ansatzes (siehe Abschnitt 1.2). Der zu einem Fallbeispiel betrachtete Entwicklungsprozeß wird in dieser Diplomarbeit immer dann beendet, wenn die zu diesem Fallbeispiel zu implementierenden ProSet-Prototypen erstellt sind. Eine Untersuchung der vollständigen Entwicklungsprozesse bei allen betrachteten Fallbeispielen bis zur Implementierung von Anwendungssystemen, die in der Praxis tatsächlich eingesetzt werden könnten, überstiege zum einen den Umfang der Diplomarbeit, zum anderen fehlt in dieser Diplomarbeit der Bezug zu den Anwendern. So wäre beim explorativen und beim evolutionären Prototyping eine Diskussion zwischen Anwendern und Entwicklern über die erstellten Prototypen nötig. Anwender, mit denen über die zu entwerfenden Systeme diskutiert werden könnte, stehen bei dieser Diplomarbeit jedoch nicht zur Verfügung.

Bei den einzelnen Fallbeispielen geht es nicht darum, Systeme zu entwerfen, die in speziellen Anwendungsbereichen tatsächlich eingesetzt werden sollen. Die Ist-Analysen der Anwendungsbereiche nehmen in dieser Diplomarbeit daher weniger Raum ein, als dies üblicherweise bei der Software-Entwicklung der Fall ist. Dieser Kompromiß ist nötig, weil die Durchführung ausführlicher Ist-Analysen für jedes Fallbeispiel und die damit verbundene notwendige intensive Einarbeitung in die Anwendungsbereiche der betrachteten Fallbeispiele zusätzlich zu den sonstigen Arbeiten, die in den betrachteten Entwicklungsprozessen durchgeführt werden, mehr Zeit in Anspruch nähme, als für die Diplomarbeit zur Verfügung steht.

Die Diplomarbeit ist in sechs Kapitel unterteilt. Im ersten Kapitel nach der Einleitung (Kapitel 2) werden zunächst Grundlagen für die Bearbeitung der Fallbeispiele in dieser Diplomarbeit beschrieben. Anschließend werden die Arbeiten zu den drei Fallbeispielen in getrennten Kapiteln vorgestellt. Dabei wird im Kapitel 3 auf das System zur Bibliotheksverwaltung eingegangen. Im Kapitel 4 sind die Ausführungen zum E-Mail-System enthalten. Kapitel 5 enthält die Beschreibungen zum System zur Pflegedienst- und Terminplanung im stationären Krankenhausbereich. Im Kapitel 6 werden zusätzliche Bemerkungen und Ausblicke zu den durchgeführten Arbeiten aufgeführt und die geforderten abschließenden Bewertungen (siehe Abschnitt 1.2) vorgenommen.

Zusätzlich wird zu dieser Diplomarbeit ein separates Dokument erstellt, in dem Glossare und Objectory-Modelle zu den betrachteten Fallbeispielen, sowie der ProSet-Quellcode und Hinweise zur Installation und zum Start der implementierten Prototypen aufgeführt sind. Dieses Dokument wird im folgenden als *Zusatzdokument* bezeichnet.

1.4 Rahmenbedingungen

In dieser Arbeit wird die Kenntnis von ProSet und von Objectory vorausgesetzt. Die Befehle von ProSet werden bei Doberkat et al. [DFH+95] beschrieben. Die Grundlagen von Objectory werden bei Jacobson et al. [JCJÖ93] vorgestellt (siehe Jacobson et al. [JCJÖ93, S. 503]). Die

Ausführungen von Jacobson et al. bilden die theoretische Basis für die Erstellung der Objectory-Modelle in dieser Diplomarbeit. Abweichungen zwischen den Ausführungen von Jacobson et al. und den Beschreibungen in der Objectory-Dokumentation [Ory95, Ory94a, Ory94b, Ory94c] in Bezug auf die zu erstellenden Modelle treten nur in Ausnahmefällen auf. Falls Abweichungen auftreten, die für diese Diplomarbeit relevant sind, wird hierauf bei der Beschreibung der entsprechenden Modelle eingegangen.

Die Begriffe, die in dieser Arbeit im Zusammenhang mit der objektorientierten Systementwicklung verwendet werden, werden, soweit es nicht anders angegeben ist, gemäß der Begriffsbestimmungen verwendet, die bei Jacobson et al. gegeben werden. Insbesondere gilt dies für die Begriffe *Objekt*, *Klasse* und *Instanz*. Demnach ist ein Objekt durch eine Anzahl von Operationen und einen Zustand, der die Auswirkungen dieser Operationen festhält, gekennzeichnet [JCJÖ93, S. 44]. Eine Klasse repräsentiert eine Schablone für mehrere Objekte und beschreibt, wie diese Objekte intern strukturiert sind. Sowohl die Definition der Operationen als auch die Defintion der Strukturen, in denen Informationen gespeichert werden, sind bei allen Objekten, die zur selben Klasse gehören, gleich. Instanzen sind Objekte, die von einer Klasse erzeugt werden [JCJÖ93, S. 50]. Da nach Jacobson et al. in objektorientierten Systemen jedes Objekt zu einer Klasse gehört, werden die Begriffe Objekt und Instanz in dieser Arbeit synonym verwendet.

Für die Implementierung der Prototypen wurde der ProSet-Compiler in der Version 0.6 benutzt. Dabei wurde der von ProSet zur Verfügung gestellte Persistenzmechanismus verwendet. Neben dem ProSet-Compiler wurden die folgenden ProSet-spezifischen Hilfsmittel benutzt:

- der in der Diplomarbeit von Kappert [Kap95] vorgestellte H-PCTE-Server,
- die graphische ProSet-Compiler-Benutzeroberfläche xpst,
- die Hilfsprogramme xpfed und xpft zum Anlegen, Bearbeiten und Löschen von P-Files.

Alle Prototpen wurden auf einer SUN-SPARCstation unter SunOS 4.1.3 implementiert. Bei der Erstellung der Prototypen wurden neben den ProSet-spezifischen Hilfsmitteln noch die Werkzeuge ar, axe, emacs, gcc, gzip, tar, vi und xfig benutzt. Beim zweiten Fallbeispiel (siehe Abschnitt 1.2) wurde zusätzlich das Software-System *PVM* [GAJ+94] verwendet. Zur Unterstützung der Erstellung der Objectory-Modelle war ursprünglich die Verwendung des Werkzeugs Objectory SE in der Version 3.6 vorgesehen. Die Lizenz für dieses Werkzeug lief jedoch kurz nach dem Beginn dieser Arbeit aus und wurde nicht verlängert. Bei der Bearbeitung des zweiten und des dritten Fallbeispiels konnte deshalb, im Gegensatz zur Bearbeitung des ersten Fallbeispiels, nicht mehr auf Objectory SE zurückgegriffen werden.

Kapitel 2

Grundlagen

In diesem Kapitel werden Grundlagen vorgestellt, die für die Diplomarbeit benötigt werden. Dazu wird im Abschnitt 2.1 ein Überblick über Grundlagen von Objectory gegeben. Abschnitt 2.2 stellt Konzepte des WAM-Ansatzes zur objektorientierten Analyse und zum objektorientierten Systementwurf vor. Im Abschnitt 2.3 wird auf das Prototyping eingegangen. Anschließend wird im Abschnitt 2.4 eine Übersicht über grundlegende Konzepte zur Bearbeitung der in dieser Diplomarbeit betrachteten Fallbeispiele gegeben. Dabei werden zum einen die geplanten Vorgehensweisen im Objectory-Entwicklungsprozeß bis zum Prototyping beschrieben. Zum anderen werden grundsätzliche Überlegungen zur Integration von Konzepten des WAM-Ansatzes in den Objectory-Entwicklungsprozeß und zur Abbildung von Modellen, die im Rahmen einer objektorientierten Systementwicklung erstellt werden, auf ProSet-Implementierungen vorgestellt.

2.1 Objectory

Objectory basiert auf dem OOSE-Konzept nach Jacobson et al. [JCJÖ93]. In diesem Abschnitt werden nur grundlegende Ideen von Objectory bzw. OOSE skizziert. Für eine umfassende Beschreibung sei auf die oben angegebene Literatur verwiesen.

OOSE ist eine Abkürzung für *object-oriented software engineering*. Eine Übersicht über die Komponenten, aus denen Objectory besteht, geben Jacobson et al. [JCJÖ93, S. 515ff.]. In Objectory wird mit fünf verschiedenen Modellen gearbeitet. Dabei wird davon ausgegangen, daß eine Beschreibung der Anforderungen bereits existiert, bevor mit den Modellen gearbeitet wird [JCJÖ93, S. 110]. Die Objectory-Modelle werden im folgenden kurz vorgestellt.

Anforderungsmodell: Aus der Beschreibung der Anforderungen wird das *Anforderungsmodell* (*requirements model*) entwickelt. Das Anforderungsmodell besteht aus den folgenden drei Teilen:

- Im *Anwendungsfallmodell* (*use case model*) werden die Anforderungen an das zu entwickelnde System in Form von *Anwendungsfällen* festgelegt. Anwendungsfälle beschreiben, wie ausgewählte Arbeitssituationen durch das zu entwickelnde System ausgeführt werden sollen. Die geforderte Funktionalität des Systems ergibt sich aus der Menge aller Anwendungsfälle. Zusätzlich wird im Anwendungsfallmodell festgelegt, welche *Akteure* (z.B. Menschen, Computer) Aktionen des Systems veranlassen können.
- *Schnittstellenbeschreibungen* spezifizieren wichtige Schnittstellen des Systems nach außen. Hierzu gehören Bildschirmmasken und Protokolle mit anderen Systemen.
- Das *Problembereichsmodell* (*domain object model*) enthält alle für das System wichtigen Objekte (zur Beschreibung des Begriffs Objekt siehe Abschnitt 1.4), für die eine direkte Entsprechung im Anwendungsbereich des Systems gefunden werden kann. Beispiele für mögliche Objekte im Problembereichsmodell eines Systems zur Bibliotheksverwaltung

sind Bücher und Zeitschriftenbände. Die internen Strukturen der Objekte Buch und Zeitschriftenband sind in den Klassen Buch und Zeitschriftenband beschrieben.

Problembereichsmodelle in verschiedenen Anwendungen können unterschiedlich ausführlich sein. So kann bereits die Funktionalität des zu entwickelnden Systems vollständig oder teilweise beschrieben werden. Es ist aber auch möglich, sich auf die Beschreibung wesentlicher Begriffe aus dem Anwendungsbereich und die Darstellung ihrer Beziehungen zueinander zu beschränken [JCJÖ93, S. 168f.].

Analysemodell: Das Anforderungsmodell bildet die Grundlage für das *Analysemodell* (*analysis model*). Hier werden alle für das System benötigten Objekte und deren Beziehungen zueinander beschrieben. Im Gegensatz zum Problembereichsmodell werden auch Objekte verwendet, denen keine direkte Entsprechung aus dem Anwendungsbereich zugeordnet werden kann. Das Ziel bei der Erstellung des Analysemodells ist der Entwurf einer änderungsfreundlichen und einfach erweiterbaren Systemstruktur. Dies soll dadurch erreicht werden, daß Änderungen oder Erweiterungen des Systems durch lokale Änderungen oder Erweiterungen einzelner Objekte vorgenommen werden können. Bei der Erstellung des Analysemodells wird von *idealen Umgebungsbedingungen* ausgegangen. Das bedeutet, daß spezielle Erfordernisse und Einschränkungen, die sich durch die verwendete Software und Hardware ergeben, noch nicht beachtet werden.

Neu gewonnene Kenntnisse bei der Erstellung des Analysemodells können eine Überarbeitung des Anforderungsmodells erfordern.

Entwurfsmodell: Das *Entwurfsmodell* (*design model*), wird auf der Grundlage des Analysemodells und des Anforderungsmodells erstellt. Es beschreibt das zu entwickelnde System unter Berücksichtigung der verwendeten Hard- und Software. Die Struktur des Analysemodells soll soweit wie möglich erhalten bleiben. Analog zum Analysemodell können auch neue Erkenntnisse bei der Entwicklung des Entwurfsmodells zu Änderungen im Anforderungsmodell oder im Analysemodell führen.

Implementierungsmodell: Auf der Basis des Entwurfsmodells wird das *Implementierungsmodell* (*implementation model*) entwickelt. Dieses Modell besteht aus dem Code in der Programmiersprache, in der das System entwickelt wird. Bei der Implementierung können Fehler im Entwurfsmodell deutlich werden. In diesem Fall müssen das Entwurfsmodell und evtl. auch vorhergehende Modelle verbessert werden.

Testmodell: Zum Test der Implementierung wird das *Testmodell* (*testing model*) entworfen. Dieses Modell besteht aus der Spezifikation der durchzuführenden Tests und aus den Testergebnissen. Abhängig vom Ausfall der Tests kann es nötig sein, früher erstellte Modelle zu überarbeiten.

Jacobson et al. favorisieren bei der Erstellung der einzelnen Modelle eine inkrementelle Vorgehensweise. Nur das Anforderungsmodell soll schon im ersten Schritt weitgehend vollständig erstellt werden [JCJÖ93, S. 456ff.]. Als eine mögliche Minimalforderung für die Erstellung des Anforderungsmodells erwähnen Jacobson et al. dabei die Identifikation aller Anwendungsfälle des zu entwickelnden Systems, bevor mit der Erstellung des Analysemodells begonnen wird. Die einzelnen Inkremente können sich zeitlich überlappen. Als Richtwert für den Umfang der Inkremente geben Jacobson et al. fünf bis 20 Anwendungsfälle an. Wenn alle Anwendungsfälle vom Anwendungsfallmodell in alle anderen Modelle übertragen sind, ist die erste Version des zu entwickelnden Systems erstellt. Zur Ergänzung des Software-Entwicklungsprozesses schlagen Jacobson et al. Prototyping vor [JCJÖ93, S. 25f.].

Die in der Diplomarbeit betrachteten Beispiele für experimentelles und evolutionäres Prototyping, besitzen weniger als 20 Anwendungsfälle. Der Prototyp beim experimentellen Prototyping bzw. das Pilotsystem beim evolutionären Prototyping können können deshalb innerhalb eines Objectory-Inkrements entwickelt werden. Der Prototyp zum Fallbeispiel für exploratives Prototyping wird als Ergänzung zu einem vollständigen Anforderungsmodell entworfen. Eine inkrementelle Entwicklung das Prototyps wird auch in diesem Fall nicht betrachtet.

2.2 WAM-Ansatz

Im folgenden werden Konzepte des WAM-Ansatzes zur objektorientierten Analyse und zum objektorientierten Entwurf, die bei Gryczan und Züllighoven [GZ92] und bei Bäumer et al. [BGZ95] beschrieben werden, zusammengefaßt. Konzepte des WAM-Ansatzes werden im Rahmen der Diplomarbeit als Ergänzung zu Objectory (siehe Abschnitt 2.1) betrachtet.

Nach Gryczan und Züllighoven und Bäumer et al. ist es bei der Entwicklung eines Software-Systems wichtig, daß sich Anwender und Software-Entwickler auf eine gemeinsame *Projektsprache* einigen. Unter einer gemeinsamen Projektsprache wird die einheitliche Verwendung von Begriffen zwischen allen an der Software-Erstellung beteiligten Personen verstanden. Um dies zu erreichen, wird ein *Glossar* erstellt, in dem Begriffe aus dem Anwendungsbereich und Begriffe, die bei der Software-Entwicklung neu entstehen, eindeutig und für alle Beteiligten verbindlich definiert werden. Bei der Definition von Begriffen aus dem Anwendungsbereich werden in erster Linie die verwendeten *Gegenstände* betrachtet. Dabei werden im folgenden unter Gegenständen nicht ausschließlich materielle Dinge verstanden. Gegenstände können auch abstrakt sein, wie z.B. ein Konto oder ein Zinssatz. Der Begriff Gegenstand dient hier als Abgrenzung zu Tätigkeiten. Gegenstände können bei der Erstellung von *Szenarien* identifiziert werden. In Szenarien werden typische Arbeitssituationen des Anwendungsbereichs beschrieben. Die Beschreibung von Szenarien ist ein Hilfsmittel für die Software-Entwickler, um sich in den Anwendungsbereich einzuarbeiten. Bei Diskussionen zwischen Entwicklern und Anwendern des Software-Systems über die erstellten Szenarien wird überprüft, ob die Entwickler die beschriebenen Situationen richtig erfaßt haben.

Nach der Beschreibung der Szenarien werden die gefundenen Gegenstände in *Werkzeuge* und *Materialien* eingeteilt. Diese Begriffe werden im folgenden gemäß der Beschreibungen verwendet, die bei Bäumer et al. gegeben werden [BGZ95, S. 48]:

> „Ein Material ist ein Gegenstand, der im Arbeitszusammenhang in das Ergebnis unserer Arbeit eingeht. Materialien zu bearbeiten bedeutet, mit Werkzeugen zu hantieren. Jedes Material hat seine anwendungsspezifische Funktionalität und kann durch geeignete Werkzeuge auf seinen Zustand überprüft oder verändert werden.“

> „Wir verwenden Werkzeuge, um eine Aufgabe zu erledigen, d.h. ein Arbeitsergebnis herzustellen. Sie sind im Arbeitszusammenhang die Arbeitsmittel zur Untersuchung und zur Veränderung von Materialien.“

Unter dem *Arbeitszusammenhang* wird dabei die Situation verstanden, in der ein Gegenstand als Werkzeug oder als Material identifiziert wird. Dem Arbeitszusammenhang kommt deshalb eine wesentliche Bedeutung zu, weil sich nach Bäumer et al. nicht kontextfrei entscheiden läßt, ob ein Gegenstand als Werkzeug oder als Material angesehen werden kann. Als Beispiel erwähnen Bäumer et al. einen Bleistift, der beim Schreiben ein Werkzeug, beim Anspitzen jedoch ein Material ist.

Anhand von Beziehungen zwischen Gegenständen des Anwendungsbereichs werden Klassenhierarchien erzeugt. Diese Hierarchien werden als *fachliche Klassenhierarchien* bezeichnet. Das Erzeugen der fachlichen Klassenhierarchien wird als *fachlicher Klassenentwurf* bezeichnet. Ziel des fachlichen Klassenentwurfs ist die Erstellung eines zyklenfreien Begriffsgerüsts der Anwendungswelt, in dem alle Gegenstände des Anwendungsbereichs, die für die betrachtete Anwendung als wichtig angesehen werden, aufgeführt sind. Der fachliche Klassenentwurf orientiert sich vor allem an den Materialien des Anwendungsbereichs, da Werkzeuge nach Gryczan und Züllighoven üblicherweise nicht direkt aus dem betrachteten Problembereich in das zu erstellende Software-System übernommen werden können. Ein Beispiel für einen Ausschnitt aus einer Begriffshierarchie eines fachlichen Klassenentwurfs zur Beschreibung des Problembereichs eines Studenten bei einer Prüfungsvorbereitung, ist in der Abbildung 2.1 dargestellt. Dieses Beispiel soll die grundsätzliche Idee des fachlichen Klassenentwurfs veranschaulichen und erhebt keinen Anspruch auf Vollständigkeit. Die Lernmittel werden hier in private und öffentliche Lernmittel eingeteilt. Bei den privaten Lernmitteln handelt es sich um Vorlesungsmitschriften. Zu öffentlichen Lernmitteln zählen Bücher Skripten und Folienkopien.

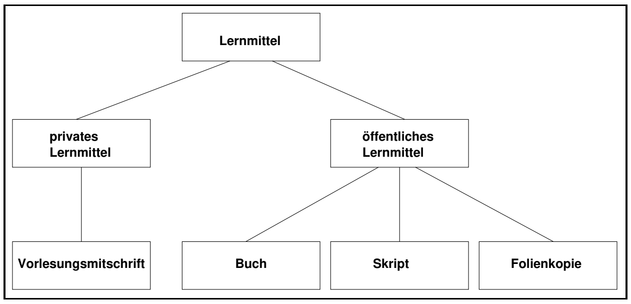

Abbildung 2.1: Beispiel für einen Ausschnitt aus einer fachlichen Klassenhierarchie.

Nach der Erstellung fachlicher Klassenhierarchien werden *Systemvisionen* erstellt. Systemvisionen beschreiben, wie ausgewählte Arbeitssituationen mit Hilfe des zu entwickelnden Software-Systems abgearbeitet werden sollen. Die Systemvisionen bilden die Grundlage für die Erstellung von Prototypen.

Anschließend wird der *technische Klassenentwurf* durchgeführt. Beim technischen Klassenentwurf werden unter Verwendung der „Mittel der objektorientierten Programmierung“ [GZ92, S. 269] aus den „Verständnishierarchien“ des fachlichen Klassenentwurfs die Klassen entworfen, aus denen das zu entwickelnde System bestehen soll. Unter den Mitteln der objektorientierten Programmierung verstehen Gryczan und Züllighoven hier Ein- und Mehrfachvererbung, den Entwurf von abstrakten Datentypen und Zusicherungen. Dabei können die im fachlichen Klassenentwurf entworfenen Hierarchien noch verändert werden. Als Beispiel erwähnen Gryczan und Züllighoven die Zusammenfassung von Klassen aus verschiedenen fachlichen Klassenhierarchien.

Die Werkzeugklassen und die Materialklassen für das zu erstellende Software-System werden getrennt voneinander entworfen. Während Materialien bei der Entwicklung eines Software-Produktes in der Regel bereits beim fachlichen Klassenentwurf aus der realen Welt direkt übernommen und auf Klassen abgebildet wurden, müssen Werkzeugklassen meist vollständig neu entwickelt werden.

Die Materialklassen bieten nach außen Methoden an, mit denen die denen die Zustände der Materialien verändert werden können. Diese Methoden sind jedoch nicht für den Benutzer des Software-Produktes sichtbar. Er kann Materialien nur mit Hilfe von Werkzeugen bearbeiten. Um eine getrennte Entwicklung von Werkzeug- und Materialklassen zu ermöglichen, werden zusätzliche Klassen entwickelt, die die Schnittstelle zwischen Werkzeug- und Materialklassen bilden. Diese Klassen werden als *Aspektklassen* bezeichnet.

Wegen der Trennung von Werkzeug-, Aspekt- und Materialklassen, wird der in diesem Abschnitt beschriebene Ansatz zur Software-Entwicklung auch als WAM(Werkzeug, Aspekt, Material)-Ansatz bezeichnet. Ergänzt werden die beschriebenen Klassen noch um *Automatenklassen*. Durch Automaten können Arbeitsroutinen abgearbeitet werden, die immmer nach dem gleichen Schema ablaufen und ein vorher festgelegtes Ergebnis hervorbringen. Im Gegensatz zu Werkzeugen können Automaten menschliche Arbeitsweisen ersetzen. Als Beispiel für mögliche Einsatzbereiche für Automaten geben Bäumer et al. die regelmäßige Überprüfung der Gültigkeit von Risikoerklärungen im Wertpapiergeschäft von Banken oder bei der Zahlung von Daueraufträgen an [BGZ95, S. 48].

2.3 Prototyping

In diesem Abschnitt werden zunächst verschiedene Arten des Prototyping vorgestellt (zur Erläuterung des Begriffs Prototyping siehe Abschnitt 1.1). Anschließend wird kurz auf die Sprache ProSet eingegangen.

2.3.1 Prototyping-Arten

Beim Prototyping können unterschiedliche Ziele verfolgt werden. Abhängig vom verfolgten Ziel werden nach Floyd [Flo84]

- exploratives Prototyping,
- experimentelles Prototyping und
- evolutionäres Prototyping

unterschieden. Die wesentlichen Kennzeichen dieser Prototyping-Arten werden bei Budde et al. zusammengefaßt [BKKZ92, S. 38f.]. An dieser Zusammenfassung orientiert sich die folgende Übersicht über die Prototyping-Arten:

Exploratives Prototyping: Beim *explorativen Prototyping* (*exploratory prototyping*) sollen mit Hilfe von Prototypen die Anforderungen an ein zu entwickelndes System zwischen Anwendern und Entwicklern geklärt werden. Die erstellten Prototypen sind ein Teil der Spezifikation für das zu entwickelnde Software-Produkt.

Experimentelles Prototyping: Beim *experimentellen Prototyping* (*experimental prototyping*) steht die technische Umsetzung eines Problems im Vordergrund. Experimentelles Prototyping kann angewendet werden, wenn über die grundsätzlichen Anforderungen an das zu entwickelnde System zwischen Anwendern und Entwicklern Einvernehmen besteht und noch technische oder software-ergonomische Fragen geklärt werden müssen. Entwickler können anhand der erstellten Prototypen die technische Umsetzbarkeit eines Problems überprüfen.

Evolutionäres Prototyping: Beim *evolutionären Prototyping* (*evolutionary prototyping*) wird zunächst ein Kern eines größeren Anwendungssystems erstellt, der grundlegende Funktionen bereitstellt und bereits von den Anwendern genutzt werden kann. Dieser Kern wird als Pilotsystem bezeichnet. In weiteren Schritten wird das Pilotsystem ständig erweitert und an veränderte Umgebungsbedingungen angepaßt.

2.3.2 ProSet

ProSet [DFG+92, DFKS93, DFH+95] ist eine mengentheoretisch orientierte imperative Programmiersprache zur Erstellung von Prototypen in der Nachfolge von *SETL* [DF89]. ProSet ist ein Akronym für *Prototyping with Sets*. Im folgenden werden nur einige Grundlagen von ProSet kurz aufgeführt. Für eine umfassende Einführung sei auf die oben angegebene Literatur verwiesen.

ProSet ermöglicht die Modellierung von Programmen und Daten und stellt hierzu neben primitiven Datentypen, die von anderen imperativen Programmiersprachen her bekannt sind, wie z.B. ganze Zahlen oder Zeichenketten, noch Tupel, Mengen, Funktionen, Moduln und Modulinstanzen als grundlegende Datentypen zur Verfügung. Funktionen sind hierbei Prozeduren, bei denen die zum Zeitpunkt der Anwendung gültigen Werte nicht-lokaler Variablen durch die Benutzung des closure-Konstrukts [DFH+95, S. 27f.] „eingefroren“ wurden. Moduln sind Schablonen, die die Arbeitsweise von Operationen auf gemeinsamen Daten beschreiben. Bevor Operationen ausgeführt werden können, die in einem Modul definiert sind, muß zunächst eine Modulinstanz des entsprechenden Moduls erzeugt werden. Anschließend kann die Ausführung von Operationen der erzeugten Modulinstanz veranlaßt werden.

Des weiteren verfügt ProSet über Kontrollstrukturen, die von den gängigen imperativen Programmiersprachen her bekannt sind: for, loop, repeat, while, if und case. Kontrollanweisungen lassen sich in ProSet insbesondere auch auf zusammengesetzte Datentypen (Tupel und Mengen) anwenden. So sind beispielsweise Iterationen über Tupel oder Mengen möglich. Zusätzlich können mit der whilefound-Anweisung [DFH+95, S. 45] Konstrukte erzeugt werden, die speziell auf zusammengesetzte Datentypen abgestimmt sind.

Die Datenmodellierung wird von ProSet durch den Persistenzmechanismus unterstützt. Daten können in Archiven gespeichert werden, um sie über die Ausführung eines Programms hinaus zu erhalten und anderen Programmen zugänglich zu machen. Diese Archive werden *P-Files* genannt. Operationen auf Daten, die in P-Files gespeichert sind, werden im Rahmen von geschachtelten Transaktionen ausgeführt.

Um Fehlersituationen abfangen zu können, ist in ProSet ein Mechanismus zur Ausnahmebehandlung integriert. Zur Behandlung von Ausnahmesituationen können Routinen (*Ausnahme-Handler*) definiert werden, in denen festgelegt ist, wie auf mögliche Ausnahmesituationen reagiert werden soll. Nach der Behandlung einer Ausnahmesituation kann das Programm, das die Ausnahme erzeugt hat, beendet oder an der Stelle, an der die Ausnahme aufgetreten ist, fortgesetzt werden.

Des weiteren bietet ProSet Konstrukte zur Parallelprogrammierung an. Die Möglichkeiten zur Parallelprogrammierung können bei den bisherigen Compilerversionen jedoch nicht gleichzeitig mit dem Persistenzmechanismus benutzt werden. Da in dieser Arbeit das Prototyping mit persistenten Daten betrachtet wird, werden die Möglichkeiten zur Parallelprogrammierung nicht genutzt.

2.4 Vorgehensweisen und grundlegende Konzepte zur Bearbeitung der Fallbeispiele

Die zu erstellenden ProSet-Prototypen werden auf der Basis von Objectory-Modellen entwickelt. Zusätzlich sollen Konzepte des WAM-Ansatzes (siehe Abschnitt 2.2) bei der Systementwicklung berücksichtigt werden. Im folgenden wird zunächst im Abschnitt 2.4.1 ein Überblick über die geplanten Vorgehensweisen in den betrachteten Objectory-Entwicklungsprozessen bis zur Erstellung der geforderten Prototypen gegeben. Im Abschnitt 2.4.2 wird auf die Einbeziehung von Konzepten des WAM-Ansatzes (siehe Abschnitt 2.2) in die Objectory-Entwicklungsprozesse eingegangen. Anschließend werden im Abschnitt 2.4.3 Überlegungen zur Abbildung von Modellen, die im Rahmen einer objektorientierten Systemenwicklung erstellt werden, auf ProSet-Implementierungen vorgestellt.

2.4.1 Der Objectory-Entwicklungsprozeß bis zum Prototyping

Abhängig von der betrachteten Art des Prototyping werden die Prototypen in verschiedenen Phasen des Objectory-Entwicklungsprozesses erstellt. Im folgenden soll betrachtet werden, wie weit der Entwicklungsprozeß bei der Betrachtung der verschiedenen Prototyping-Arten durchlaufen werden soll, bis die geforderten Prototypen erstellt werden.

Exploratives Prototyping: Prototypen, die beim explorativen Prototyping erstellt werden, sind ein Hilfsmittel, um Anforderungen an ein System zu spezifizieren. Der Prototyp des Systems zur Bibliotheksverwaltung, der im Kapitel 3 betrachtet wird, soll als Prototyp im engeren Sinne (siehe Abschnitt 1.2) das Objectory Anforderungsmodell ergänzen.

Wegen des großen Umfangs des betrachteten Fallbeispiels wird jedoch vor der Implementierung des Prototyps neben dem Objectory-Anforderungsmodell noch ein Teil des Analysemodells erstellt, in dem die benötigten Datenobjekte und ihre Beziehungen zueinander modelliert werden. Dadurch soll die Implementierung des Prototyps vereinfacht werden.

Experimentelles Prototyping: Experimentelles Prototyping dient zur Klärung technischer oder software-ergonomischer Fragen, nachdem die grundlegenden Anforderungen an ein Software-Produkt spezifiziert sind. Der zu erstellende Prototyp wird parallel zum Entwurfsmodell entwickelt. Dabei sollen anhand des Prototyps erstellte Teile des Entwurfsmodells getestet werden können. Des weiteren soll der Prototyp Anregungen für die Weiterentwicklung des Entwurfsmodells geben.

Evolutionäres Prototyping Das Pilotsystem, das beim evolutionären Prototyping erstellt wird, wird so entworfen und implementiert, wie es im Objectory-Entwicklungsprozeß für Systeme, mit denen Endanwender arbeiten sollen, vorgesehen ist. Zur Erweiterung eines Pilotsystems wird der Objectory-Entwicklungsprozeß ein weiteres Mal durchlaufen.

Zusätzlich sollen in dieser Diplomarbeit Konzepte des WAM-Ansatzes in die zu den einzelnen Fallbeispielen betrachteten Entwicklungsprozesse integriert werden. Hierauf wird im folgenden Abschnitt eingegangen.

2.4.2 Einbeziehung von Konzepten des WAM-Ansatzes in den Entwicklungsprozeß

Die in der Diplomarbeit zu entwickelnden ProSet-Prototypen werden auf der Basis von Modellen implementiert, die im Objectory-Entwicklungsprozeß erstellt werden (siehe Abschnitt 2.4.1). Die im Abschnitt 2.2 vorgestellten Konzepte des WAM-Ansatzes sind im Gegensatz dazu Teile des Software-Entwicklungsprozesses, der der Systementwicklung, die bei Gryczan und Züllighoven [GZ92] sowie bei Bäumer et al. [BGZ95] beschrieben wird, zugrunde liegt. Dieser Entwicklungsprozeß unterscheidet sich von dem Objectory-Entwicklungsprozeß.

Die im Abschnitt 2.2 vorgestellten Konzepte können Objectory jedoch an einigen Stellen ergänzen. Im folgenden wird beschrieben, welche Konzepte des WAM-Ansatzes im Zusammenhang mit Objectory bei den zu bearbeitenden Fallbeispielen genutzt werden sollen:

Szenarien: Bei der Entwicklung von Software-Systemen können Szenarien im Rahmen der Ist-Analyse zusätzlich zu den Objectory-Modellen erstellt werden. Objectory bietet kein Modell, das mit Szenarien vergleichbar wäre. In der Diplomarbeit müssen bei der Bearbeitung der Fallbeispiele die Ist-Analysen jedoch aus Zeitgründen verkürzt werden. Deshalb wird bei den betrachteten Fallstudien im Rahmen der Ist-Analyse jeweils nur ein allgemeiner Überblick über den betroffenen Anwendungsbereich anstelle der Beschreibung einzelner Szenarien gegeben.

Glossare: Glossare werden in der Diplomarbeit zusätzlich zu den Objectory-Modellen erstellt. Das Werkzeug Objectory SE, das die Erstellung von Objectory-Modellen unterstützt und das zum Beginn dieser Diplomarbeit noch zur Verfügung stand, bietet hierzu den Menüpunkt items an. Hier können alle wichtigen Begriffe aufgeführt und definiert werden.

Systemvisionen: Systemvisionen entsprechen den Objectory-Anwendungsfällen, die im Rahmen dieser Diplomarbeit erstellt werden.

Verwendung der Metaphern Werkzeug und Material: Bei der Erstellung der benötigten Problembereichsmodelle sollen die gefundenen Objekte Werkzeugen oder Materialien zugeordnet werden. Sofern es sich bei der Bearbeitung der Fallbeispiele als sinnvoll erweist, werden die Metaphern Werkzeug und Material auch für Objekte der Analysemodell und der Entwurfsmodelle verwendet. Evtl. wird zusätzlich die Metapher Automat (siehe Abschnitt 2.2) verwendet, falls sich dies als sinnvoll erweist.

Fachlicher Klassenentwurf: Die Aufgaben eines fachlicher Klassenentwurfs — die Erfassung der Gegenstände des Anwendungsbereichs und die Darstellung von Beziehungen zwischen diesen Gegenständen — werden in der Diplomarbeit von den Problembereichsmodellen, die zu den betrachteten Fallbeispielen erstellt werden, übernommen (bei Gegenständen kann es sich auch um abstrakte Dinge, wie z.B. Zinssätze, handeln, siehe hierzu Abschnitt 2.2). Wenn es bei der Bearbeitung der Fallbeispiele sinnvoll erscheint, werden in den betrachteten Problembereichsmodellen im Gegensatz zu fachlichen Klassenentwürfen neben Beziehungen zwischen Klassen auch Beziehungen zwischen Instanzen dargestellt.

Technischer Klassenentwurf: Beim Fallbeispiel zum explorativen Prototyping spielt der technische Klassenentwurf keine Rolle, da der Prototyp zum explorativen Prototyping ein Hilfsmittel zur Spezifikation der Anforderungen an das zu erstellende System sein soll und deshalb vor einem technischen Klassenentwurf implementiert werden würde.

Bei der Betrachtung des experimentellen Prototyping und des explorativen Prototyping im Rahmen dieser Diplomarbeit werden die Klassen und zugehörige Objekte für zu entwickelnden Systeme bei der Erstellung Analysemodelle (ohne Berücksichtigung der verwendeten Soft- und Hardware) und bei der Erstellung der Entwurfsmodelle (unter Berücksichtigung der verwendeten Soft- und Hardware) entworfen. Diese Modelle ersetzen den technischen Klassenentwurf, der in dieser Arbeit nicht weiter berücksichtigt wird.

2.4.3 Abbildung objektorientierter Modelle auf ProSet-Implementierungen

Während im Rahmen von Objectory eine objektorientierte Analyse des Anwendungsbereichs und ein objektorientierter Systementwurf durchgeführt werden, handelt es sich bei ProSet [DFG+92, DFH+95] um eine imperative Programmiersprache. ProSet bietet jedoch Konzepte an, die in Teilbereichen eine objektorientierte Programmierung zulassen.

Im folgenden werden die Konzepte, die nach Jacobson et al. von einer objektorientierten Programmiersprache gefordert werden [JCJÖ93, S. 84], im Zusammenhang mit ProSet betrachtet. Bei den geforderten Konzepten handelt es sich um

- Objekte, in denen Operationen gekapselt sind und die einen Zustand besitzen,
- Klassen und Instanzen,
- Vererbung und
- Polymorphie.

Objekte mit Operationen und Zuständen: Objekte können durch ProSet-Modulinstanzen dargestellt werden. Die Operationen werden in Form von Prozeduren implementiert. Die Zustände werden in Variablen gespeichert.

Klassen und Instanzen: Klassen entsprechen ProSet-Moduln. Die Instanzen der Klassen entsprechen den Modulinstanzen.

Vererbung: Die Vererbung wird durch ProSet nicht unterstützt.

Polymorphie: Nach Jacobson et al. bedeutet Polymorphie, daß der Initiatior einer Kommunikation mit einem beliebigen Objekt A nicht die Klasse dieses Objektes A kennt [JCJÖ93, S. 55]. Das Objekt A kann zu einer beliebigen Klasse gehören. Eine Möglichkeit, Polymorphie innerhalb eines ProSet-Programm zu benutzen, soll im folgenden beispielhaft demonstriert werden:

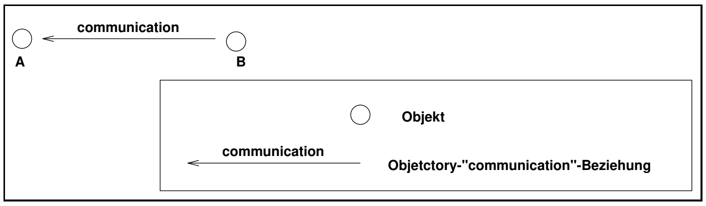

Abbildung 2.2: Beispiel für eine Objectory-„communication"-Beziehung.

Beispiel: Das Beispiel basiert auf der in der Abbildung 2.2 dargestellten „communication"-Beziehung [JCJÖ93, S. 190]. Das Objekt B ist der Initiator einer Kommunikation mit dem Objekt A. Um Aktionen im Objekt A zu veranlassen, sendet das Objekt B einen sog. *Stimulus* an das Objekt A (für weitere Erläuterungen zum Stimuluskonzept siehe Jacobson et al. [JCJÖ93, S. 47]). Im Rahmen der dargestellten Kommunikationsbeziehung kann eine Reaktion auf diesen Stimulus vom Objekt A an das Objekt B zurückgesendet werden.

Wenn Polymorphie genutzt werden soll, darf das Objekt A zu einer beliebigen Klasse gehören. Diese Klasse ist dem Objekt B nicht bekannt. Eine Möglichkeit, diese Anforderungen durch ein ProSet-Programm zu erfüllen, wird im folgenden vorgestellt.

Das Objekt A wird im ProSet-Programm durch eine Modulinstanz A repräsentiert. Das Objekt B wird durch eine ProSet-Modulinstanz B dargestellt. Die ProSet-Modulinstanz A wird persistent gemacht.

Zur Simulation des Sendens eines Stimulus vom Objekt B an das Objekt A lädt die Modulinstanz B des ProSet-Programms zunächst innerhalb einer Prozedur die persistente Modulinstanz A. Anschließend wird in der Prozedur der Modulinstanz B, in der die persistente Modulinstanz A geladen wurde, eine Prozedur der geladenen Modulinstanz A aufgerufen. Dieser Prozeduraufruf repräsentiert das Versenden eines Stimulus.

Der Stimulus, der vom Objekt B an das Objekt A gesendet werden könnte z.B. die Bezeichnung stimulus_von_A_nach_B haben. In diesem Fall könnte Implementierung des ProSet-Moduls zur Instanz B folgendermaßen aussehen:

```
module Modul_zur_Instanz_B;
----------------------------
-- Hier koennten sich noch
-- Schnittstellendefinitonen
-- fuer den Export und den
-- Import und Variablen-
-- deklarationen befinden
----------------------------
...
begin
...
-------------------------
-- beliebige Befehle
-------------------------
...
reaktion :=
stimulus_senden_und_reaktion_empfangen
                              (parameter);
-----------------------------------------
-- Der Variablen Parameter wurde vor dem
-- Aufruf der Prozedur
-- stimulus_senden_und_reaktion_empfangen
-- ein Wert zugewiesen.
-----------------------------------------
...
procedure
   stimulus_senden_und_reaktion_empfangen
                              (rd param);
hidden persistent A: "P_File_mit_Instanz_A";
-------------------------------------------
-- Das P-File "P_File_mit_Instanz_A"
-- enthaelt die persistente Instanz A
-------------------------------------------
begin
   ----------------------------------------
   -- Jetzt wird der Stimulus von A nach B
   -- gesendet und eine Reaktion empfangen.
   ----------------------------------------
   return
      A.stimulus_von_A_nach_B(param);
end stimulus_senden;
...
end Modul_zur_Instanz_B;
```

Weder der Modul zu A noch die Instanz A müssen bei der hier vorgestellten Möglichkeit zur Abbildung der Polymorphie auf ein ProSet-Program in der Importschnittstelle des Moduls zur Instanz B aufgeführt werden.

Damit stellt die Vererbung das einzige von Jacobson et al. für eine objektorientierte Programmiersprache geforderte Konzept dar, das durch ProSet nicht unterstützt wird. Die fehlende Unterstützung bei der Abbildung von Vererbungsbeziehungen durch ProSet wirkt sich bei den in dieser Diplomarbeit betrachteten Fallbeispielen, abhängig von der betrachteten Art des Prototyping, unterschiedlich aus:

Exploratives Prototyping: Der Prototyp für exploratives Prototyping basiert in der Diplomarbeit auf dem Anwendungsfallmodell, dem Problembereichsmodell und einem Teil des Analysemodells, in dem die benötigten Datenobjekte und die Beziehungen zwischen den Datenobjekten dargestellt werden (siehe Abschnitt 2.4.1). Hier kann es vorkommen, daß Vererbungsbeziehungen auf ProSet-Programme abgebildet werden müssen, weil Einschränkungen, die sich durch die verwendete Software oder Hardware ergeben, erst im Objectory-Entwurfsmodell berücksichtigt werden. Auf die Abbildung der beim betrachteten Fallbeispiel für exploratives Prototyping benötigten Vererbungsbeziehungen auf ein ProSet-Programm wird bei der Beschreibung des Fallbeispiels eingegangen (siehe Abschnitt 3.5).

Experimentelles Prototyping: Der Prototyp zum experimentellen Prototyping wird als Ergänzung des Entwurfsmodells erstellt. Im Entwurfsmodell werden bereits Einschränkungen, die sich durch verwendete Soft- und Hardware ergeben, berücksichtigt [JCJÖ93, S. 210]. Welche objektorientierten Konzepte im Entwurfsmodell benutzt werden, hängt davon ab, in welcher *Zielsprache* das Software-System implementiert werden soll. Die Zielsprache ist die Programmiersprache, in der das System, mit dem die Anwender arbeiten, implementiert wird. Diese Sprache kann sich von der Sprache, in der Prototypen des Anwendungssystems erstellt werden, unterscheiden. Bei dem betrachteten Fallbeispiel wird davon ausgegangen, daß auch die Zielsprache des Software-Systems keine Vererbung anbietet.

Ob Vererbungsbeziehungen auf den erstellten ProSet-Prototyp abgebildet werden müssen, hängt davon ab, ob anhand des Prototyps erstellte Teile des Entwurfsmodells getestet werden sollen oder ob anhand des Prototyps auf der Basis des Analysemodells Erkenntnisse für die

Entwicklung des Entwurfsmodells gewonnen werden sollen. Während bei dieser Diplomarbeit im Entwurfsmodell bereits auf Vererbungsbeziehungen verzichtet wird, können sie im Analysemodell noch auftreten.

Auf die in der Diplomarbeit zum experimentellen Prototyping benötigten Konzepte wird im Abschnitt 4.6 detailliert eingegangen.

Evolutionäres Prototyping: Die Einschränkungen, die durch ProSet vorgegeben sind, werden bereits im Objectory-Entwurfsmodell berücksichtigt. Das Problem, Konzepte auf ProSet-Programme abzubilden, die durch ProSet nicht unterstützt werden, tritt daher nicht auf.

Kapitel 3

Erstes Fallbeispiel: Modellierung eines Systems zur Bibliotheksverwaltung

In diesem Kapitel wird die Modellierung eines Systems zur Bibliotheksverwaltung als Beispiel zum explorativen Prototyping betrachtet. Im Rahmen eines „realen" Projektes sollte der zu erstellende Prototyp dazu dienen können, die Anforderungen an das zu entwickelnde System zwischen Anwendern und Software-Entwicklern zu klären. Im Abschnitt 3.1 wird ein Überblick über die Vorgehensweise bei diesem Fallbeispiel gegeben. Die darauf folgenden Abschnitte beschreiben die einzelnen Phasen bis zur Erstellung des Prototyps detaillierter. Abschnitt 3.2 beschreibt die derzeitige Praxis bei der Verarbeitung der anfallenden Daten in einer Bibliothek. Im Abschnitt 3.3 werden die Anforderungen vorgestellt, die an das zu modellierende System und an den Prototyp gestellt werden. Im Abschnitt 3.4 wird auf den Objectory-Entwicklungsprozeß bis zur Erstellung des ProSet-Prototyps eingegangen. Abschnitt 3.5 beschreibt die Erzeugung des ProSet-Prototyps aus den erstellten Objectory-Modellen. Eine Zusammenfassung der gewonnenen Erkenntnisse befindet sich im Abschnitt 3.6.

3.1 Vorgehensweise

Prototypen, die beim explorativen Prototyping erstellt werden, sind Hilfsmittel, um die Anforderungen an ein System zu spezifizieren (siehe Abschnitt 2.3.1). Der zu erstellende Prototyp soll als Prototyp im engeren Sinne (siehe Abschnitt 1.2) als Ergänzung zum Objectory-Anforderungsmodell (siehe Abschnitt 2.1) erstellt werden.

Das Vorgehen im Entwicklungsprozeß bis zur Erstellung des ProSet-Prototyps, das bei diesem Fallbeispiel angewendet wird, wird im folgenden vorgestellt.

Zur Einarbeitung in den Anwendungsbereich schlagen Gryczan und Züllighoven [GZ92] und Bäumer et al. [BGZ95] die Beschreibung von Szenarien, in denen typische Situationen aus dem Anwendungsbereich dargestellt werden, vor (siehe Abschnitt 2.2). In dieser Diplomarbeit wird auf die Beschreibung einzelner Szenarien verzichtet. Stattdessen wird eine allgemeine Beschreibung des Ist-Zustandes im Abschnitt 3.2 gegeben. Eine vollständige Analyse des Anwendungsbereichs und eine ausführliche Beschreibung von Szenarien hätte wesentlich mehr Zeit erfordert, als im Rahmen dieser Diplomarbeit zur Verfügung stand. Begriffe, die für die weitere Software-Entwicklung als wichtig betrachtet werden, sind in einem Glossar aufgeführt. Das vollständige Glossar befindet sich im Zusatzdokument zu dieser Diplomarbeit.

Die Beschreibung des Ist-Zustandes bildet die Grundlage für eine informelle Beschreibung der Anforderungen, die an das zu entwickelnde System gestellt werden. Anschließend werden die benötigten Objectory-Modelle erstellt. Da im Abschnitt 3.4 eine ausführliche Beschreibung der

Modelle erfolgt, wird auf diese Modelle im folgenden nur kurz eingegangen.

Anwendungsfälle (siehe Abschnitt 2.1) beschreiben in der Terminologie des Glossars, wie mit dem zukünftigen System gearbeitet werden soll. Dabei wird das Glossar um weitere Begriffe ergänzt.

Im Problembereichsmodell werden Objekte aus dem Problembereich jedes einzelnen Anwendungsfalls und ihre Beziehungen zueinander dargestellt. Das Problembereichsmodell unterscheidet sich von dem Glossar. Während im Glossar sämtliche Begriffe beschrieben werden, die zur Herausbildung einer einheitlichen Projektsprache (siehe Abschnitt 2.2) notwendig sind, umfaßt das zu dem betrachteten Fallbeispiel erstellte Problembereichsmodell nur Objekte, die die folgenden Bedingungen erfüllen:

- Für ein Objekt kann eine Entsprechung im Problembereich des betrachteten Fallbeispiels gefunden werden.

- Die durch ein Objekt repräsentierte Information wird nicht in einem *Attribut* zu einem anderen Objekt gespeichert. Der Begriff Attribut wird gemäß der Beschreibung von Jacobson et al. verwendet [JCJÖ93, S. 185ff.]. Attribute werden demnach von Objekten benutzt, um Informationen zu speichern. Unter welchen Bedingungen für eine Information ein eigenständiges Objekt modelliert werden soll bzw. unter welchen Bedingungen Informationen in Attributen gespeichert werden sollen, wird bei Jacobson et al. diskutiert [JCJÖ93, S. 188].

  So wird im Glossar des betrachteten Fallbeispiels z.B. die ISBN eines Buches, mit deren Hilfe Bücher identifiziert werden können, aufgeführt, während sie im Problembereichsmodell vernachlässigt wird, weil die ISBN in einem Attribut des Objektes Buch gespeichert werden soll.

  Durch diese Beschränkung des Problembereichsmodells auf die Objekte, die die oben beschriebenen Bedingungen erfüllen, soll verhindert werden, daß das Problembereichsmodell durch die Aufnahme von Objekten, die keine Informationen liefern, die nicht bereits in anderen Objekten enthalten wären, unübersichtlich wird.

Attribute von Objekten werden im Problembereichsmodell noch nicht dargestellt. Die Menge der Beschreibungen der Objekte des Problembereichsmodells ist eine Teilmenge der Menge der Glossareinträge.

Ein vollständiges Anforderungsmodell enthält zusätzlich zum Anwendungsfallmodell und zum Problembereichsmodell noch Schnittstellenbeschreibungen (siehe Abschnitt 2.1). Da Benutzerschnittstellen in dieser Diplomarbeit jedoch nur eine untergeordnete Rolle spielen (siehe Abschnitt 1.2) und Schnittstellen zu fremden Systemen nicht existieren, wird auf Schnittstellenbeschreibungen im Anforderungsmodell verzichtet.

Obwohl der zu erstellende ProSet-Prototyp in erster Linie das Objectory-Anforderungsmodell ergänzen soll, wird wegen des großen Umfangs dieses Fallbeispiels vor der Implementierung des Prototyps noch ein Teil des Analysemodells erstellt, in dem die Beziehungen zwischen den benötigten Datenobjekten bzw. zwischen den Klassen dieser Objekte dargestellt sind (Abschnitt 3.4.2. Hier werden im Gegensatz zum Problembereichsmodell auch Attribute von Objekten modelliert. Anschließend wird der Prototyp erstellt (siehe Abschnitt 3.5).

3.2 Ist-Analyse

Die folgende Beschreibung des Ist-Zustandes einer Bibliothek orientiert sich an der Bereichsbibliothek Informatik der Universität Dortmund. Die Beschreibung bezieht sich auf den Stand vom März 1996.

In der Bereichsbibliothek Informatik sind Bücher, Zeitschriftenhefte und Zeitschriftenbände für den Fachbereich Informatik der Universität Dortmund gesammelt. Bücher können von Benutzern der Bibliothek ausgeliehen werden. Zeitschriftenhefte und Zeitschriftenbände werden nicht verliehen. Es gibt drei Gruppen von Benutzern, die berechtigt sind, Bücher aus der Bibliothek auszuleihen:

- Professoren, wissenschaftliche Mitarbeiter und sonstige Angestellte im Fachbereich Informatik der Universität Dortmund können Bücher für maximal ein Semester ausleihen.
- Professoren, wissenschaftliche Mitarbeiter und sonstige Angestellte anderer Fachbereiche der Universität Dortmund können Bücher für maximal vier Wochen ausleihen.
- Studenten der Universität Dortmund können Bücher über Nacht ausleihen.

Ausgenommen von dieser Regelung sind Bücher, die neu angeschafft wurden, und Bücher, die für Lehrveranstaltungen benötigt werden. Diese Bücher gehören zum *Handapparat* und werden nicht verliehen.

Personen, die Bücher entleihen, müssen einen Benutzerausweis der Zentralbibliothek der Universität Dortmund besitzen. Eine eigene Kartei oder Datenbank der Bereichsbibliothek, in der die Namen der zum Ausleihen berechtigten Personen abgelegt sind, existiert nicht. Die Daten dieser Personen sind in einer Datenbank der Zentralbibliothek gespeichert. Entliehene Bücher und Entleiher werden auf Karteikarten erfaßt. Falls ein Benutzer seine Leihfrist überschreitet, müssen Mahnungen manuell erstellt werden.

Der Bücherbestand der Bereichsbibliothek wird zentral in einer Datenbank des *Hochschul-Bibliotheksinformationszentrums* (*HBZ*) in Köln erfaßt. Hier sind die Bücherbestände sämtlicher Hochschulbibliotheken Nordrhein-Westfalens mit Ausnahme der Buchsammlungen der Lehrstühle gespeichert. Neue Bücher werden in der Bereichsbibliothek mit Hilfe eines Software-Systems *katalogisiert*. Unter dem Begriff *Katalogisierung* wird im Bibliotheksbereich das Erfassen der Daten von Büchern oder Zeitschriften verstanden. Die Daten werden elektronisch an das HBZ gesendet. Wenn ein neues Buch bereits im Datenbestand einer weiteren an das HBZ angeschlossenen Bibliothek katalogisiert ist, können die bereits gespeicherten Daten übernommen werden. Der Bibliothekar muß nur die Daten eingeben, die von den übernommenen Daten abweichen.

Der Katalog, in dem Bibliotheksbenutzer per Computer nach Büchern der Bereichsbibliothek Informatik sowie aller anderer Bibliotheken der Universität Dortmund suchen können, wird vom Hochschulrechenzentrum der Universität Dortmund bereitgestellt. Hierzu verwaltet das Hochschulrechenzentrum eigene Datenbanken mit den Bücherbeständen der Bibliotheken der Universität Dortmund. Die Daten hierfür stammen vom HBZ. Zur Zeit werden diese Datenbanken halbjährlich aktualisiert.

Der Zeitschriftenbestand der Bereichsbibliothek ist in der zentralen *Zeitschriftendatenbank* (*ZDB*) in Berlin erfaßt. Hier werden bundesweit die Zeitschriftenbestände von Universitätsbibliotheken und von anderen großen Bibliotheken gespeichert. Die Erfassung von Zeitschriften funktioniert analog zu der Erfassung von Büchern. Es existiert jedoch kein Publikumskatalog, in dem per Computer eine Literaturrecherche im Zeitschriftenbestand der Bereichsbibliothek durchgeführt werden könnte. Halbjährlich erhält die Bereichsbibliothek Informatik eine Übersicht über die in der ZDB erfaßten Zeitschriften auf Mikrofiches.

Die Daten der Mitarbeiter und der *Geräte* sind in Datenbanken der Zentralbibliothek der Universität Dortmund gespeichert. Unter Geräten wird hier das neben den Büchern und Zeitschriften zusätzlich vorhandenene Inventar der Bibliothek verstanden (z.B. Computer, Tische).

3.3 Beschreibung der Anforderungen an das System und an den Prototyp

Bei der Beschreibung der Anforderungen dient die Ist-Analyse aus dem Abschnitt 3.2 als Grundlage. Zugunsten einer umfangreicheren Datenmodellierung wird bei dem zu erstellenden System an den folgenden Stellen von den in der Ist-Analyse beschriebenen Grundlagen abgewichen.

Verwaltung des Personals und der Geräte: Das zu entwerfende System soll die Möglichkeit bieten, Daten des Personals und der Geräte der Bibliothek zu verwalten. Diese Möglichkeit wird von der Bereichsbibliothek Informatik nicht benötigt, weil Daten des Personals und der Geräte von der Zentralbibliothek der Universität Dortmund verwaltet werden.

Verwaltung der Daten der Benutzer: Im Gegensatz zur Bereichsbibliothek Informatik werden Namen und Adressen der Benutzer der Bibliothek gespeichert.

Online-Literaturrecherche für Zeitschriften: Das zu entwerfende System soll die Möglichkeit bieten, Daten von Zeitschriften in einer Datenbank abzulegen und in dieser Datenbank auch eine Literaturrecherche durchzuführen.

Insgesamt sollen die folgenden Leistungen eines Bibliotheksverwaltungssystems durch den erstellten Prototyp simuliert werden:

Verwaltung des Bücherbestandes: Daten neu angeschaffter Bücher können in einer Datenbank aufgenommen werden. Die Daten können geändert und gelöscht werden.

Verwaltung des Zeitschriftenbestandes: Daten von Zeitschriften können in einer Datenbank aufgenommen werden. Es ist möglich, alte Daten zu ändern oder zu löschen. Es werden sowohl die zu einer Zeitschrift vorhandenen einzelnen Hefte als auch Zeitschriftenbände verwaltet.

Verwaltung der Benutzer: Daten über die Benutzer der Bibliothek werden gespeichert. Dabei soll es möglich sein, verschiedene Benutzergruppen mit unterschiedlichen Rechten zu verwalten. Die Einteilung der Gruppen und die Informationen, die über die Angehörigen der einzelnen Gruppen gespeichert werden, können flexibel an die Erfordernisse der Bibliothek angepaßt werden. Sie sind nicht starr durch das Programm vorgegeben.

Verwaltung der ausgeliehenen Bücher, Zeitschriftenhefte und Zeitschriftenbände: Es wird gespeichert, wer welche Bücher, Zeitschriftenhefte und Zeitschriftenbände ausgeliehen hat und bis wann die Bücher, Zeitschriftenhefte und Zeitschriftenbände zurückgegeben werden müssen. Zusätzlich wird die Möglichkeit angeboten, die Fristen für einzelne Entleihungen um einen vorgegebenen Zeitraum zu verlängern.

Überprüfung der Einhaltung von Fristen: Die Namen der Personen, die noch Bücher, Zeitschriftenhefte oder Zeitschriftenbände besitzen, die sie bereits hätten zurückgeben müssen, können ermittelt werden. Eine solche Funktion kann bei einer späteren Erweiterung des Prototyps dazu ausgebaut werden, direkt Mahnungen zu erstellen.

Suchen von Büchern und Zeitschriften: Das System bietet die Möglichkeit, nach Büchern und Zeitschriften zu suchen. Dabei können sowohl die Bestände der eigenen Bibliothek durchsucht werden als auch die Bestände fremder Bibliotheken, wenn zur Demonstration für die fremden Bibliotheken Datenbestände angelegt worden sind. Die Datenbestände fremder Bibliotheken können nicht manipuliert werden. Wird nach einer Zeitschrift gesucht, zeigt das Programm, welche Hefte und welche Bände zu der gesuchten Zeitschrift in der Bibliothek vorhanden sind.

Verwaltung des Personals und der Geräte: Das System bietet an, die Daten der Geräte und des Personals zu speichern, abzufragen, zu verändern und zu löschen

Gegenüber einem vollständigen System zur Bibliotheksverwaltung weist der zu erstellende Prototyp die folgenden Einschränkungen auf:

Keine automatische Bestellung von Büchern: Die automatische Bestellung von neuen Büchern wird nicht durch den Prototyp simuliert, da kein Modell eines Systems existiert, das Bestellungen aufnehmen oder weiterleiten könnte.

Einschränkungen bei den erfaßten Daten: Bei der elektronischen Übermittlung von Daten an das HBZ gibt es feste Regeln, nach denen die Bücher katalogisiert worden sein müssen. Diese Regeln schreiben vor, welche Informationen in welcher Form zu verschiedenen Arten von Büchern erfaßt werden. Jedem Datum, das zu einem Buch erfaßt werden kann, ist eine eindeutige Nummer zugeordnet (die Nummer 520 steht z.B. für den Verlag eines Buches).

Hierbei gibt es mehrere hundert verschiedene Nummern. Bei der Katalogisierung eines einzelnen Buches werden von diesen Nummern immer nur wenige benötigt.

Da eine Modellierung der Daten der Bücher mit mehreren hundert Attributen für exploratives Prototyping zu umfangreich wäre, werden nur einge typische Informationen zu den erfaßten Büchern berücksichtigt. Dies gilt analog für Zeitschriften.

3.4 Die Objectory-Modelle

Der Prototyp für das System zur Bibliotheksverwaltung wird auf der Basis des Objectory-Anforderungsmodells sowie einer zusätzlichen Beschreibung der Beziehungen zwischen den benötigten Datenobjekten erstellt. Hierfür wird bereits das Analysemodell verwendet. Das erstellte Anforderungsmodell und der erstellte Teil des Analysemodells werden in diesem Abschnitt vorgestellt.

3.4.1 Das Anforderungsmodell

Das Anforderungsmodell besteht im allgemeinen aus dem Anwendungsfallmodell, dem Problembereichsmodell sowie aus Schnittstellenbeschreibungen (siehe Abschnitt 2.1). Da Schnittstellenbeschreibungen bei dem hier betrachteten Fallbeispiel nicht berücksichtigt werden (siehe Abschnitt 3.1), besteht das Anforderungsmodell nur aus dem Anwendungsfallmodell und aus dem Problembereichsmodell.

Das Anwendungsfallmodell

Das Anwendungsfallmodell besteht aus 32 verschiedenen Anwendungsfällen und den vier Akteuren Bibliothekar, Verwaltungsangestellter, Systemverwalter und Bibliotheksbenutzer. Die Vererbungshierarchie der Akteure ist in der Abbildung 3.1 dargestellt.

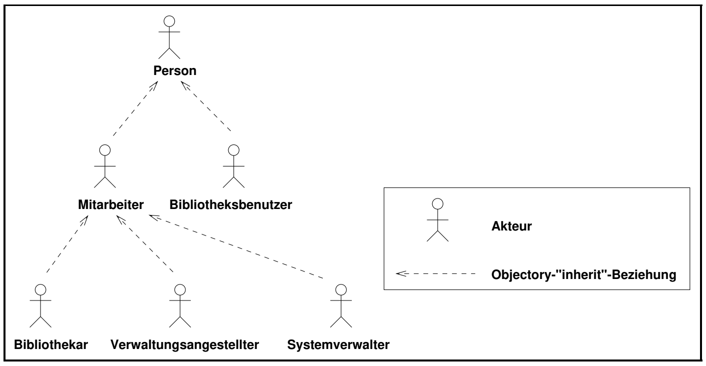

Abbildung 3.1: Vererbungshierarchie der Akteure.

Die Anwendungsfälle für das zu entwickelnde Bibliotheksverwaltungssystem sind in der Abbildung 3.2 aufgeführt.

Zwischen den Akteuren und den Anwendungsfällen sind in dieser Abbildung gerichtete Kanten eingetragen. Diese Kanten zeigen an, daß zwischen Instanzen der Akteure und Anwendungsfallinstanzen Stimuli ausgetauscht werden (zur Erläuterung des Stimuluskonzeptes siehe [JCJÖ93, S. 47]). In Objectory werden diese Beziehungen bereits als „communication“-Beziehungen [Ory94b, S. 10] bezeichnet. Hier zeigen sich leichte Abweichungen zwischen der Terminologie, die in der Objectory-Dokumentation benutzt wird, und der Terminologie, die Jacobson et al. verwenden. Im Gegensatz zu den Beschreibungen in der Objectory-Dokumentation benutzen Jacobson et al. den Begriff „communication“-Beziehung nur für die Kommunikation zwischen Objekten [JCJÖ93, S. 190], aber nicht für die Kommunikation zwischen Anwendungsfallinstanzen und Instanzen von Akteuren.

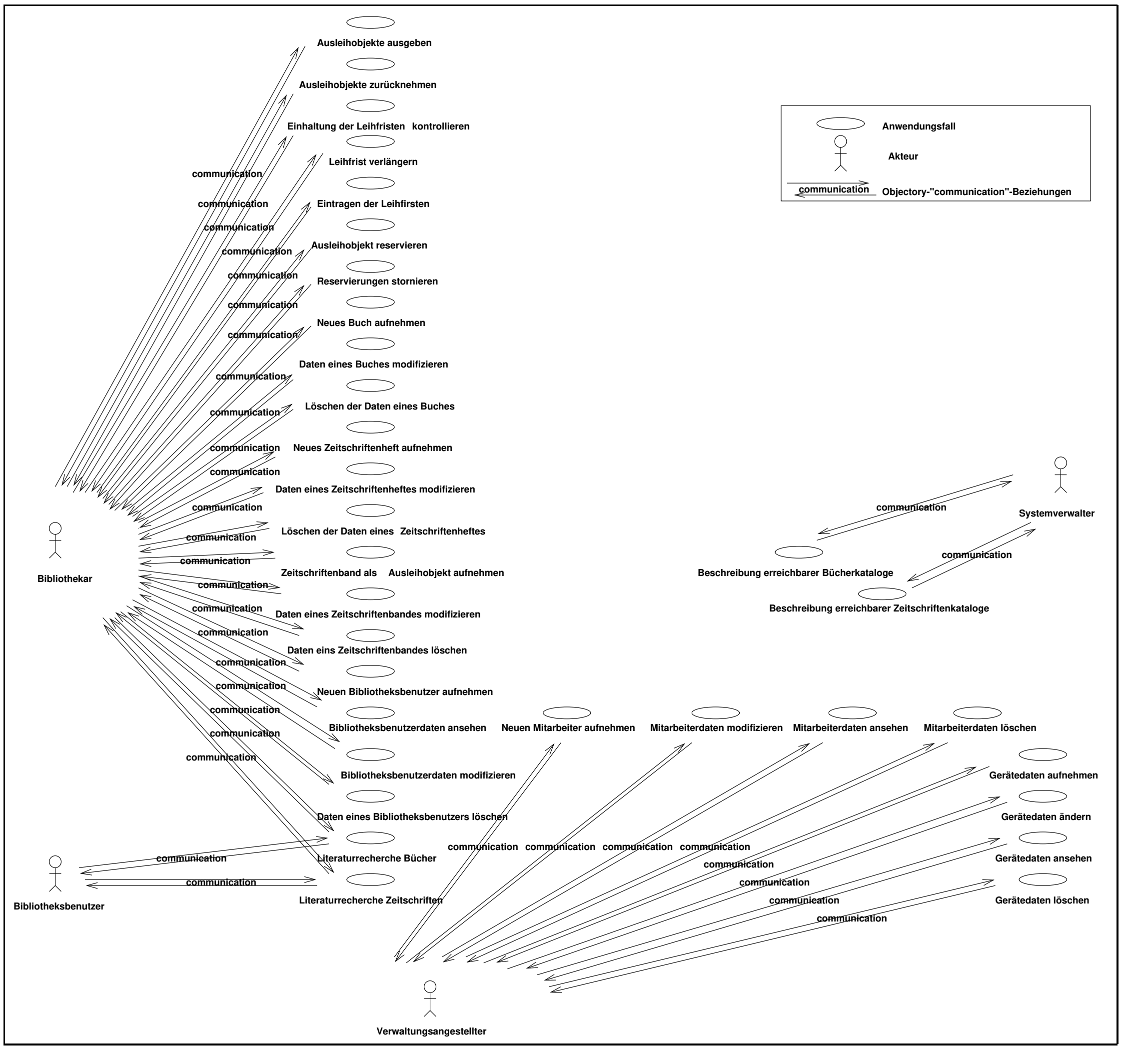

Abbildung 3.2: Anwendungsfälle und Akteure.

In der Diplomarbeit wird für die in der Abbildung 3.2 dargestellten Beziehungen die Bezeichnung „communication"-Beziehung verwendet, wie es in Objectory vorgesehen ist. Eine Kante, die bei einem Akteur beginnt und bei einem Anwendungsfall endet, zeigt an, daß eine Instanz des Akteurs Stimuli an eine Instanz des Anwendungsfalls senden kann, um dort die Ausführung von Operationen zu veranlassen (z.B. zum Start einer Anwendungsfallinstanz). Eine Kante, die bei einem Anwendungsfall beginnt und bei einem Akteur endet, zeigt an, daß eine Instanz des Anwendungsfalls Stimuli an eine Instanz des Akteurs senden kann (z.B. zur Ausgabe einer Fehlermeldung). Zusätzlich kann zu einer „communication"-Beziehung noch eine *Kardinalität* angegeben werden. Bei einer „communication"-Beziehung, die bei einem Akteur beginnt und bei einem Anwendungsfall endet, gibt die Kardinalität an, mit wievielen Instanzen des betroffenen Anwendungsfalls eine Instanz des Akteurs gleichzeitig kommunizieren kann. Bei einer „communication"-Beziehung, die bei einem Anwendungsfall beginnt und bei einem Akteur endet, gibt die Kardinalität an, an wieviele Instanzen dieses Akteurs Stimuli gesendet werden, wenn eine Instanz dieses Anwendungsfalls ausgeführt wird. In dieser Diplomarbeit wird die Kardinalität 1 für alle dargestellten „communication"-Beziehungen, die zwischen Akteuren und Anwendungsfällen modelliert sind, vorausgesetzt, sofern in dem Textteil, in dem eine Kommunikationsbeziehung erläutert wird, keine anderen Angaben hierzu gemacht werden.

Im folgenden ist exemplarisch die Beschreibung des Anwendungsfalls Neues Buch aufnehmen aufgeführt. Das komplette Anwendungsfallmodell mit den vollständigen Beschreibungen sämtlicher Anwendungsfälle befindet sich aus Platzgründen im Zusatzdokument zu dieser Diplomarbeit. Eine kurze Übersicht über alle Anwendungsfälle wird in Tabelle 3.1 gegeben.

Der Anwendungsfall Neues Buch aufnehmen

Ereignisfluß

Dieser Anwendungsfall ermöglicht die Aufnahme der Daten eines neu angeschafften Buches.

Auswahl der Art der Katalogisierung

Der Bibliothekar gibt ein, ob zunächst nach Büchern gesucht werden soll, um Daten zu übernehmen oder ob die Katalogisierung des neuen Buches durchgeführt werden soll, ohne alte Daten zu übernehmen. Alternativ kann er die Ausführung dieses Anwendungsfalls vorzeitig abbrechen. Soll nach Büchern gesucht werden, um Daten zu übernehmen, wird eine Liste der Kataloge ausgegeben, in denen nach Daten gesucht werden kann. Der Bibliothekar wählt aus, in welchen Katalogen gesucht werden soll.

Suche nach Büchern

Wenn ausgewählt wurde, daß bereits gespeicherte Daten übernommen werden sollen, werden die Suchkriterien, nach denen Bücher gesucht werden sollen, eingegeben. Zur Suche können

- Titel,
- Verfasser,
- Verlag,
- Stichworte,
- Erscheinungsjahr,
- ISBN

angegeben werden. Alternativ kann auf die Suche verzichtet oder die Katalogisierung vorzeitig abgebrochen werden. Durch die Auswahl des Befehls *Zurücksetzen* werden alle Eingaben gelöscht. Die Suchkriterien können dann noch einmal neu eingegeben werden.

Anzeige der Bücher, die die Suchkriterien erfüllen und Übernahme von Daten

Die Anzeige von Büchern wird nur dann durchgeführt, wenn vor der Katalogisierung eines Buches zunächst alle erreichbaren Bücherkataloge durchsucht werden sollen. Der Bibliothekar kann

- sich die Daten der gefundenen Bücher anzeigen lassen,
- die Daten eines gefundenen Buches für seine eigene Katalogisierung übernehmen,
- die Anzeige abbrechen, ohne Daten für seine Katalogisierung zu übernehmen oder
- den gesamten Katalogisierungsvorgang abbrechen.

Wenn die Anzeige der Bücherdaten beendet und mit der Katalogisierung begonnen werden soll oder wenn die Katalogisierung vollständig abgebrochen werden soll, wird vorher zunächst noch eine Abfrage durchgeführt, bei der der Bibliothekar seine Auswahl bestätigen muß.

Katalogisierung

Die Daten des Buches, das katalogisiert werden soll, werden eingegeben. Jeder möglichen Datenart (z.B. Autor, Titel) ist eine Zahl eindeutig zugeordnet. Diese Zahl wird zuerst eingegeben, um die Art des einzugebenden Datums auszuwählen. Anschließend wird der Wert des Datums eingegeben. Wurden für die Katalogisierung bereits Daten aus einem der erreichbaren Bücherkataloge übernommen, müssen nur die Daten eingegeben werden, die von den übernommen Daten abweichen. Ist die Katalogisierung abgeschlossen, wählt der Bibliothekar den Befehl *Daten übernehmen*. Alternativ kann er den Befehl *Abbrechen* wählen und damit auf die Aufnahme der neuen Daten verzichten und die Ausführung dieses Anwendungsfalls vorzeitig beenden. Wird *Abbrechen* gewählt, muß der Bibliothekar seine Entscheidung in einer zusätzlichen Abfrage noch einmal bestätigen.

Festlegen der Leihfristen

Es ist möglich, verschiedene Leihfristen für verschiedene Arten von Büchern festzulegen. Zu jeder Benutzergruppe werden alle erlaubten Leihfristen angezeigt. Der Bibliothekar wählt die Leihfristen und mögliche Verlängerungen der Leihfristen für das katalogisierte Buch aus.

Überprüfung der eingegebenen Daten

Die Daten, die dem neuen Buch zugeordnet wurden, werden angezeigt. Der Bibliothekar kann

- noch einmal zur Stelle *Katalogisierung* zurückkehren, um einzelne Daten zu verändern,
- die Daten in den Bücherkatalog übernehmen und die Katalogisierung beenden,
- die Daten nicht übernehmen und die Katalogisierung beenden.

Wenn die Katalogisierung beendet werden soll, ohne die eingegebenen Daten zu übernehmen, wird noch eine Abfrage durchgeführt, bei der der Bibliothekar seine Auswahl bestätigen muß.

Vergabe einer Ausleihnummer

Für das katalogisierte Buch wird vom Programm eine Ausleihnummer festgelegt. Die Ausleihnummer wird ausgegeben.

Ausnahmen

Unvollständige Eingabe

Ein Buch wird nur dann in den Bücherkatalog aufgenommen, wenn ihm

- mindestens ein Verfasser,
- ein Titel und
- das Erscheinungsjahr

zugeordnet sind. Wurde versucht, ein Buch aufzunehmen, bei dem ein Teil dieser Angaben fehlt, wird eine Fehlermeldung ausgegeben. Das Programm springt anschließend zur Stelle *Katalogisierung* zurück.

Kein Zugriffsrecht für die Datenbank

Wenn der Anwender des Programms kein Zugriffsrecht für die Datenbank besitzt, die den Bücherkatalog enthält, wird eine Fehlermeldung ausgegeben. Dieser Anwendungsfall kann dann nicht ausgeführt werden.

Die Suche nach Büchern (Stelle Suche nach Büchern) ist auch im Anwendungsfall Literaturrecherche Bücher enthalten. Es wäre deshalb möglich gewesen, einen unabhängigen Anwendungsfall Suche nach Büchern, der von den Anwendungsfällen Neues Buch aufnehmen und Literaturrecherche Bücher gemeinsam genutzt wird, zu erzeugen.

Objectory bietet sog. „*extends*"-Beziehungen und „*uses*"-Beziehungen an, um Beziehungen zwischen Anwendungsfällen zu beschreiben. Eine detaillierte Beschreibung dieser Beziehungsarten, ihrer Unterschiede und eine Gegenüberstellung der Vor- und Nachteile, die sich durch die Nutzung dieser Beziehungen ergeben, wird bei Jacobson et al. [JCJÖ93, S. 163ff., S. 170ff.] gegeben. Gemäß der Empfehlung von Jacobson et al. wird bei dem vorgestellten Beispiel die Erstellung eines umfassenden Anwendungsfalls dem Entwurf mehrerer kleiner Anwendungsfälle vogezogen [JCJÖ93, S. 174].

Anwendungsfall	Beschreibung
Ausleihobjekt reservieren	Der Bibliothekar reserviert Ausleihobjekte für einen Bibliotheksbenutzer.
Ausleihobjekte ausgeben	Ausleihobjekte werden an einen Bibliotheksbenutzer verliehen. Daten der verliehenen Ausleihobjekte, Daten des Entleihers und die Leihfristen werden gespeichert.
Ausleihobjekte zurücknehmen	Die Rückgabe von Ausleihobjekten wird vermerkt.
Beschreibung erreichbarer Bücherkataloge	Der Systemverwalter legt fest, auf welche externen Datenbanken, die Bücherkataloge enthalten, lesend zugegriffen werden kann.
Beschreibung erreichbarer Zeitschriftenkataloge	Der Systemverwalter legt fest, auf welche externen Datenbanken die Zeitschriftenkataloge enthalten, lesend zugegriffen werden kann.
Bibliotheksbenutzerdaten ansehen	Die zu einem Bibliotheksbenutzer gespeicherten Daten werden angezeigt.
Bibliotheksbenutzerdaten modifizieren	Die zu einem Bibliotheksbenutzer gespeicherten Daten können geändert werden. Die Ausleihberechtigung kann entzogen oder zurückgegeben werden.
Daten eines Bibliotheksbenutzers löschen	Die zu einem Bibliotheksbenutzer gespeicherten Daten werden gelöscht.
Daten eines Buches modifizieren	Die Daten, die zu einem Buch gespeichert sind, können geändert werden.
Daten eines Zeitschriftenbandes löschen	Der Bibliothekar entfernt einen Zeitschriftenband aus dem Bestand der Ausleihobjekte der Bibliothek. Die Ausleihnummer dieses Zeitschriftenbandes wird aus dem Datenbestand gelöscht.
Daten eines Zeitschriftenbandes modifizieren	Der Bibliothekar kann die Leihfristen, die für einen Zeitschriftenband eingetragen sind, oder die Signatur eines Zeitschriftenbandes modifizieren.
Daten eines Zeitschriftenheftes modifizieren	Die zu einem Zeitschriftenheft gespeicherten Daten können geändert werden. Neben den spezifischen Daten zu einem Zeitschriftenheft können auch die allgemeinen Daten der zugehörigen Zeitschrift modifiziert werden.
Einhaltung der Leihfristen kontrollieren	Das Programm gibt aus, welche Benutzer eine Leihfrist überschritten haben. Zusätzlich wird die Anzahl der Tage ausgegeben, die die einzelnen Leihfristen überschritten wurden.
Eintragen der Leihfristen	Der Bibliothekar trägt die möglichen Leihfristen ein.
Gerätedaten ansehen	Die Daten der in der Biblitohek vorhandenen Geräte werden angezeigt.
Gerätedaten aufnehmen	Daten über neu angeschaffte Geräte werden eingegeben.
Gerätedaten löschen	Die zu einem Gerät gespeicherten Daten werden gelöscht.
Gerätedaten ändern	Die gespeicherten Gerätedaten können geändert werden.
Leihfrist verlängern	Die Leihfrist für ein Ausleihobjekt wird verlängert.
Literaturrecherche Bücher	Der Bibliothekar oder ein Bibliotheksbenutzer sucht nach Büchern.
Literaturrecherche Zeitschriften	Der Bibliothekar oder ein Bibliotheksbenutzer sucht nach einer Zeitschrift, nach einem Zeitschriftenheft oder nach einem Zeitschriftenband.
Löschen der Daten eines Buches	Die Daten eines Buches werden aus dem Bücherkatalog entfernt.
Löschen der Daten eines Zeitschriftenbandes	Daten eines Zeitschriftenbandes werden gelöscht.
Löschen der Daten eines Zeitschriftenheftes	Daten eines Zeitschriftenheftes werden gelöscht.
Mitarbeiterdaten ansehen	Daten, die zu den Mitarbeitern der Bibliothek gespeichert sind, werden angezeigt.
Mitarbeiterdaten löschen	Die Daten eines Mitarbeiters werden aus der Personaldatenbank gelöscht.
Mitarbeiterdaten modifizieren	Der Verwaltungsangestellte ändert Daten, die zu einem Mitarbeiter gespeichert sind.
Neuen Bibliotheksbenutzer aufnehmen	Daten eines neuen Bibliotheksbenutzers werden gespeichert.
Neuen Mitarbeiter aufnehmen	Die Daten eines Mitarbeiters werden gespeichert.
Neues Buch aufnehmen	Die Daten eines neu angeschafften Buches können gespeichert werden.
Neues Zeitschriftenheft aufnehmen	Daten eines Zeitschriftenheftes werden in den Zeitschriftenkatalog aufgenommen. Wenn es sich bei dem Zeitschriftenheft um das erste Heft einer Zeitschrift handelt, werden auch die Daten der Zeitschrift eingegeben.
Reservierungen stornieren	Reservierungen von Ausleihobjekten werden storniert.
Zeitschriftenband als Ausleihobjekt aufnehmen	Der Bibliothekar nimmt einen Zeitschriftenband in den Bestand der Ausleihobjekte der Bibliothek auf.

Tabelle 3.1: Beschreibung der Anwendungsfälle.

Das Problembereichsmodell

Bei der Identifikation von Objekten für das Problembereichsmodell werden zunächst nur die Materialien des Anwendungsbereichs erfaßt. Bei den Materialien kann es sich sowohl um konkrete als auch um abstrakte Dinge, wie z.B. Leihfristen, handeln (siehe Abschnitt 2.2). Werkzeuge sind nicht aufgeführt. Dies liegt daran, daß bei der Ist-Analyse nur ein allgemeiner Überblick über den Anwendungsbereich gegeben wird, während auf eine detaillierte Beschreibung einzelner Szenarien (siehe Abschnitt 2.2) verzichtet wird. Bei einer detaillierten Beschreibung von Szenarien, hätten auch benutzte Werkzeuge identifiziert werden können. Eine solche ausführliche Ist-Analyse war aus Zeitgründen im Rahmen der Diplomarbeit jedoch nicht möglich. Da nach Gryczan und Züllighoven [GZ92, S. 269] die Werkzeuge zwar „einen guten Überblick über die vorhandenen Arbeitszusammenhänge" geben, jedoch nur „selten unmittelbar in ein DV-Modell übernommen werden" können, sollte das Fehlen der Werkzeuge im Problembereichsmodell keinen Einfluß auf die weitere prinzipielle Vorgehensweise bei der Systementwicklung haben.

Zusätzlich werden die handelnden Personen als Objekte erfaßt. Obwohl Personen bei einer strengen Auslegung des Materialbegriffs gemäß Bäumer et al. (siehe Abschnitt 2.2) keine Materialien sind, werden auch diese Objekte bei der weiteren Beschreibung dieses Fallbeispiels als Materialobjekte angesehen. Der Grund für die Modellierung der handelnden Personen als Objekte wird im folgenden dargestellt:

> Die Metaphern Werkzeug und Material werden bei der Entwicklung von Systemen benutzt, die zur Unterstützung von vorher manuell durchgeführten Arbeitsvorgängen gedacht sind. Bei dem betrachteten Fallbeispiel soll jedoch ein System entworfen werden, mit dem auch Arbeitsvorgänge unterstützt werden, die in der Bereichsbibliothek nicht durchgeführt werden (siehe Abschnitt 3.3). Insbesondere für die Verwaltung der Daten von Mitarbeitern und Bibliotheksbenutzern können daher keine Materialien gefunden werden. Da diese Daten jedoch benötigt werden, werden die Personen direkt als Objekte in das Problembereichsmodell aufgenommen. Auch durch eine Orientierung am Ist-Zustand der Zentralbibliothek können die fehlenden Materialien nicht gewonnen werden, weil hier bereits computergestützte Systeme eingesetzt werden, die aber nicht auf den Metaphern Werkzeug und Material basieren. Dies soll am Beispiel der Bibliotheksbenutzer verdeutlicht werden:
>
> **Beispiel:** Ein typisches Material, das bearbeitet wird, um Bibliotheksbenutzerdaten aufzunehmen, wäre ein Aufnahmebogen. In der Bereichsbibliothek Informatik sind jedoch weder ein Aufnahmebogen noch andere Materialien vorhanden, die bearbeitet werden, um Daten von Bibliotheksbenutzern aufzunehmen. Dies liegt daran, daß Daten von Bibliotheksbenutzern in der Zentralbibliothek gespeichert werden.
>
> Objekte die für die Aufnahme von Bibliotheksbenutzerdaten relevant sind, werden bei diesem Fallbeispiel deshalb aus dem Problembereich der Zentralbibliothek entnommen. Hier werden die Daten der Benutzer der Bibliothek mit Hilfe eines computergestützten Systems erfaßt und gespeichert. Durch den im Rahmen der Diplomarbeit zu erstellenden Prototyp wird dieses System simuliert. Das System zur Erfassung und Speicherung der Benutzerdaten in der Zentralbibliothek wurde jedoch nicht auf der Basis der Metaphern Werkzeug und Material implementiert.

Objectory ermöglicht die Erstellung von unterschiedlich detaillierten Problembereichsmodellen (siehe Abschnitt 2.1). Bei dem hier betrachteten Fallbeispiel wird ein sehr ausführliches Problembereichsmodell erstellt. Dabei werden zusätzlich zur Beschreibung der Objekte des Problembereichs die Beziehungen dieser Objekte zueinander separat für jeden modellierten Anwendungsfall (siehe Abschnitt 3.4.1) beschrieben. Diese ausführliche Variante des Problembereichsmodells wurde gewählt, weil der zu erstellende Prototyp in einer sehr frühen Phase des Entwicklungsprozesses erstellt wird, in der das Analysemodell nur teilweise und nachfolgende Modelle noch nicht erstellt sind. Durch das ausführliche Problembereichsmodell soll das Fehlen der nicht erstellten Modelle zum Teil kompensiert werden. Die Beschreibung der Beziehungen zwischen den für die einzelnen Anwendungsfälle identifizierten Objekten im Problembereichsmodell soll Zusammenhänge zwischen diesen Objekten verdeutlichen. Über den im Abschnitt 2.2 vorgestellten fachlichen Klassenentwurf geht das erstellte Problembereichsmodell weit hinaus.

Das vollständige Problembereichsmodell befindet sich im Zusatzdokument, das zu dieser Diplomarbeit erstellt wurde. Im folgenden wird aus Platzgründen nur der Teil des Problembereichsmodells, der zum Anwendungfall Neues Buch aufnehmen (siehe Abschnitt 3.4.1) erstellt wurde, detailliert beschrieben. Die Beziehungen zwischen Objekten aus dem Problembereich dieses Anwendungsfalls sind in der Abbildung 3.3 skizziert. Die Namen der hier verwendeten Objekte sollten die Bedeutung dieser Objekte bereits erklären. Eine kurze zusätzliche Beschreibung zu den Objekten wird in der Tabelle 3.2 gegeben.

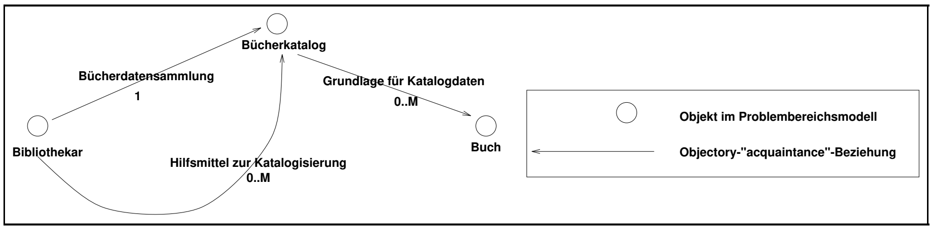

Abbildung 3.3: Beziehungen zwischen den Objekten aus dem Problembereich des Anwendungsfalls Neues Buch aufnehmen.
Vererbungsbeziehungen und Objekte vererbender Klassen sind in dieser Abbildung nicht berücksichtigt (die Klasse des Objektes Buch erbt z.B. von der Klasse eines Objektes Ausleihobjekt, zur Beschreibung des Objektes Ausleihobjekt siehe Tabelle 3.2). Die Vererbungshierarchien des Problembereichsmodells sind in der vollständigen Beschreibung des Anforderungsmodells des hier betrachteten Fallbeispiels im Zusatzdokument zur Diplomarbeit enthalten.

Bei den in der Abbildung 3.3 dargestellten Beziehungen handelt es sich um Objectory-„*acquaintance*“-Beziehungen. „Acquaintance“-Beziehungen sind statische Beziehungen zwischen Instanzen, die beschreiben, daß ein Objekt, von dem eine Kante ausgeht, Kenntnis von dem Objekt, auf das die Kante zeigt, besitzt [JCJÖ93, S. 178]. Die Beschriftung einer Kante gibt an, welche Rolle das Objekt, auf das die gerichtete Kante zeigt, in Bezug auf das Objekt, von dem die Kante ausgeht, spielt. Diese Art der Kantenbeschriftung wird von Jacobson et al. anstelle einer Beschriftung der Kanten mit Verben empfohlen [JCJÖ93, S. 179ff.]. Durch die Angabe einer Kardinalität wird beschrieben, wieviele Instanzen dem Objekt, von dem die Kante ausgeht, bekannt sind.

Im folgenden wird die Skizze aus Abbildung 3.3 anhand der dargestellten Beziehungen erläutert:

1-Beziehung Bücherdatensammlung: Diese Kante beschreibt eine Instanzbeziehung zwischen dem Objekt Bibliothekar und dem Objekt Bücherkatalog. Im Bücherkatalog werden alle neuen Bücher aufgenommen. Dabei wird davon ausgegangen, daß in einer Bibliothek nur ein Bücherkatalog existiert.

0..M-Beziehung Hilfsmittel zur Katalogisierung: Die Daten bereits katalogisierter Bücher aus dem eigenen Bücherkatalog und aus Bücherkatalogen fremder Bibliotheken können bei Aufnahme neuer Bücher übernommen werden, um die Katalogisierung zu erleichtern. Es ist dem Bibliothekar freigestellt, ob er Bücherkataloge für die Übernahme von Daten nutzt.

0..M-Beziehung Grundlage für Katalogdaten: Ein Bücherkatalog enthält Daten von Büchern.

Eine Übersicht über alle im Problembereichsmodell verwendeten Objekte wird in der Tabelle 3.2 gegeben. In dieser Tabelle sind die Objekte Benutzerdatenbank, Personaldatenbank und Gerätedatenbank aufgeführt. Diese Datenbanken sind in der Bereichsbibliothek Informatik nicht vorhanden, weil die Verwaltung von Daten Geräten, Mitarbeiten und Benutzern in der Zentralbibliothek, und nicht in der Bereichsbibliothek, durchgeführt wird. Da das zu modellierende Bibliotheksverwaltungssystem in diesem Bereich über die Anforderungen der Bereichsbibliothek hinausgeht, wurden die Objekte Benutzerdatenbank, Personaldatenbank und Gerätedatenbank aus dem Problembereich der Zentralbibliothek übernommen. Dies gilt analog für das Objekt Verwaltungsangestellter.

Objekt/Klasse	Beschreibung
Ausleihkarte	Entleihungen von Ausleihobjekten werden manuell mit Hilfe von Karteikarten verwaltet. Diese Karteikarten werden im folgenden als Ausleihkarten bezeichnet. Mit einer Ausleihkarte werden zu jedem entliehenen Ausleihobjekt die Daten des Ausleihobjektes und des Entleihers, sowie das Datum der Entleihung erfaßt.
Ausleihkartei	Mit Hilfe der Ausleihkartei werden verliehene Ausleihobjekte und Entleiher verwaltet.
Ausleihobjekt	Ein Ausleihobjekt ist ein Gegenstand, der an Benutzer der Bibliothek verliehen wird.
Benutzerdatenbank	In der Benutzerdatenbank sind Daten der Bibliotheksbenutzer gesammelt.
Bibliothekar	Der Bibliothekar ist für die Verwaltung der Ausleihobjekte der Bibliothek und der Daten der Bibliotheksbenutzer zuständig.
Bibliotheksbenutzer	Der Bibliotheksbenutzer führt mit dem Bibliotheksverwaltungssystem Literaturrecherchen durch. Er entleiht Ausleihobjekte aus der Bibliothek und muß sie innerhalb der Leihfrist zurückgeben.
Buch	Die Bücher bilden eine Teilmenge der Menge der Ausleihobjekte der Bibliothek.
Bücherkatalog	In einem Bücherkatalog sind Daten von Büchern erfaßt.
Gerät	Die Geräte sind die Inventargegenstände der Bibliothek, die nicht verliehen werden.
Gerätedatenbank	In der Gerätedatenbank sind die Daten der in der Bibliothek vorhandenen Geräte gespeichert.
Inventargegenstand	Alle Gegenstände, die zum Inventar der Bibliothek gehören, werden hier unter dem Begriff Inventargegenstand zusammengefaßt.
Katalog	In einem Katalog sind Daten von Ausleihobjekten erfaßt.
Leihfrist	Die Leihfrist ist der Zeitraum, in dem ein entliehenes Ausleihobjekt vom Bibiliotheksbenutzer an die Bibliothek zurückgegeben werden muß.
Mitarbeiter	Ein Mitarbeiter ist eine Person, die in der Bibliothek arbeitet.
Person	Eine Person arbeitet in der Bibliothek oder nutzt die Dienstleistungen der Bibliothek.
Personaldatenbank	In der Personaldatenbank sind die Daten der Mitarbeiter der Bibliothek gespeichert.
Systemverwalter	Der Systemverwalter sorgt für die Einrichtung und Wartung der eingesetzten Soft- und Hardware.
Verwaltungsangestellter	Ein Verwaltungsangestellter verwaltet die Daten von Mitarbeitern, und Geräten der Bibliothek.
Zeitschrift	Eine Zeitschrift bezeichnet eine Menge aller Zeitschriftenhefte mit demselben Titel, die zu dem Themenbereich der Zeitschrift herausgegeben werden. Jeder Zeitschrift ist eindeutig eine ISSN (siehe Glossar im Zusatzdokument) zugeordnet.
Zeitschriftenband	Ein Zeitschriftenband enthält den Inhalt mehrerer verschiedener Zeitschriftenhefte einer Zeitschrift. Die Menge der Zeitschriftenbände ist eine Teilmenge der Menge der Ausleihobjekte. Ein Zeitschriftenband kann einer Zeitschrift eindeutig zugeordnet werden.
Zeitschriftenheft	Die Menge der Zeitschriftenhefte ist eine Teilmenge der Menge der Ausleihobjekte einer Bibliothek. Es enthält mehrere Artikel und kann einer Zeitschrift eindeutig zugeordnet werden.
Zeitschriftenkatalog	In einem Zeitschriftenkatalog sind Daten von Zeitschriften und den zugehörigen Zeitschriftenheften und Zeitschriftenbänden erfaßt.

Tabelle 3.2: Übersicht über Objekte bzw. Klassen des Problembereichsmodells.

3.4.2 Das Analysemodell

Der zu erstellende Prototyp dient als Ergänzung des Objectory-Anforderungsmodells. Wegen des großen Umfangs dieses Fallbeispiels wird zusätzlich zu den Modellen des Anforderungsmodells noch ein Teil des Analysemodells erstellt. Hier werden alle Objekte berücksichtigt, die zur Speicherung von Daten dienen. Diese Objekte werden als *Entitätsobjekte* bezeichnet [JCJÖ93, S. 184]. Diese Objekte werden weitgehend aus dem Problembereichsmodell übernommen. An einigen Stellen wird jedoch vom Problembereichsmodell abgewichen. So befinden sich im Analysemodell Objekte, die nicht im Problembereichsmodell aufgeführt sind. Gründe für die Abweichungen vom Problembereichsmodell werden im Verlauf dieses Abschnitts noch erläutert.

Analog zum Abschnitt 3.4.1 werden auch bei der Beschreibung des erstellten Teil des Analysemodells zunächst exemplarisch die Objekte, die zum Anwendungsfall Neues Buch aufnehmen gehören, beschrieben. Diese Objekte sind in den Abbildungen 3.4 und 3.5 dargestellt. Das vollständige Modell befindet sich im Zusatzdokument zur Diplomarbeit.

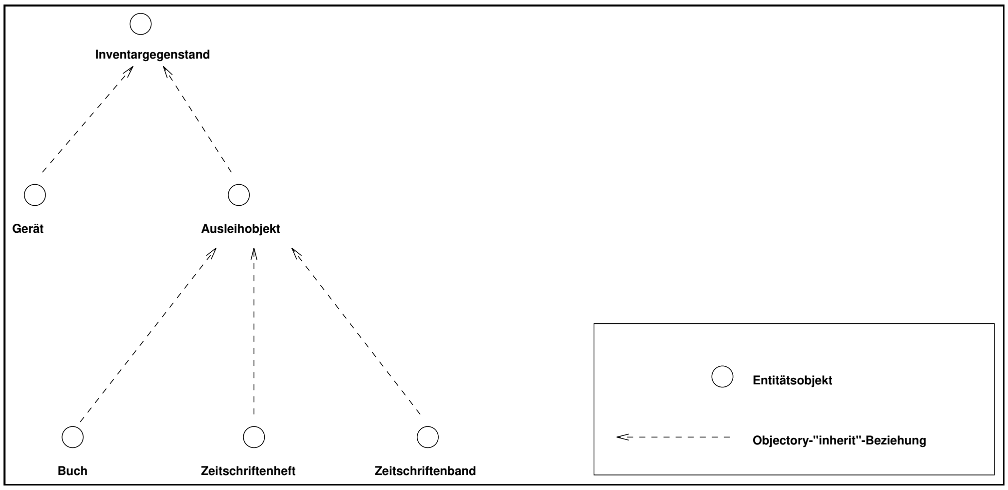

Abbildung 3.4: Ein Teil der Vererbungsbeziehungen im Analysemodell.

In der Abbildung 3.4 wird eine Vererbungshierarchie dargestellt. Für den Anwendungsfall Neues Buch aufnehmen ist bei dieser Abbildung vor allem die Klasse des Objektes Buch wichtig. Zusätzlich müssen jedoch auch die vererbenden Klassen der Objekte Ausleihobjekt und Inventargegenstand beachtet werden. Die Klassen der Objekte Gerät, Zeitschriftenband und Zeitschriftenheft sind im Zusammenhang mit dem hier betrachteten Anwendungsfall unwichtig. Objekte erbender Klassen besitzen immer alle Attribute und Verweise auf andere Objekte, die bereits für die Objekte der Vorgängerklassen definiert wurden (Attribute werden von Objekten zur Speicherung von Informationen benutzt, siehe Abschnitt 3.1).

Die Verweise und Attribute der Objekte Buch, Ausleihobjekt und Inventargegenstand sind in der Abbildung 3.5 skizziert. Die in dieser Abbildung aufgeführten Objekte werden im folgenden vorgestellt. Dabei wird insbesondere die Modellierung des Objektes Leihfristenkombination motiviert.

Das Objekt Inventargegenstand: Zu jedem Gegenstand, der zum Inventar der Bibliothek gehört, werden eine eindeutige Inventarnummer und das Inventarisierungsdatum gespeichert.

Die Objekte Ausleihobjekt und Leihfristenkombination: Die Menge der Ausleihobjekte ist eine Teilmenge der Menge des gesamten Inventars der Bibliothek. Sie umfaßt alle Gegenstände, die verliehen werden. Die Attribute des Objektes Inventargegenstand werden geerbt. Zusätzlich werden Ausleihobjekte üblicherweise mit einer bibliotheksinternen Nummer, der *Signatur*, versehen. Ausleihobjekte dürfen nur eine begrenzte Zeit ausgeliehen werden. Diese Zeit wird als *Leihfrist* bezeichnet. Das zu erstellende System ermöglicht zusätzlich eine Verlängerung der Leihfrist. Dabei läge es nahe, die Leihfristen und die Verlängerungen von Leihfristen

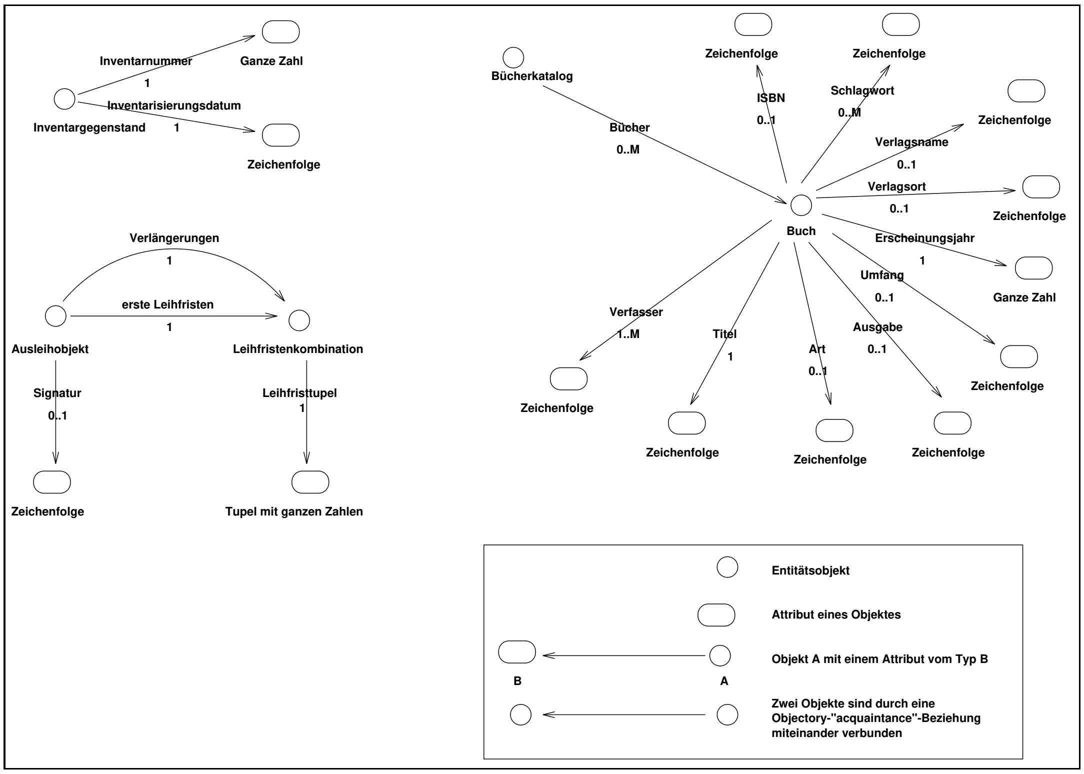

Abbildung 3.5: Analysemodellobjekte zum Anwendungsfall Neues Buch aufnehmen.
Zur Motivation für die Einführung des Objektes Leihfristenkombination siehe Beschreibung der Objekte Ausleihobjekt und Leihfristenkombination.

als Attribute zum Objekt Ausleihobjekt zu speichern, wie es Abbildung 3.6 skizziert ist. Die Leihfristen und die Verlängerungen werden nach der Darstellung in der Abbildung 3.6 als Tupel mit ganzen Zahlen gespeichert. Für jede Benutzergruppe der Bibliothek existiert ein Eintrag im Tupel, in dem die Anzahl der Tage angegeben ist, die ein Ausleihobjekt von Angehörigen dieser Gruppe entliehen werden darf. Bei einer Modellierung gemäß Abbildung 3.6 wird jedoch nicht berücksichtigt, daß Leihfristen und Verlängerungen nicht für jedes Ausleihobjekt individuell festgelegt werden. Ausleihobjekte werden in Gruppen eingeteilt (siehe auch Abschnitt 3.2). Probleme, die bei der individuellen Festlegung von Leihfristen für jedes einzelne Ausleihobjekt entstehen können, sollen anhand eines Beispiels verdeutlicht werden:

Beispiel: Grundlage dieses Beispiels ist die Festlegung der Leihfristen, wie sie derzeit in der Bereichsbibliothek Informatik der Universität Dortmund (siehe Abschnitt 3.2) zu finden ist. Es gibt drei Benutzergruppen: die Mitarbeiter im Fachbereich Informatik, die Mitarbeiter anderer Fachbereiche und Studenten. Verlängerungen von Leihfristen sind in der Bereichsbibliothek nicht vorgesehen. Ein Tupel für Verlängerungen bestünde daher ausschließlich aus Nullen und wird in diesem Beispiel nicht weiter betrachtet. Einträge in dem Tupel, in dem die Leihfristen zu einem Ausleihobjekt gespeichert sind, könnten die folgenden Bedeutungen haben:

erster Eintrag: Leihfrist für die Mitarbeiter im Fachbereich Informatik.

zweiter Eintrag: Leihfrist für die Mitarbeiter anderer Fachbereiche.

dritter Eintrag: Leihfrist für Studenten.

Bei allen Büchern (ein Buch ist ein Ausleihobjekt, siehe Abbildung 3.4), die verliehen werden dürfen, bestünde der zweite Eintrag also aus der Zahl 28, weil Mitarbeiter fremder Fachbereiche diese Bücher maximal vier Wochen lang entleihen dürfen. Wenn die Fristen für die Benutzergruppe der Mitarbeiter anderer Fachbereiche modifiziert werden sollen, müßte der Eintrag für die Leihfristen bei jedem Buch, das in der Bibliothek vorhanden ist und verliehen werden darf, geändert werden. In einer Bibliothek wären von dieser Änderung sehr viele Daten betroffen.

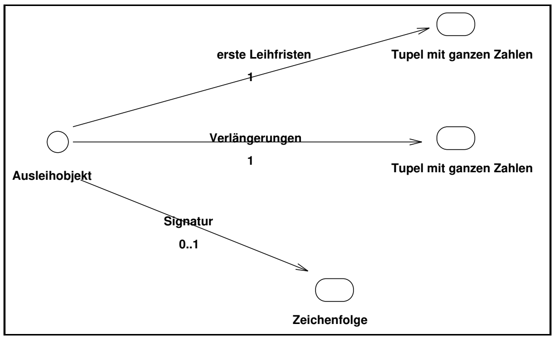

Abbildung 3.6: Nicht realisierter Ansatz zur Modellierung eines Ausleihobjektes.

Um diese Probleme zu umgehen, wird für jede mögliche Kombination von Leihfristen und für jede mögliche Kombination von Leihfristverlängerungen genau ein Objekt Leihfristenkombination eingeführt (siehe Abbildung 3.5). Ein Objekt Leihfristenkombination besitzt genau ein Attribut, das so aufgebaut ist, wie es für die Attribute für die Leihfristen bzw. für die Fristverlängerungen aus Abbildung 3.6 bereits beschrieben wurde. Die Ausleihobjekte besitzen jeweils zwei Referenzen auf Objekte Leihfristenkombination. Eine Referenz zeigt auf das Objekt, in dem die Leihfristen für alle möglichen Entleihergruppen gespeichert sind. Die zweite Referenz zeigt auf das Objekt, in dem die möglichen Verlängerungen der Leihfristen für alle möglichen Entleihergruppen gespeichert sind. Bei einer Änderung von Leihfristen oder von erlaubten Verlängerungen der Leihfristen muß jetzt nur noch das Attribut des betroffenen Objektes Leihfristenkombination geändert werden. Die Ausleihobjekte werden nicht mehr manipuliert.

Das Objekt Buch: Ein Objekt der Klasse Buch besitzt mehrere Attribute. Die Attribute sind in der Abbildung 3.5 dargestellt. Die Bedeutung der einzelnen Attribute wird durch die Bezeichnungen der Pfeile beschrieben.

Bei dem dargestellten Modell kann zwischen Attributen, die in jedem Fall vorhanden sein müssen (1-Beziehungen und 1..M-Beziehungen) und Attributen, die weggelassen werden können (0..1-Beziehungen und 0..M-Beziehungen), unterschieden werden. Bei dieser Einteilung wurden die folgenden Entscheidungsgrundlagen angewendet:

1. Attribute, nach deren Inhalt nicht im Rahmen einer Literaturrecherche gesucht werden kann, sind immer frei wählbar. Dies betrifft die Attribute zur Ausgabe, zum Umfang, zur Art und zum Verlagsort des Buches.
2. Attribute, nach deren Inhalt im Rahmen einer Literaturrecherche gesucht werden kann, die aber nicht bei jedem real existierenden Buch vorhanden sein müssen, sind ebenfalls frei wählbar (die Bezeichnung real existierendes Buch soll den Unterschied zum Analysemodellobjekt Buch verdeutlichen).

   Dies betrifft die Attribute, in denen die ISBN und der Verlag gespeichert werden. Bei internen Veröffentlichungen eines Fachbereichs einer Universität wären z.B. keine Daten für diese Attribute vorhanden. Zum anderen müssen Schlagworte, mit denen die Literatursuche unterstützt werden kann, nicht notwendigerweise vorhanden sein.
3. Alle Attribute, auf die die ersten beiden Kriterien nicht zutreffen, können nicht weggelassen werden.

Bücherkatalog: In einem Bücherkatalog sind Daten von Büchern enthalten.

Das Analysemodell besteht nicht ausschließlich aus Objekten, die direkt aus dem Problembereichsmodell übernommen werden können. In einigen Fällen ist es sinnvoll, von den Vorgaben des Problembereichsmodells abzuweichen. Dies soll beispielhaft anhand der Abbildung 3.7 dargestellt werden.

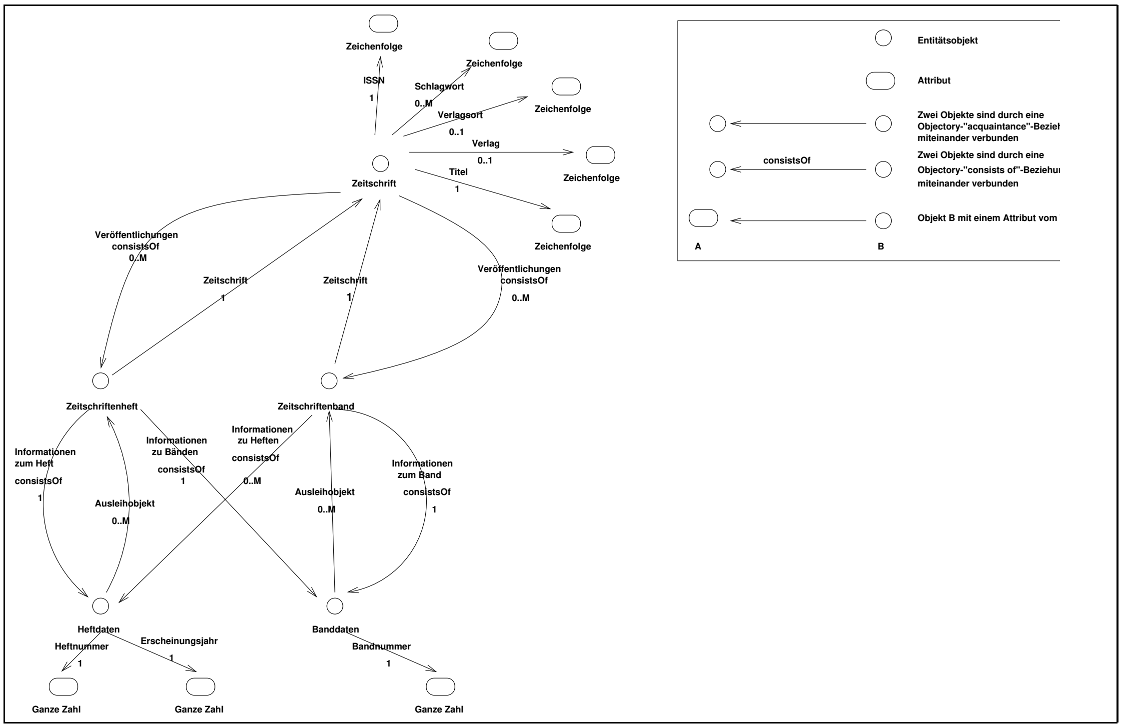

Abbildung 3.7: Objekte, mit Daten für Zeitschriften, Zeitschriftenheften und Zeitschriftenbänden. Das Fehlen statischer Beziehungen zwischen den Objekten Zeitschriftenheft und Zeitschriftenband ist beabsichtigt (die Erläuterung hierzu wird auf Seite 36 gegeben).

In dieser Abbildung sind wiederum einige Objekte des Analysemodells und ihre statischen Beziehungen zueinander dargestellt. Dabei werden teilweise gleiche Bezeichnungen für statische Beziehungen und für Objekte verwendet (z.B. die Bezeichnung Zeitschrift). Dies wird von Jacobson et al. ausdrücklich zugelassen und sogar positiv bewertet [JCJÖ93, S. 180]. Zusätzlich zu den „acquaintance"-Beziehungen, die bereits bei der Beschreibung des Problembereichsmodells für das betrachtete Fallbeispiel vorgestellt wurden (siehe Abschnitt 3.4.1), werden hier auch „*consists of*"-Beziehungen benutzt. Mit Hilfe von „consists of"-Beziehungen wird beschrieben, daß sich ein Objekt aus mehreren anderen Objekten zusammensetzt. Eine Beschreibung der „consists of"-Beziehungen geben Jacobson et al. [JCJÖ93, S. 181].

Eine Zeitschrift besteht also nach der Skizze aus der Abbildung 3.7 aus Zeitschriftenheften (Objekt Zeitschriftenheft) und aus Zeitschriftenbänden (Objekt Zeitschriftenband). Ein Zeitschriftenheft setzt sich aus Daten, die sich speziell auf dieses Heft beziehen und aus Daten, die sich auf den gesamten Zeitschriftenband beziehen, zu dem das Heft gehört, zusammen (Objekte Heftdaten und Banddaten). Ein Zeitschriftenband setzt sich aus Daten, die sich auf die Hefte beziehen, deren Inhalte in dem Zeitschriftenband enthalten sind und aus Daten, die sich auf den Band selbst beziehen, zusammen. Während die Objekte Zeitschrift, Zeitschriftenheft und Zeitschriftenband direkt aus dem Problembereichsmodell übernommen werden, existieren zu den Objekten Heftdaten und Banddaten keine entsprechenden Objekte im Problembereichsmodell (siehe Tabelle 3.2 im Abschnitt 3.4.1). Die Objekte Heftdaten und Banddaten werden zur Speicherung der Informationen benutzt, die zu Zeitschriftenheften bzw. zu Zeitschriftenbänden benötigt werden. Die Einführung der neuen Objekte soll im folgenden begründet werden. Dabei wird auch darauf eingegangen, warum die Objekte Zeitschriftenheft und Zeitschriftenband im Analysemodell nicht unmittelbar durch statische Beziehungen miteinander verbunden sind.

Zunächst werden hierzu einige Forderungen an die Ausleihobjekte Zeitschriftenheft und Zeitschriftenband in einer Bibliothek betrachtet:

Forderung 1: Zeitschriftenhefte können als eigenständige Ausleihobjekte vorhanden sein, ohne daß der zugehörige Zeitschriftenband als Ausleihobjekt verfügbar ist.

Forderung 2: Ein Zeitschriftenband kann in der Bibliothek als eigenständiges Ausleihobjekt vorhanden sein, ohne daß die Zeitschriftenhefte, deren Inhalte im Zeitschriftenband enthalten sind, noch als eigenständige Ausleihobjekte existieren.

Eine Speicherung der Bandnummer in einem Attribut des Objektes Zeitschriftenband und die Modellierung eines Verweises vom Objekt Zeitschriftenheft auf das Objekt Zeitschriftenband bietet sich nicht an, weil in diesem Fall die Nummer eines Bandes, zu dem ein Zeitschriftenheft gehört nicht gespeichert werden könnte, wenn dieser Band nicht als eigenständiges Ausleihobjekt in der Bibliothek vorhanden ist (siehe Forderung 1).

Eine Speicherung der Heftnummer und des Erscheinungsjahres in Attributen des Objektes Zeitschriftenheft und die Modellierung eines Verweises vom Objekt Zeitschriftenband auf das Objekt Zeitschriftenheft bietet sich nicht an, weil in diesem Fall die Heftnummern und die Erscheinungsjahre der Hefte, deren Inhalte in einem Zeitschriftenband enthalten sind, nicht zu dem entsprechenden Zeitschriftenband gespeichert werden können, wenn diese Hefte nicht als eigenständige Ausleihobjekte in der Bibliothek vorhanden sind (siehe Forderung 2).

Während die Objekte Zeitschriftenheft und Zeitschriftenband immer Ausleihobjekte repräsentieren, die in der tatsächlich Bibliothek vorhanden sind (siehe Abbildung 3.4), beziehen sich die Objekte Heftdaten und Banddaten nicht auf konkrete Ausleihobjekte. Sie dienen lediglich als Datenspeicher. Deshalb könnnen hier auch Daten zu Zeitschriftenheften bzw. zu Zeitschriftenbänden gespeichert werden, wenn diese Hefte bzw. Bände nicht als Ausleihobjekte in der betrachteten Bibliothek vorhanden sind.

Der für das erste Fallbeispiel erstellte Teil des Analysemodells enthält noch weitere Objekte, auf die im folgenden aus Platzgründen nicht mehr ausführlich eingegangen werden kann. Sie sind in der Tabelle 3.3 enthalten, die eine Übersicht über sämtliche modellierten Analysemodellobjekte liefert. Das vollständige Analysemodell ist im Zusatzdokument zu dieser Diplomarbeit aufgeführt.

Objekt/Klasse	Beschreibung
Ausleihkarte	Im Objekt Ausleihkarte werden die Daten eines verliehenen Ausleihobjektes, des Entleihers und einer möglichen Reservierung erfaßt.
Ausleihkartei	In der Ausleihkartei werden Ausleihkarten gesammelt.
Ausleihobjekt	Das Objekt Ausleihobjekt enthält die Daten, die allen Ausleihobjekten (siehe Glossar) gemeinsam zugeordnet werden können.
Banddaten	Dieses Objekt enthält zusätzliche Informationen zu einem Zeitschriftenband. Im Objekt Banddaten werden die Daten zusammengefaßt, die für alle Exemplare eines Zeitschriftenbandes mit einer bestimmten Bandnummer gemeinsam gelten (siehe auch Objekt Zeitschriftenband).
Benutzerdatenbank	In der Benutzerdatenbank werden die zu den Bibliotheksbenutzern erfaßten Daten verwaltet.
Bibliotheksbenutzer	Das Objekt Bibliotheksbenutzer enthält die Daten, die zu einem Bibliotheksbenutzer erfaßt werden.
Buch	Das Objekt Buch enthält die Daten, die zu einem Buch erfaßt sind.
Bücherkatalog	Im Bücherkatalog werden Daten von Büchern verwaltet.
Gerätedatenbank	In der Gerätedatenbank werden Daten, die zu den in der Bibliothek vorhandenen Geräten erfaßt sind, verwaltet.
Gerät	Das Objekt Gerät enthält die Daten, die zu einem in der Bibliothek vorhandenen Gerät erfaßt sind.
Heftdaten	Dieses Objekt enthält neben dem Objekt zusätzliche Informationen zu einem Zeitschriftenheft. Im Objekt Heftdaten werden die Daten zusammengefaßt, die für alle Exemplare eines Zeitschriftenheftes mit einer bestimmten Heftnummer gemeinsam gelten (siehe auch Objekt Zeitschriftenheft).
Inventargegenstand	Alle Gegenstände, die zum Inventar der Bibliothek gehören, werden hier unter dem Begriff Inventargegenstand zusammengefaßt. In der Klasse Inventargegenstand sind die Attribute definiert, die zu allen Gegenständen der Bibliothek gespeichert werden.
Katalog	Die Klasse Katalog dient als gemeinsame Oberklasse für die Klassen Bücherkatalog und Zeitschriftenkatalog.
Leihfristenkombination	Im Objekt Leihfristenkombination werden Leihfristen bzw. Verlängerungen von Leihfristen für jede Benutzergruppe der Bibliothek gespeichert.
Mitarbeiter	Das Objekt Mitarbeiter enthält die Daten, die zu einem Mitarbeiter der Bibliothek erfaßt sind.
Person	Die Klasse Person enthält die Attribute, die sowohl für die Daten der Mitarbeiter der Bibliothek als auch für die Daten der Bibliotheksbenutzer benötigt werden.
Personaldatenbank	In der Personaldatenbank werden die Daten der Mitarbeiter der Bibliothek verwaltet.
Zeitschrift	Das Objekt Zeitschrift enthält die Daten, die zu einer Zeitschrift erfaßt sind.
Zeitschriftenband	Das Objekt Zeitschriftenband enthält die Daten, die zu einem Zeitschriftenband erfaßt sind. Für jedes in der Bibliothek vorhandene Exemplar eines Zeitschriftenbandes existiert ein solches Objekt (siehe auch Objekt Banddaten).
Zeitschriftenheft	Das Objekt Zeitschriftenheft enthält die Daten, die zu einem Zeitschriftenheft erfaßt sind. Für jedes in der Bibliothek vorhandene Exemplar eines Zeitschriftenheftes existiert ein solches Objekt (siehe auch Objekt Heftdaten).
Zeitschriftenkatalog	Im Zeitschriftenkatalog werden Daten von Zeitschriften sowie von den zu der Zeischrift gehörenden Zeitschriftenheften und Zeitschriftenbänden verwaltet.

Tabelle 3.3: Objekte (bzw. Klassen) im erstellten Teil des Analysemodells.

3.5 Abbildung der erstellten Objectory-Modelle auf ProSet

Im folgenden wird die Entwicklung des ProSet-Prototyps auf der Basis der erstellten Objectory-Modelle beschrieben. Dabei wird im Abschnitt 3.5.1 wird zunächst auf die Implementierung der Materialien eingegangen. Abschnitt 3.5.2 beschreibt die Vorgehensweise bei der Implementierung der Werkzeuge. Im Rahmen der Diplomarbeit werden die Werkzeugklassen zeitlich nach den Materialklassen entwickelt. Die Schnittstellen der Werkzeugklassen orientieren sich an den Schnittstellen, die von den Materialklassen angeboten werden. Auf die Implementierung von Aspektklassen (siehe Abschnitt 2.2) wird verzichtet.

Im Abschnitt 3.5.3 wird abschließend ein Überblick über die erstellten Programme für den Prototyp des Systems zur Bibliotheksverwaltung sowie über die Organisation der benötigten persistenten Daten gegeben.

3.5.1 Materialien

Die Klassen aller Objekte, die im erstellten Teil des Analysemodells aufgeführt sind, werden im folgenden als Materialklassen bezeichnet. Die Abbildung dieser Materialklassen auf ein ProSet-Programm wird in diesem Abschnitt betrachtet. Dieser Abschnitt ist in zwei Teilabschnitte unterteilt. Im ersten Teil wird die grundsätzliche Vorgehensweise bei der Implementierung der Materialklassen festgelegt und motiviert. Im zweiten Teilabschnitt wird anhand der Klasse Buch beispielhaft auf die Umsetzung der vorgestellten Konzepte eingegangen.

Konzepte zur Abbildung der Materialklassen auf ProSet

Zur Implementierung der Materialklassen bzw. der Materialinstanzen werden hier zunächst zwei Möglichkeiten gegenübergestellt und bewertet. Die Implementierung von Beziehungen zwischen Klassen bzw. zwischen Instanzen wird anschließend betrachtet.

Variante 1: Für jede Materialklasse aus dem Analysemodell wird ein ProSet-Modul erstellt. In diesem Modul sind die Datenstrukturen und Operationen zum Zugriff auf gespeicherte Daten in Form von Prozeduren definiert. Für jedes benötigte Datenobjekt wird eine Modulinstanz erzeugt und persistent gemacht.

Beispiel: Um Daten von Büchern abzuspeichern, wird zunächst ein Modul Buch definiert, in dem die Datenstrukturen zum Speichern der Attribute und die Zugriffsoperationen auf die Daten definiert sind. Für jedes Buch in der Bibliothek wird anschließend eine Instanz dieses Moduls erzeugt und persistent gemacht.

Variante 2: Für jedes benötigte Datenobjekt wird eine Variable eines zusammengesetzten Datentyps persistent gemacht. Mögliche zusammengesetzte Datentypen sind in ProSet Tupel (*tuples*) und Mengen (*sets*), mit denen auch Abbildungen (*maps*) realisiert werden können. In einer solchen persistenten Variablen sind die Daten eines Objektes gespeichert, nicht jedoch die Zugriffsoperationen auf die Daten. Damit von Instanzen der Werkzeugklassen aus nicht direkt auf die Daten zugegriffen werden muß, wird zu jeder Materialklasse ein Modul implementiert, in dem die Zugriffsoperationen für alle Daten der Objekte dieser Klasse definiert sind. Für alle Daten einer Materialklasse existiert nur ein Modul mit Zugriffsoperationen. Instanzen der Werkzeugklassen greifen auf die Daten über die bereitgestellten Zugriffsoperationen zu. Beim Aufruf einer Zugriffsoperation muß immer angegeben werden, auf welches Datenobjekt der betroffenen Klasse sich der Zugriff bezieht.

Beispiel: Um die Daten der in der Bibliothek vorhandenen Bücher zu speichern, müßte für jedes Buch eine Variable, in der die Daten für die Attribute der jeweiligen Bücher gespeichert sind, definiert und persistent gemacht werden. Hierbei könnte es sich um ein Tupel handeln, in dem die Daten für alle Attribute eines Buches in einer vorgegebenen Reihenfolge gespeichert sind (das Tupel ist nur ein Beispiel für eine mögliche Realisierung und bezieht sich noch nicht auf die tatsächlich benutzte Datenstruktur). Attribute eines Buches seien in diesem Beispiel der Titel, der Verlag und die ISBN. Um dieses Beispiel zu vereinfachen, werden übrigen Attribute aus der Abbildung 3.5 (Abschnitt 3.4.2) nicht berücksichtigt.

Für jedes in der Bibliothek vorhandene Buch wird ein 3-Tupel der folgenden Form persistent gemacht:

[Buchtitel, Buchverlag, ISBN des Buches].

Zusätzlich wird ein Modul Buch implementiert, in dem die Zugriffsoperationen für die gespeicherten Daten der Bücher definiert sind. Wenn die Daten eines Buches gelesen werden sollen, wird eine Prozedur buchdaten_lesen einer Instanz des Moduls Buch aufgerufen. Beim Aufruf dieser Prozedur wird eine eindeutige Kennzeichnung des Buches, dessen Daten gelesen werden sollen, übergeben.

Auf die Speicherung der persistenten Variablen, das Erzeugen eindeutiger Kennzeichnungen und die entstehenden Probleme wird weiter unten im Abschnitt 3.5.1 noch detailliert eingegangen.

Die erste vorgestellte Realisierungsalternative erscheint aus software-technologischer Sicht sauberer, weil hier jeder Instanz einer Objektklasse des Analysemodells eine Instanz eines ProSet-Moduls zugeordnet wird. Dies ist bei der zweiten vorgestellten Realisierungsalternative nicht der Fall. Die zweite Variante bietet demgegenüber den Vorteil, daß im persistenten Speicher nur die Daten abgelegt werden, die verändert werden können. Dadurch wird der Platz im persistenten Datenspeicher effizienter genutzt. So müßten bei einer Realisierung der zuerst vorgestellten Alternative zu jedem Buch, das in den Datenbestand aufgenommen wird, alle Zugriffsoperationen abgespeichert werden.

Bei der Implementierung des ProSet-Prototyps für das betrachtete Fallbeispiel wird die zweite vorgestellte Alternative realisiert, weil die Menge der in einer Bibliothek anfallenden Daten als so groß eingeschätzt wird, daß der Aspekt einer effizienten Nutzung des persistenten Datenspeichers bei dem zu erstellenden ProSet-Prototyp nicht vernachlässigt werden kann, obwohl Überlegungen zur Effizienz ansonsten beim explorativen Prototyping weniger wichtig sind. Ein Eintrag im persistenten Datenspeicher bzw. der Bestandteil eines Eintrags im persistenten Datenspeicher, in dem die Daten genau eines Materialobjektes des Analysemodells gespeichert sind, wird bei der weiteren Beschreibung dieses Fallbeispiels als *Datensatz* bezeichnet.

Im erstellten Teil des Analysemodells sind neben den Datenobjekten auch Beziehungen zwischen Klassen dieser Objekte und Beziehungen zwischen Instanzen aufgeführt. Die Darstellung dieser Beziehungen im ProSet-Programm soll im folgenden betrachtet werden. Dabei wird immer vorausgesetzt, daß die Implementierung der Materialklassen bzw. -instanzen des Analysemodells gemäß der oben vorgestellten Variante 2 erfolgt. Die Instanzen implementierter Moduln stellen also ausschließlich Zugriffsoperationen für persistente Daten bereit, besitzen selbst aber keine eigenen Variablen, um interne Zustände abzuspeichern.

Abbildung der Beziehungen zwischen Instanzen: Jedem Datensatz wird ein Schlüsselattribut zugeordnet, durch das der zugehörige Datensatz eindeutig identifiziert werden kann. In einigen Fällen kann hierfür ein eindeutiges Attribut, das im Objectory-Analysemodell aufgeführt ist, genommen werden. In anderen Fällen muß ein Schlüsselattribut neu erzeugt werden. So ist z.B. jeder Datensatz, in dem die Daten des Datenobjektes Inventargegenstand (siehe Abbildung 3.5 im Abschnitt 3.4.2) gespeichert sind, durch die Inventarnummer bereits eindeutig gekennzeichnet, während das Analysemodellobjekt Banddaten (siehe Abbildung 3.7 im Abschnitt 3.4.2) kein Schlüsselattribut besitzt. Für jeden Datensatz, in dem die Daten eines Objektes Banddaten gespeichert sind, wird ein Schlüsselattribut explizit eingefügt.

Mit Hilfe der Schlüssel werden Beziehungen zwischen Instanzen implementiert. Jeder Datensatz enthält neben den Daten des zugehörigen Analysemodellobjektes Einträge, in denen die Schlüssel der Objekte gespeichert sind, zu denen eine Instanzbeziehung besteht. Für die Sicherung der Konsistenz des Datenbestandes sind die Prozeduren in den Werkzeugklassen zuständig. Wenn ein Datensatz mit Daten eines Materialobjektes gelöscht wird, sorgt die Prozedur der Werkzeugklasseninstanz, die das Löschen veranlaßt hat, dafür, daß auch die ungültig gewordenen Verweise in anderen Datensätzen entfernt werden.

Abbildung der Beziehungen zwischen Klassen: Die Beziehungen zwischen den im erstellten Teil des Analysemodells dargestellten Materialklassen dienen dazu, ausgehend von allgemeinen Klassen, Spezialisierungen vorzunehmen, wie es in der Abbildung 3.4 im Abschnitt 3.4.2 beispielhaft dargestellt ist (andere mögliche Gründe für die Modellierung von Beziehungen zwischen Klassen werden bei Jacobson et al. [JCJÖ93, S. 64f.] beschrieben).

Ausgehend von der Klasse Inventargegenstand werden bei diesem Beispiel die spezielleren Klassen Gerät und Ausleihobjekt abgeleitet. Die Klasse Ausleihobjekt dient als Oberklasse für die spezielleren Klassen Buch, Zeitschriftenheft und Zeitschriftenband. Um die Spezialisierungen zu modellieren, werden die von Objectory angebotenen „inherit“-Beziehungen [JCJÖ93, S. 59ff.] benutzt. Im folgenden werden zunächst die grundlegenden Ideen zweier verschiedener Ansätze zur Abbildung der Vererbungsbeziehungen beschrieben und miteinander verglichen. Anschließend wird die bei der Bearbeitung dieses Fallbeispiels realisierte Umsetzung der vorgestellten Konzepte auf eine ProSet-Implementierung vorgestellt.

Zur Vereinfachung der Beschreibung werden Moduln, in denen die Zugriffsoperationen für die Daten erbender Klassen definiert sind, im folgenden als *erbende Moduln* oder *Nachfolger* bezeichnet. Moduln mit Zugriffsoperationen für die Daten vererbender Klassen werden als *vererbende Moduln* oder *Vorgänger* bezeichnet. Sind zwei Klassen im Objectory-Analysemodell direkt durch eine Kante verbunden, werden die zugehörigen Moduln als *direkter Vorgänger* bzw. *direkter Nachfolger* bezeichnet.

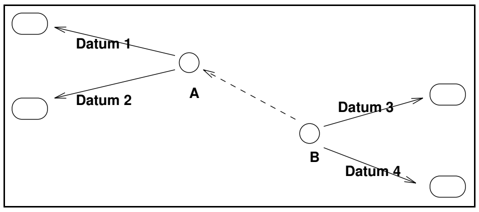

Abbildung 3.8: Beispiel für eine Vererbungsbeziehung im Analysemodell.

Variante 1: In erbenden Moduln werden alle direkten Vorgänger instanziiert. Nachfolger importieren sämtliche Prozeduren, die in der Exportschnittstelle der direkten Vorgänger aufgeführt sind. Zusätzlich werden in den erbenden Moduln Prozeduren definiert, die die gleichen Namen besitzen wie die von den direkten Vorgängern importierten Prozeduren. In diesen Prozeduren werden zum einen die gleichnamigen Prozeduren der direkten Vorgänger aufgerufen, zum anderen werden Anpassungen an die Bedürfnisse des erbenden Moduls vorgenommen. Die wesentlichen Grundlagen sollen anhand des folgenden Beispiels verdeutlicht werden:

Beispiel: Bei einer Vererbungsbeziehung gemäß Abbildung 3.8 gibt es einen Modul A und einen Modul B. Der Modul A enthält Zugriffsoperationen für die Datensätze, in denen die Daten für die Attribute Datum 1 und Datum 2 gespeichert sind. Dabei soll es sich um die Prozeduren aufnehmen und loeschen handeln. Der Modul A ist folgendermaßen implementiert:

```
module A
export aufnehmen, loeschen;

begin
   pass;
   procedure aufnehmen(rd d_1,
                       rd d_2);
   hidden persistent
      datum_1, datum_2: "Daten_P_File";
   begin
      ------------------------------------
      -- Implementierung der Prozedur auf-
      -- nehmen; Bearbeitung der P-File-
      -- Eintraege mit den Daten der
      -- Attribute Datum 1 und Datum 2
      ------------------------------------

      ...

   end aufnehmen;

   procedure loeschen();
   hidden persistent
      datum_1, datum_2: "Daten_P_File";
   begin
      ------------------------------------
      -- Implementierung der Prozedur loe-
      -- schen; Bearbeitung der P-File-
      -- Eintraege mit den Daten der
      -- Attribute Datum 1 und Datum 2
      ------------------------------------

      ...

   end loeschen;
end A;
```

Der Modul B besitzt in jedem Fall auch die Prozeduren aufnehmen und loeschen. Während diese Prozeduren beim direkten Vorgänger, dem Modul A, nur die Daten der Attribute Datum 1 und Datum 2 manipulieren, sollen diese Prozeduren im Modul B die Daten der Attribute Datum 1, Datum 2, Datum 3 und Datum 4 manipulieren. Der prinzipielle Aufbau des Moduls B sieht dann folgendermaßen aus:

```
module B
import A;
export aufnehmen, loeschen;

visible constant
   a := instantiate (closure A)
        import
           aufnehmen,
           loeschen;
        end instantiate;

           -- Instanzen besitzen hier
           -- nur Zugriffsoperationen,
           -- aber keine veraenderbaren
           -- Zustaende
begin
   pass;
   procedure aufnehmen(rd d_1,
                       rd d_2,
                       rd d_3,
                       rd d_4);
   hidden persistent
      datum_3, datum_4: "Daten_P_File";
   begin
      a.aufnehmen(d_1,d_2);

      ...

      ----------------------------------
      -- zusaetzliche Befehle zur Bear-
      -- beitung der P-File-Eintraege mit
      -- den Daten der Attribute Datum 3
      -- und Datum 4
      ----------------------------------
   end aufnehmen;

   procedure loeschen();
   hidden persistent
      datum_3, datum_4: "Daten_P_File";
   begin
      a.loeschen();

      ...

      ----------------------------------
      -- zusaetzliche Befehle zur Bear-
      -- beitung der P-File-Eintraege mit
      -- mit Daten der Attribute Datum 3
      -- und Datum 4
      ----------------------------------
   end loeschen;

   ...

end B;
```

Zusätzlich können im Modul B weitere Prozeduren definiert sein, die nicht von Vorgängern geerbt werden.

Variante 2: Bei dieser Variante wird die folgende Besonderheit des für das Bibliotheksverwaltungssystem erstellten Teils des Analysemodells ausgenutzt: Bei vererbenden Klassen handelt es sich ausschließlich um abstrakte Klassen, d.h. es werden keine Instanzen vererbender Klassen benötigt. Dies soll exemplarisch am Beispiel der Abbildung 3.4 veranschaulicht werden, die bei der Beschreibung des Analysemodells im Abschnitt 3.4.2 erläutert wurde:

Es gibt keinen Inventargegenstand, der weder Gerät noch Ausleihobjekt ist. Ist ein Inventargegenstand ein Ausleihobjekt, dann ist er entweder ein Buch, ein Zeitschriftenband oder ein Zeitschriftenheft. Bei der folgenden Beschreibung einer Realisierungsmöglichkeit wird die Besonderheit, daß es sich bei vererbenden Klassen immer um abstrakte Klassen handelt, vorausgesetzt.

Alle Prozeduren der Vorgänger werden in den erbenden Moduln erneut definiert und, falls nötig, erweitert. Es wird nicht auf Prozeduren, die in den vererbenden Moduln definiert sind, zurückgegriffen. Ein Datensatz besitzt neben den Attributen des Analysemodellobjektes, dem er zu geordnet ist, zusätzlich alle Attribute der Analysemodellobjekte vererbender Klassen. Auf die Implementierung sämtlicher Klassen, von denen geerbt wird, kann verzichtet werden, weil von diesen Klassen keine Instanzen benötigt werden. Die grundlegende Idee wird anhand eines Beispiels demonstriert.

Beispiel: Grundlage dieses Beispiels bildet wiederum die in der Abbildung 3.8 skizzierte Vererbungsbeziehung. Es wird vorausgesetzt, daß keine Instanz der Klasse A erforderlich ist.

Auf die Implementierung eines Moduls zur Klasse A kann nun vollständig verzichtet werden. Der Datensatz, auf den durch die Prozeduren aufnehmen und loeschen des Moduls zur Klasse B zugegriffen wird, enthält jetzt nicht mehr ausschließlich Daten für die Attribute Datum 3 und Datum 4, sondern zusätzlich noch Daten für die „geerbten" Attribute Datum 1 und Datum 2. Bei der Implementierung des Moduls B werden die Prozeduren aufnehmen und loeschen vollständig neu implementiert:

```
module B
export aufnehmen, loeschen;

begin
   pass;
   procedure aufnehmen(rd d_1,
                       rd d_2,
                       rd d_3,
```

```
                  rd d_4);
hidden persistent
   datum_1, datum_2,
   datum_3, datum_4: "Daten_P_File";
begin
   -----------------------------------
   -- Implementierung der Prozedur auf-
   -- nehmen; Bearbeitung der P-File-
   -- Eintraege mit den Daten der
   -- Attribute Datum 1, Datum 2,
   -- Datum 3 und Datum 4
   -----------------------------------

   ...

end aufnehmen;

procedure loeschen();
hidden persistent
   datum_1, datum_2,
   datum_3, datum_4: "Daten_P_File";
begin
   -----------------------------------
   -- Implementierung der Prozedur
   -- loeschen; Bearbeitung der
   -- P-File Eintraege mit den Daten
   -- der Attribute Datum 1, Datum 2,
   -- Datum 3 und Datum 4
   -----------------------------------

   ...

end loeschen;

...

end B;
```

Im weiteren Verlauf dieses Abschnittes wird das zuerst vorgestellte Konzept abkürzend als *Variante 1* und das zweite Konzept als *Variante 2* bezeichnet.

Die vorgestellten Alternativen stellen nicht die einzigen möglichen Vorgehensweisen dar. Sie werden exemplarisch für zwei grundsätzlich verschiedene Ansätze bei der Abbildung der Vererbungsbeziehungen betrachtet. Während bei der Variante 1 Vererbungsbeziehungen dadurch dargestellt werden, daß auf bereits bestehende Moduln zurückgegriffen wird, wird bei der Variante 2 jeder Modul so implementiert, daß er von anderen Moduln unabhängig ist.

Bei der Entscheidung, auf welchem Konzept die Abbildung der Vererbungsbeziehungen auf einen ProSet-Prototyp basieren soll, müssen neben software-technologischen Gesichtspunkten noch Einschränkungen berücksichtigt werden, die durch die verwendete Version 0.6 des ProSet-Compilers vorgegeben sind. So bricht der Compiler einen Übersetzungsvorgang mit der Fehlermeldung internal assembler error: out of memory ab, wenn die folgenden Voraussetzungen gleichzeitig gegeben sind:

- In mindestens einem Modul sind Prozeduren implementiert, die persistente Variablen benutzen. Auf diesen Modul wird das closure-Konstrukt [DFH+95, S. 72] angewendet.
- Die Anzahl der Prozeduren in Moduln, auf die das closure-Konstrukt angewendet wird, ist zu hoch. Eine detaillierte Analyse über die zulässige Anzahl und den erlaubten Umfang der Prozeduren wurde aus Zeitgründen im Rahmen dieser Diplomarbeit nicht durchgeführt. Bei dem betrachteten Fallbeispiel liegt die obere Grenze bei einem Modul mit ca. 22 Prozeduren. Die Zahl 22 gibt hierbei nur einen Richtwert an. Je nach Umfang der implementierten Prozeduren kann sich dieser Wert sowohl nach oben als auch nach unten verschieben.

Diese Schwierigkeiten können dadurch umgangen werden, daß sämtliche Moduln getrennt voneinander übersetzt und persistent gemacht werden. Die übersetzten persistenten Moduln können anschließend bei Bedarf geladen und instanziiert werden. Bei einer Realisierung der Variante 2 bereitete dies keine Probleme. Die Variante 1 könnte jedoch nur in abgewandelter Form realisiert werden. Es wäre nicht möglich, direkte Vorgänger als globale persistente Konstanten in erbenden Moduln zu deklarieren, weil ProSet keine globalen persistenten Konstanten in Moduln erlaubt [DFH+95, S. 73]. Eine Möglichkeit, diese Einschränkung zu umgehen, wird im folgenden vorgestellt:

Der persistente Vorgängermodul mit den Zugriffsoperationen wird lokal in einer Prozedur eines direkten Nachfolgers, die direkt nach der Instanziierung dieses direkten Nachfolgers aufgerufen wird, geladen und instanziiert. Die Instanz des vererbenden Moduls wird einer globalen Variable im erbenden Modul übergeben. Diese modifizierte Variante 1 wird wiederum anhand eines Beispiels auf Basis der Abbildung 3.8 erläutert.

Beispiel: Die Implementierung des vererbenden Moduls A ändert sich gegenüber der ursprünglichen Version der Variante 1 nicht. Der Modul A wird in einem P-File gespeichert. Der Name des zugehörigen P-File-Eintrags sei bei diesem Beispiel P_A. Bevor der Modul A als P-File-Eintrag P_A gespeichert wird, wird das closure-Konstrukt auf den Modul A angewendet. Der Name des P-Files mit dem Eintrag P_A sei P_File_mit_persistenten_Moduln.

Die Implementierung des Moduls B unterscheidet sich von der ursprünglichen Version. Nach der Instanziierung des erbenden Moduls B wird bei der modifizierten Version die Prozedur vererbung_realisieren aufgerufen, in der der P-File-Eintrag P_A geladen wird. Der gespeicherte Modul wird instanziiert und anschließend an die globale Variable a übergeben.

```
module B
export aufnehmen, loeschen;
-- Die Importschnittstelle des
-- Moduls wird nicht mehr benoetigt

visible a;

begin
   vererbung_realisieren();

   procedure vererbung_realisieren();
   hidden persistent constant
      P_A: "P_File_mit_persistenten_Moduln";
   begin
      a := instantiate P_A
            -- das closure-Konstrukt wurde
            -- bereits auf den Modul A an-
            -- gewendet, bevor A als P-File-
            -- Eintrag P_A gespeichert wurde
         import
            aufnehmen,
            loeschen;
         end instantiate;
   end vererbung_realisieren;


   procedure aufnehmen(rd d_1,
                       rd d_2,
                       rd d_3,
                       rd d_4);
   hidden persistent
      datum_3, datum_4: "Daten_P_File";
   begin
      ------------------------------------
      -- Implementierung wie bei alter
      -- Variante 1
      ------------------------------------

      ...

   end aufnehmen;

   procedure loeschen();
   hidden persistent
      datum_3, datum_4: "Daten_P_File";
   begin
      ------------------------------------
      -- Implementierung wie bei alter
      -- Variante 1
      ------------------------------------

      ...

   end loeschen;
end B;
```

Bei einem Vergleich der beiden vorgestellten Varianten zur Abbildung der Vererbungsbeziehungen, erscheint der Implementierungsaufwand bei der Variante 2 geringer, weil hier auf die Implementierung von vererbenden Moduln verzichtet werden kann. Bei der Variante 1 kann zwar in erbenden Moduln auf den Code vererbender Moduln zurückgegriffen werden, jedoch wird aus den folgenden Gründen nicht erwartet, daß sich hieraus Vorteile für den Implementierungsaufwand oder für die Änderungsfreundlichkeit ergeben:

- Es reicht bei der Variante 1 nicht aus, in erbenden Moduln nur die Prozeduren zu implementieren, die zusätzlich zu geerbten Prozeduren benötigt werden. Es muß für jede Prozedur, die geerbt wird, eine neue Prozedur implementiert werden, in der auf die zugehörige Prozedur des direkten Vorgängers verwiesen wird (siehe Beschreibung der Variante 1).

  Da bei einer Vererbungsbeziehung immer sämtliche Prozeduren der Vorgänger geerbt werden, ist die Anzahl der Prozeduren, die in erbenden Moduln definiert sein müssen, bei einer Realisierung der Variante 1 nicht geringer als bei einer Realisierung der Variante 2. In erbenden Moduln muß bei der Variante 1 sogar noch eine zusätzliche Prozedur zum Laden und zur Instanziierung der vererbenden Moduln implementiert werden.

- Wenn innerhalb einer bereits implementierten Prozedur eines vererbenden Moduls eine Änderung vorgenommen werden soll, bietet die Variante 1 zwar den Vorteil, daß es hier ausreicht, diese Änderung nur einmal durchzuführen, während die Änderung bei der Variante 2 üblicherweise mehrfach durchgeführt werden muß.

  Demgegenüber bietet die Variante 2 jedoch Vorteile, wenn neue Prozeduren in vererbende Moduln eingefügt oder aus vererbenden Moduln entfernt werden sollen oder wenn

Prozeduren vererbender Moduln umbenannt werden sollen. So reicht es bei der Variante 2 aus, diese Änderungen in den erbenden Moduln durchzuführen, die selbst keine Nachfolger mehr besitzen. Bei einer Realisierung der Variante 1 ziehen das Einfügen, das Löschen und das Umbenennen von Prozeduren eines vererbenden Moduls mehrere Folgeänderungen in sämtlichen Nachfolgern — also auch in den Nachfolgern, die selbst wieder weitere Nachfolger besitzen — nach sich. Dabei müssen sowohl die Exportlisten der Moduln als auch die Importlisten, die bei Instanziierungen von Vorgängermoduln angegeben werden, überarbeitet werden. Des weiteren müssen zusätzliche Anpassungen in denjenigen Prozeduren vorgenommen werden, in denen die geerbten Prozeduren direkter Vorgänger aufgerufen werden (bei diesen Anpassungen kann es sich z.B. um die Änderung des Namens der aufgerufenen Prozedur handeln).

Da nach den oben aufgeführten Argumenten die Variante 1 keine wesentlichen Vorteile gegenüber der Variante 2 in Bezug auf die Änderungsfreundlichkeit der implementierten Moduln bietet und andererseits bei einer Realisierung der Variante 2 auf die Implementierung der Moduln zur Repräsentation abstrakter Klassen verzichtet werden kann, wird eine abgewandelte Form der Variante 2 realisiert, die im folgenden Teilabschnitt beschrieben wird.

Umsetzung der vorgestellten Konzepte

Die im Rahmen dieses Fallbeispiels realisierte Abbildung der Vererbungsbeziehungen weicht dadurch von der beschriebenen Variante 2 ab, daß nicht die Moduln mit den Zugriffsoperationen auf Datensätze persistent gemacht werden, sondern Instanzen dieser Moduln. Dies hat den Nachteil, daß in Programmen bzw. Prozeduren, in denen diese persistenten Instanzen benötigt werden, die Importschnittstelle nicht mehr frei gewählt werden kann. Die Importschnittstelle wird bereits festgelegt, wenn die Instanzen erzeugt werden. Sie umfaßt bei den erstellten Moduln alle Prozeduren, die in der Exportschnittstelle dieser Moduln aufgeführt sind. Diese Abweichung der realisierten Implementierung von der beschriebenen Variante kommt dadurch zustande, daß die Entwicklung eines Konzeptes zur Implementierung der Materialien und zum Teil auch die Implementierung selbst parallel zur Entwicklung einer stabilen Version des verwendeten ProSet-Compilers verliefen. Dadurch waren die Testmöglichkeiten in Bezug auf die Nutzung der Persistenz vor Beginn der Implementierung eingeschränkt. Bei der Implementierung wurde zunächst davon ausgegangen, daß die beschriebenen Speicherplatzprobleme des Assemblers durch die Instanziierung von Moduln und nicht durch die Anwendung des closure-Konstrukts ausgelöst werden. Erst durch Experimente mit einer stabilen Version des ProSet-Compilers wurde die Fehleinschätzung deutlich. Da ein Prototyp zur Gewinnung neuer Erkenntnisse gedacht ist und sich im Verhalten des Prototyps nach außen keine Unterschiede dadurch ergeben, ob persistente Moduln oder persistente Instanzen genutzt werden, wurde die Implementierung nicht nachträglich geändert.

Bei der Betrachtung der konkreten Implementierung des Prototyps für das System zur Bibliotheksverwaltung wird aus Platzgründen exemplarisch auf die Abbildung der Klasse Buch auf ProSet eingegangen. Die Implementierung der Datenstrukturen und Moduln zu den anderen Klassen verläuft analog zum vorgestellten Beispiel.

Die Vererbungshierarchie und die Attribute zum Analysemodellobjekt Buch wurden bereits bei den Beschreibungen der Abbildungen 3.4 und 3.5 im Abschnitt 3.4.2 vorgestellt. Bei der gewählten Art der Implementierung müssen in einem Datensatz sowohl die Attribute des Analysemodellobjektes Buch als auch die Attribute der Analysemodellobjekte der vererbenden Klassen Ausleihobjekt und Inventargegenstand berücksichtigt werden. Insgesamt sind also in einem Datensatz für ein Buch die Daten folgender Attribute enthalten: Inventarnummer, Inventarisierungsdatum, Signatur, Verfasser, Titel, Art, Ausgabe, Umfang, Erscheinungsjahr, Verlagsort, Verlagsname, Schlagwort, ISBN. Zusätzlich müssen zwei Referenzen auf Objekte der Klasse Leihfristenkombination gespeichert werden. Das Schlüsselattribut eines Buchdatensatzes ist die Inventarnummer. Als Datenstruktur zur Speicherung der Daten der Attribute der Verweise werden endliche Abbildungen (*maps*) genutzt. Dabei war ursprünglich geplant, endliche Abbildungen zu definieren, bei denen über Atome (*atoms*) [DFH+95, S. 23] auf die Daten eines Datensatzes zugegriffen werden kann. Für einen Datensatz zum Objekt Buch wäre dann die Abbildung *buchdatensatz*(x) gemäß Tabelle 3.4 definiert worden.

x	*buchdatensatz*(x)
inventarnummer	Inventarnummer des Buches
inventarisierungsdatum	Inventarisierungsdatum des Buches
signatur	Signatur des Buches
erste_leihfristen	Schlüssel für Datensatz zum Objekt Leihfristenkombination
verlaengerungen	Schlüssel für Datensatz zum Objekt Leihfristenkombination
verfasser	Verfasser des Buches
titel	Buchtitel
⋮	⋮
isbn	ISBN des Buches

Tabelle 3.4: Der Datensatz buchdatensatz, Zugriff über Atome.

Bei der Benennung der Atome hätte es sich angeboten, auf die Bezeichnungen der Attribute aus dem Objectory-Analysemodell zurückzugreifen. Damit wäre eine sehr elegante Programmierung möglich gewesen. Bei Testläufen erwies sich der Zugriff auf Daten über Atome jedoch aus dem folgenden Grund als problematisch:

- Da für ProSet noch kein Debugger existiert, mit dem die Inhalte verwendeter Variablen angezeigt werden können, ist es zur Fehlersuche häufig notwendig, Variableninhalte während des Programmablaufs auf dem Bildschirm auszugeben. Insbesondere die Ausgabe des Inhalts endlicher Abbildungen war in dieser Diplomarbeit wichtig, weil neben Fehlern in der eigenen Implementierung noch Fehler des ProSet-Compilers und des H-PCTE-Servers bei der Benutzung endlicher Abbildungen deutlich wurden, die parallel zur Implementierung des Prototyps für das betrachtete Fallbeispiel durch Mitglieder der ProSet-Entwicklergruppe behoben wurden (H-PCTE ist das Datenbanksystem, das zur Realisierung der Persistenz benutzt wird).

  Die endlichen Abbildungen können mit Hilfe des ProSet-Befehls putf [DFH+95, S. 76ff.] ausgegeben werden. An den Stellen, an denen Atome vorkommen, werden bei einer Benutzung des Befehls putf auf dem Bildschirm 28-stellige Zeichenfolgen angezeigt. Durch diese langen Zeichenfolgen werden die Bildschirmausgaben zum einen bei großen endlichen Abbildungen unübersichtlich. Zum anderen kann von den ausgegebenen Zeichenfolgen nicht direkt auf die Atome geschlossen werden, die durch die Zeichenfolgen repräsentiert werden. Durch die Benutzung von ganzen Zahlen anstelle von Atomen konnte die Fehlersuche vereinfacht werden.

Anstelle von Atomen wird bei der realisierten Implementierung über ganze Zahlen (Datentyp integer) auf die benötigten Daten zugegriffen. Die Abbildung *buchdatensatz*(x) ist in der Tabelle 3.5 dargestellt. Bei dieser Tabelle fällt zunächst auf, daß die Inventarnummer nicht aufgenommen wurde. Zusätzlich müssen die Einträge *Lesezugriffe* (Nr. 1) und *Schreibzugriff* (Nr. 2) erläutert werden. Diese Punkte werden im folgenden behandelt:

Inventarnummer: Die Inventarnummer ist das Schlüsselattribut zum Datensatz buch. Ein gesuchter Datensatz wird anhand seines Schlüsselattributes identifiziert. Daher erhält das Schlüsselattribut eine besondere Position, um die Zugriffe auf einen Datensatz zu beschleunigen. Zunächst war dabei die folgende Art der Realisierung geplant:

Es wird ein P-File erzeugt, in dem alle benötigten Datensätze gespeichert sind. Für jeden Datensatz existiert ein P-File-Eintrag. Der Name eines solchen P-File-Eintrags besteht aus dem Namen des Objektes, zu dem dieser Datensatz gehört und einer Endung, die durch den Eintrag für das Schlüsselattribut gebildet wird. Der Datensatz zum Buch mit der Inventarnummer 25 hätte also zum Beispiel im P-File die Bezeichnung buch25 (vor Beginn der Implementierung wurde festgelegt, Variablennamen grundsätzlich klein zu schreiben).

x	$buchdatensatz(x)$	Datentyp
1	Lesezugriffe	tuple
2	Schreibzugriff	integer
3	Inventarisierungsdatum	string
4	Schlüssel für Datensatz zum Objekt Leihfristenkombination	integer
5	Schlüssel für Datensatz zum Objekt Leihfristenkombination	integer
6	Signatur	string
7	Verfasser	tuple
8	Buchtitel	string
9	Erscheinungsjahr	integer
10	Ausgabe des Buches	string
11	Umfang des Buches	string
12	Art des Buches	string
13	Verlagsort	string
14	Verlag	string
15	Schlagworte	tuple
16	ISBN des Buches	string

Tabelle 3.5: Der Datensatz buchdatensatz, Zugriff über ganze Zahlen.

Es kann vorkommen, daß für einzelne Einträge in einem Datensatz durch den Benutzer kein Wert spezifiziert wird (z.B. für die Signatur). Diese Einträge besitzen dann den (undefinierten) Wert om. Die verwendeten Datentypen und der undefinierte Wert om werden bei Doberkat et al. beschrieben [DFH+95, S. 19ff.].

Diese Vorgehensweise läßt sich jedoch nicht realisieren, weil die Namen benötigter P-File-Einträge im ProSet-Programm Variablennamen entsprechen. Variablennamen müssen vor der Übersetzung eines Programms festgelegt werden. Zu diesem Zeitpunkt sind die Daten der Schlüsselattribute jedoch nicht bekannt. Um diese Probleme zu umgehen, werden die Datensätze folgendermaßen gespeichert:

Für jede implementierte Objektklasse existiert ein P-File-Eintrag mit dem Namen dieser Klasse. Bei einem solchen P-File-Eintrag handelt es sich um eine persistente Variable, die eine endliche Abbildung enthält. Dabei wird über die Daten der Schlüsselattribute auf die einzelnen Datensätze zugegriffen.

Bei dem hier exemplarisch betrachteten Beispiel, der Klasse Buch, existiert demnach ein P-File-Eintrag buch. Dabei handelt es sich um eine Abbildung, bei der über die Inventarnummer auf jeden einzelnen Datensatz mit den Daten eines Buches gemäß Tabelle 3.5 zugegriffen werden kann:

$$buch : \begin{cases} \mathsf{integer} & \rightarrow & \mathsf{map} \\ inventarnummer & \mapsto & buchdatensatz. \end{cases}$$

Diese Art der Implementierung hat den Nachteil, daß bei einem Zugriff auf den Datensatz eines Buches auch immer die Datensätze aller anderen Bücher geladen werden müssen. Dies soll beispielhaft anhand der Prozedur ausmusterung demonstriert werden:

Beispiel:

```
procedure ausmusterung(rd inventarnummer);
---------------------------------------------------------------------
-- Beschreibung:     Durch diese Prozedur wird der Eintrag eines Buches
--                   entfernt.
-- Aufrufparameter:  inventarnummer: Inventarnummer fuer zu entfernendes
--                   Buch.
-- Rueckgabewert:    -
---------------------------------------------------------------------
hidden persistent
   buch: daten_p_file;
begin
   buch := buch lessf inventarnummer;
      ---------------------------------------------------------------
      -- Die Kurzschreibweise   buch lessf := inventarnummer   wurde zum
      -- Zeitpunkt der Implementierung dieser Prozedur vom ProSet-Compiler
      -- noch nicht akzeptiert.
```

```
    ------------------------------------------------------------------
end ausmusterung;
```

Obwohl nur der Datensatz des Buches mit der übergebenen Inventarnummer gelöscht werden soll, müssen die Daten sämtlicher Bücher, die in der endlichen Abbildung buch gespeichert sind, geladen werden. Da sich die Prozedur ausmusterung in einem Modul befindet, muß die Abbildung buch als lokale persistente Variable geladen werden. Globale persistente Variablen sind in Moduln verboten.

Dadurch, daß bei einem Zugriff auf den Datensatz eines Buches auch immer die Datensätze aller anderen Bücher geladen werden müssen, sind die Möglichkeiten zum parallelen Datenzugriff eingeschränkt. Auf diese Problematik wird im Abschnitt 3.6 noch genauer eingegangen.

Da die Namen der P-File-Einträge nicht dynamisch zur Laufzeit des Programms festgelegt werden können, ließe sich das Problem, daß die Daten aller Bücher geladen werden müssen, wenn auf die Daten eines Buches zugegriffen werden soll, nur dadurch umgehen, daß mit einem entsprechenden Werkzeug für jedes Buch ein eigenes P-File angelegt wird, in dem sich die Daten dieses Buches befinden. Die Namen der P-Files, auf die zugegriffen wird, können im Gegensatz zu den Namen der P-File-Einträge zur Laufzeit des Programms bestimmt werden. Diese Art der Implementierung erschien für eine Realisierung jedoch zu praxisfern.

Dennoch hat sich durch die Entwicklung des Prototyps gezeigt, daß das realisierte Konzept verbesserungsfähig ist. So wäre es zur Steigerung der Geschwindigkeit sinnvoll gewesen, Zugriffsprozeduren und Datenstrukturen im selben Modul zu kapseln. Für jede Materialklasse existierte in diesem Fall eine genau eine persistente Modulinstanz, in der Zugriffsprozeduren und alle benötigten Daten der Objekte dieser Klasse gespeichert wären. Die Geschwindigkeit wird bei dieser Art der Realisierung deshalb verbessert, weil häufig mehrere Prozeduren zum Datenzugriff, die sich im selben Modul befinden, innerhalb einer Prozedur einer Werkzeugklasseninstanz aufgerufen werden (auf die Implementierung der Werkzeugklassen wird im Abschnitt 3.5.2 eingegangen). Bei der implementierten Trennung von Zugriffsoperationen und Datensätzen müssen alle Datensätze des P-File-Eintrags, auf den zugegriffen werden soll, bei jedem Aufruf einer Zugriffsoperation vom persistenten Datenspeicher geladen werden, weil globale persistente Variablen und Konstanten in Moduln verboten sind. Bei der vorgestellten Verbesserung werden diese Datensätze nur einmal beim Start der Prozedur der Werkzeugklasseninstanz geladen.

Beispiel: Wenn innerhalb einer Prozedur, in der der Ablauf eines Anwendungsfalls kontrolliert wird, hintereinander die Prozedur buchdaten_lesen und die Prozedur ausmusterung aufgerufen werden, müssen die Daten sämtlicher Bücher sowohl beim Aufruf der Prozedur buchdaten_lesen als auch beim Aufruf der Prozedur ausmusterung vom Datenspeicher geladen werden, obwohl sich die Instanz des Moduls Buch mit den Zugriffsoperationen die gesamte Zeit im Hauptspeicher befindet (auf die Abarbeitung der Anwendungsfälle wird im Abschnitt 3.5.2 detaillierter eingegangen). Wären die Daten der Bücher zusammen mit den Zugriffsoperationen in einer Instanz des Moduls Buch gespeichert, wären diese Daten zusammen mit den Zugriffsoperationen geladen worden. Beim Aufruf der Prozedur buchdaten_lesen und der Prozedur ausmusterung müßte dann nicht mehr auf den persistenten Datenspeicher zugegriffen werden. Da die Daten, auf die zugegriffen wird, nur einmal statt zweimal geladen werden müßten, ergäbe sich eine Zeitersparnis.

Diese Überlegungen wurden vor der Implementierung des Prototyps nicht angestellt, da zu diesem Zeitpunkt noch nicht berücksichtigt wurde, daß die Namen der P-File-Einträge bereits vor der Laufzeit des Prototyps festgelegt werden müssen. Die Kriterien, die zur Entscheidung geführt haben, Daten separat zu speichern, wurden im Teilabschnitt *Konzepte zur Abbildung der Materialklassen auf ProSet* vorgestellt.

Lesezugriffe und Schreibzugriff: Diese Einträge waren zur Vermeidung von Zugriffskonflikten geplant, falls mehrere Benutzer gleichzeitig mit denselben Daten arbeiten.

Durch die Einträge Lesezugriffe und Schreibzugriff sollten Transaktionen ermöglicht werden, die sich über mehrere Prozeduren erstrecken. Prozeduren der Werkzeugklassen sollten Datensätze vorübergehend für Lese- und Schreibzugriffe sperren können. Wenn ein Benutzer einen Datensatz liest, um Daten zu verändern, kann so sichergestellt werden, daß der betroffene Datensatz nicht durch einen zweiten Benutzer mainipuliert wird, bevor der erste Benutzer die Daten wieder freigegeben hat.

Die Entwicklung des Prototyps hat jedoch gezeigt, daß sich das im folgenden vorgestellte Konzept zur Überwachung paralleler Zugriffe durch den erstellten Prototyp nicht realisieren läßt. Dies liegt daran, daß die Möglichkeiten für einen parallelen Zugriff auf Daten stärker eingeschränkt sind, als dies ursprünglich angenommen wurde. Hierauf wird im Anschluß an die Beschreibung des Konzeptes eingegangen.

In dem Modul, in dem die Zugriffsprozeduren auf die Datensätze für Bücher gesammelt sind, werden die Einträge Lesezugriffe und Schreibzugriff mit Hilfe der Prozeduren lesen_anmelden, lesen_abmelden, schreiben_anmelden und schreiben_abmelden manipuliert. Die Prozeduren lesen_anmelden und schreiben_anmelden werden von Instanzen der Werkzeugklassen heraus aufgerufen, wenn ein Datensatz gelesen bzw. manipuliert werden soll. Die Prozeduren lesen_abmelden und schreiben_abmelden werden von Instanzen der Werkzeugklassen heraus aufgerufen, wenn die Lesevorgänge bzw. die Manipulationen eines Datensatzes abgeschlossen sind. Die Prozeduren erfüllen im einzelnen die folgenden Aufgaben:

lesen_anmelden

Beschreibung:	In dem Tupel, in dem sich die Leseeinträge befinden, wird ein neuer Eintrag erzeugt. Durch den Eintrag wird angezeigt, daß ein Datensatz gelesen wird. Dieser Eintrag kann nur dann eingefügt werden, wenn auf den ausgewählten Datensatz nicht zur selben Zeit bereits schreibend zugegriffen wird. Gegenseitiger Ausschluß wird durch den ProSet-Transaktionsmechanismus gewährleistet.
Aufrufparameter:	Inventarnummer für gewählten Datensatz.
Rückgabewert:	0: Auftrag ausgeführt. 1: Eintrag nicht möglich, weil gesuchter Datensatz nicht existiert. 2: Eintrag nicht möglich, weil für diesen Datensatz bereits ein Eintrag zum Schreiben besteht.

lesen_abmelden

Beschreibung:	Durch diese Prozedur wird ein Eintrag aus dem Tupel, in dem sich die Leseeinträge befinden, gelöscht. Wenn das Tupel bereits leer ist, wird das Tupel nicht verändert. Gegenseitiger Ausschluß wird durch den ProSet-Transaktionsmechanismus gewährleistet.
Aufrufparameter:	Inventarnummer für gesuchten Datensatz.
Rückgabewert:	0: Auftrag ausgeführt. 1: Inventarnummer nicht gefunden.

schreiben_anmelden

Beschreibung:	Durch diese Prozedur wird eingetragen, daß ein Datensatz geändert oder modifiziert wird. Gegenseitiger Ausschluß wird durch den ProSet-Transaktionsmechanismus gewährleistet.
Aufrufparameter:	Inventarnummer für gewählten Datensatz.
Rückgabewert:	0: Auftrag ausgeführt. 1: gesuchter Datensatz wurde nicht gefunden. 2: Eintrag ist nicht möglich, weil noch Leseeinträge vorhanden sind. Der Aspekt der „Fairness“ zwischen verschiedenen Prozessen wird bei der Implementierung des Prototyps vereinfachend außer acht gelassen. Bei einer Implementierung des Systems, mit dem die Endanwender arbeiten sollen, müßte dieser Aspekt zusätzlich berücksichtigt werden. Das Auftreten von Livelocks müßte verhindert werden. 3: Eintrag ist nicht möglich, weil ein Schreibeintrag bereits existiert.

schreiben_abmelden

Beschreibung:	Der Schreibeintrag wird wieder auf „frei“ gesetzt. Gegenseitiger Ausschluß wird durch den ProSet-Transaktionsmechanismus gewährleistet.
Aufrufparameter:	Inventarnummer für gesuchten Datensatz.
Rückgabewert:	0: Auftrag ausgeführt. 1: Datensatz nicht gefunden.

Da ein Datensatz von mehreren Programinstanzen gleichzeitig gelesen aber nur von einer Programminstanz gleichzeitig beschrieben werden darf, besteht der Eintrag für die Lesezugriffe aus einem Tupel, während der Eintrag für die Schreibzugriffe aus einer einzelnen Zahl besteht (siehe Tabelle 3.5).

Bei der Implementierung der Werkzeugklassen (siehe Abschnitt 3.5.2) hat sich jedoch gezeigt, daß diese Einträge überflüssig sind, da die ProSet-Lese- und -Schreibsperren für P-File-Einträge, die durch das ProSet-Laufzeitsystem verwaltet werden (siehe hierzu Doberkat et al.[DFH+95, S. 59ff.]), bei der Ausführung von Prozeduren der Materialinstanzen länger gesetzt bleiben, als dies ursprünglich erwartet wurde. Dies wird im folgenden näher erläutert:

Da Materialinstanzen vor ihrer Ausführung vom persistenten Datenspeicher geladen werden müssen (siehe Seite 42), wird in den Prozeduren, die auf Operationen in den Materialinstanzen zugreifen, bereits der ProSet-Persistenzmechanismus benutzt. Operationen sind hier Prozeduren der Materialinstanzen.

Dies führt dazu, daß ProSet-Sperren für P-File-Einträge, die bei der Ausführung der Operationen der Materialinstanzen durch das ProSet-Laufzeitsystem gesetzt werden, bei Beendigung dieser Operationen an die aufrufenden Prozeduren (genauer: an die Elterntransaktionen) weitergegeben werden.

Beim Entwurf des Prototyps wurde davon ausgegangen, daß die Sperren nicht weitergegeben werden. Es wurde erwartet, daß eine ProSet-Lese- oder Schreibsperre bereits zurückgesetzt wird, wenn die Materialinstanzoperation, bei deren Ausführung das Setzen dieser Sperre durch das ProSet-Laufzeitsystem veranlaßt wurde, beendet wird.

Beispiel: In der persistenten Materialinstanz buch befindet sich die Prozedur buchdaten_modifizieren. In dieser Prozedur wird schreibend auf den Datensatz buch zugegriffen. Die persistente Variable für den Datensatz buch und die persistente Modulinstanz buch befinden hier sich in verschiedenen P-Files. Wenn die Prozedur buchdaten_modifizieren ausgeführt wird, wird durch das ProSet-Laufzeitsystem für den Datensatz buch eine Schreibsperre gesetzt.

Damit die Prozedur buchdaten_modifizieren von einer Prozedur eines fremden Moduls aufgerufen werden kann, wird in der Prozedur des fremden Moduls, die auf die Prozedur buchdaten_modifizieren zugreift, zunächst die Materialinstanz buch vom persistenten Datenspeicher geladen. Diese Nutzung des Persistenzmechanismus in der aufrufenden Prozedur führt jedoch dazu, daß die Schreibsperre für den Datensatz buch durch das ProSet-Laufzeitsystem bei der Beendigung der Prozedur buchdaten_modifizieren nicht zurückgenommen wird, was ursprünglich erwartet wurde. Die Schreibsperre wird stattdessen an die aufrufende Prozedur (genauer: an die Elterntransaktion) weitergegeben.

Eine Übersicht über alle Zugriffsoperationen des Moduls Buch auf die benötigten Daten wird in der Tabelle 3.6 gegeben. Der ProSet-Code dieses Moduls befindet sich neben dem Code der anderen implementierten Moduln im Zusatzdokument zur Diplomarbeit. Die Vorgehensweise bei der Implementierung der anderen Moduln verläuft analog zu der hier vorgestellten Vorgehensweise.

Die Tabelle 3.3 dient als Grundlage für die zu implementierenden Moduln. Nicht implementiert werden Moduln mit Zugriffsoperationen für Daten von Objekten vererbender Klassen. Hiervon sind die vererbenden Klassen Ausleihobjekt, Inventargegenstand, Katalog und Person betroffen Für vererbende Klassen müssen keine Moduln implementiert werden, weil keine Instanzen dieser Klassen benötigt werden. Hierauf wurde bereits im Teilabschnitt *Konzepte zur Abbildung der Materialklassen auf* ProSet eingegangen.

Des weiteren werden keine Moduln zu den Klassen Ausleihkartei, Benutzerdatenbank, Bücherkatalog, Gerätedatenbank und Zeitschriftenkatalog implementiert, da Datensätze für Objekte dieser Klassen keine Informationen enthielten, die noch nicht in anderen Datensätzen gespeichert wären.

Name	Beschreibung
lesen_anmelden	Es wird eingetragen, daß ein Datensatz gelesen wird.
schreiben_anmelden	Es wird eingetragen, daß ein Datensatz modifiziert oder gelöscht werden soll.
lesen_abmelden	Ein Leseeintrag wird entfernt.
schreiben_abmelden	Der Schreibeintrag wird wieder zurückgesetzt.
inventarisierung	Ein neues Buch wird aufgenommen.
ausmusterung	Ein Buch wird ausgemustert.
inventarisierungsdatum_lesen	Das Inventarisierungsdatum eines ausgewählten Buches wird gelesen.
fristen_lesen	Die Verweise auf Objekte der Klasse Leihfristenkombination, in denen die die Leihfrist und erlaubte Fristverlängerungen für ein ausgewähltes Buch gespeichert sind, werden zurückgegeben.
fristen_modifizieren	Die Verweise auf Objekte der Klasse Leihfristenkombination, in denen die die Leihfrist und erlaubte Fristverlängerungen für ein ausgewähltes Buch gespeichert sind, werden geändert.
signatur_lesen	Die Signatur eines ausgewählten Buches wird gelesen.
signatur_modifizieren	Die Signatur eines ausgewählten Buches wird modifiziert.
buchdaten_lesen	Die Daten der Attribute eines ausgewählten Buches, die nicht von Vorgängern geerbt wurden, werden gelesen.
buchdaten_modifizieren	Die Daten der Attribute eines ausgewählten Buches, die nicht von Vorgängern geerbt wurden, werden überschrieben.
suche	Es werden Bucheinträge gesucht, die die gewünschten Suchkriterien erfüllen. Werden mehrere Suchkriterien angegeben, wird UND-Verknüpft. Es kann nach den folgenden Kriterien gesucht werden: Signatur, Verfasser, Titel, Erscheinungsjahr, Verlag, Stichworte, ISBN. Es wird eine Menge mit den Inventarnummern der Bücher zurückgeliefert, die die übergebenen Suchkriterien erfüllen. Die Suche nach Büchern hätte auch ausschließlich auf Prozeduren der Werkzeugklassen verlagert werden können. Sie wird in diesem Fall direkt durch ein Modul für Materialien unterstützt, um die Geschwindigkeit bei der Suche zu erhöhen.

Tabelle 3.6: Prozeduren des Moduls Buch.

3.5.2 Werkzeuge

Werkzeugklassen werden auf der Grundlage des Objectory-Anwendungsfallmodells (siehe Abschnitt 3.4.1) entwickelt. Dabei orientieren sich die Werkzeugklassen an den Schnittstellen, die von den Materialklassen angeboten werden. Auf die Implementierung von Aspektklassen wird aus Zeitgründen verzichtet.

Zusammengehörende Anwendungsfälle werden einer Werkzeugklasse zugeordnet. Die Entscheidung, welche Anwendungsfälle zusammengehören, hängt davon ab, welche Datensätze in diesen Anwendungsfällen bearbeitet werden. Im folgenden wird die gewählte Einteilung vorgestellt. Dabei werden jeweils die Anwendungsfälle aufgeführt, die einer Klasse zugeordnet wurden (siehe auch Tabelle 3.1 im Abschnitt 3.4.1).

Bibliotheksbenutzerdaten bearbeiten: Neuen Bibliotheksbenutzer aufnehmen, Bibliotheksbenutzerdaten ansehen, Bibliotheksbenutzerdaten modifizieren, Daten eines Bibliotheksbenutzers löschen, Einhaltung der Leihfristen kontrollieren.

Buch bearbeiten: Neues Buch aufnehmen, Daten eines Buches modifizieren, Löschen der Daten eines Buches.

Entleihung bearbeiten: Ausleihobjekte ausgeben, Ausleihobjekt reservieren, Reservierungen stornieren, Leihfrist verlängern, Ausleihobjekte zurücknehmen.

Gerät bearbeiten: Gerätedaten aufnehmen, Gerätedaten ändern, Gerätedaten ansehen, Gerätedaten löschen.

Literaturrecherche Bücher: Literaturrecherche Bücher.

Literaturrecherche Zeitschriften: Literaturrecherche Zeitschriften.

Mitarbeiterdaten bearbeiten: Neuen Mitarbeiter aufnehmen, Mitarbeiterdaten modifizieren, Mitarbeiterdaten ansehen, Mitarbeiterdaten löschen.

System einrichten: Eintragen der Leihfristen, Beschreibung erreichbarer Bücherkataloge, Beschreibung erreichbarer Zeitschriftenkataloge.

Zeitschrift bearbeiten: Neues Zeitschriftenheft aufnehmen, Daten eines Zeitschriftenheftes modifizieren, Löschen der Daten eines Zeitschriftenheftes, Zeitschriftenband als Ausleihobjekt aufnehmen, Daten eines Zeitschriftenbandes modifizieren, Daten eines Zeitschriftenbandes löschen.

Für jede der neun Klassen wird ein Modul erzeugt. Diese Moduln werden im folgenden als *Werkzeugmoduln* bezeichnet. Für jeden aufgeführten Anwendungsfall wird eine Prozedur implementiert, durch die die Durchführung dieses Anwendungsfalls gesteuert wird. In einer solchen Prozedur werden die Zugriffe auf die benötigten Materialien (siehe Abschnitt 3.5.1) und die Interaktionen mit dem Benutzer koordiniert. Der Anwendungsfall Neues Buch aufnehmen wird z.B. durch die Prozedur neue_aufnahme ausgeführt, die sich im Werkzeugmodul Buch_bearbeiten befindet.

Sämtliche Ein-/Ausgabeoperationen, die Prozeduren in den Werkzeugmoduln für die Kommunikation mit dem Benutzer benötigen, sind in einem Modul Benutzerschnittstelle gekapselt.

3.5.3 Aufbau des Prototyps

Zu jeder der im Abschnitt 3.5.2 beschriebenen Werkzeugklasssen ist ein Programm implementiert, in dem abgefragt wird, welcher Anwendungsfall ausgeführt werden soll. Anschließend wird aus diesen Programmen die zur Ausführung des gewählten Anwendungsfalls benötigte Prozedur des zugehörigen Werkzeugmoduls aufgerufen. Die Namen der Programme und die Anwendungsfälle, die in den Programmen gewählt werden können, sind in der Tabelle 3.7 aufgeführt.

Programmname	Anwendungsfälle
bibliotheksbenutzer_bearbeiten	Neuen Bibliotheksbenutzer aufnehmen, Bibliotheksbenutzerdaten ansehen, Bibliotheksbenutzerdaten modifizieren, Daten eines Bibliotheksbenutzers löschen, Einhaltung der Leihfristen kontrollieren
buch_bearbeiten	Neues Buch aufnehmen, Daten eines Buches modifizieren, Löschen der Daten eines Buches
entleihung_bearbeiten	Ausleihobjekte ausgeben, Ausleihobjekt reservieren, Reservierungen stornieren, Leihfrist verlängern, Ausleihobjekte zurücknehmen
geraet_bearbeiten	Gerätedaten aufnehmen, Gerätedaten ändern, Gerätedaten ansehen, Gerätedaten löschen
literaturrecherche_buecher	Literaturrecherche Bücher
literaturrecherche_zeitschriften	Literaturrecherche Zeitschriften
mitarbeiter_bearbeiten	Neuen Mitarbeiter aufnehmen, Mitarbeiterdaten modifizieren, Mitarbeiterdaten ansehen, Mitarbeiterdaten löschen
einrichten	Eintragen der Leihfristen, Beschreibung erreichbarer Bücherkataloge, Beschreibung erreichbarer Zeitschriftenkataloge
zeitschrift_bearbeiten	Neues Zeitschriftenheft aufnehmen, Daten eines Zeitschriftenheftes modifizieren, Löschen der Daten eines Zeitschriftenheftes, Zeitschriftenband als Ausleihobjekt aufnehmen, Daten eines Zeitschriftenbandes modifizieren, Daten eines Zeitschriftenbandes löschen

Tabelle 3.7: Programme und zugehörige Anwendungsfälle.

Zur Speicherung der benötigten persistenten Daten (siehe Abschnitt 3.5.1) werden zwei P-Files benötigt:

P-File Daten_der_Bibliothek: Hier werden die Daten, die zu den benötigten Objekten gespeichert sind, abgelegt.

P-File Bibliothek_Materialinstanzen: Hier werden die Instanzen der Moduln gespeichert, in denen die Zugriffsoperationen für die benötigten Daten implementiert sind.

Die P-Files und die benötigten P-File-Einträge müssen vor der Ausführung des Prototyps erzeugt werden. Die P-Files können mit einem externen Werkzeug, wie z.B. dem xpft angelegt werden. Die benötigten Einträge der P-Files können mit Hilfe von zusätzlichen ProSet-Programmen erzeugt werden (siehe Beschreibung der Installation des Prototyps im Zusatzdokument zur Diplomarbeit).

Zusätzlich können weitere P-Files erzeugt werden, um die Literatursuche in den Datenbeständen fremder Bibliotheken durch den Prototyp zu simulieren. In den zusätzlichen P-Files müssen Modulinstanzen mit Prozeduren vorhanden sein, die zur Literatursuche und zum Lesen von Daten gefundener Bücher oder Zeitschriften die gleichen Schnittstellen anbieten, wie die Prozeduren der Instanzen im P-File Bibliothek_Materialinstanzen. Modulinstanzen zur Suche nach Zeitschriften haben den Namen zeitschrift, Modulinstanzen zur Suche nach Büchern haben den Namen buch. Die Namen der Zugriffsoperationen müssen mit den Namen der Operationen aus den Instanzen des P-Files Bibliothek_Materialinstanzen übereinstimmen. Um die simulierten fremden Datenbanken dem Prototyp zur Literatursuche zugänglich zu machen, muß der Anwendungsfall Beschreibung erreichbarer Bücherkataloge bzw. der Anwendungsfall Beschreibung erreichbarer Zeitschriftenkataloge ausgeführt werden.

Beispiel: Um eine Literaturrecherche im Bücherbestand einer fremden Bibliothek zu simulieren, wird ein neues P-File erzeugt. Dieses P-File soll im folgenden Neues_P_File heißen. Im P-File Neues_P_File muß sich ein Eintrag buch befinden, in dem eine Modulinstanz gespeichert ist. Die Prozeduren zum Zugriff auf die Daten der fremden Bibliothek besitzen die gleichen Namen und bieten die gleichen Schnittstellen an wie die Prozeduren in der Modulinstanz buch im P-File Bibliothek_Materialinstanzen.

Eine detaillierte Beschreibung aller Prozeduren sowie deren Schnittstellen, hätte mehr Platz beansprucht, als für die Beschreibung dieses Fallbeispiels innerhalb der Diplomarbeit zur Verfügung steht. Auf die Prozeduren und auf die Spezifikation der Schnittstellen wird im folgenden deshalb nicht weiter eingegangen. Der ProSet-Code der implementierten Programme und Moduln ist im Zusatzdokument zu dieser Diplomarbeit aufgeführt.

3.6 Fazit

In diesem Abschnitt werden zum Abschluß dieses Fallbeispiels die gewonnen Erkenntnisse zusammengefaßt. Dazu werden zunächst noch einmal wesentliche Punkte, die bereits in den Abschnitten 3.4 und 3.5 erwähnt wurden, kurz wiederholt:

Analyse und Entwurf (Abschnitt 3.4)

- Die Metaphern Werkzeug und Material sollten nach der ursprünglichen Planung zur Strukturierung des Problembereichsmodells verwendet werden. Bei der Erstellung des Problembereichsmodells wurde deutlich, daß die Identifizierung von Werkzeugen eine ausführlichere Ist-Analyse vorausgesetzt hätte, als dies zeitlich im Rahmen der Diplomarbeit möglich war. Das Fehlen von Werkzeugobjekten im Problembereichsmodell hat sich jedoch nicht negativ auf die Entwicklung des Prototyps ausgewirkt.

  Probleme ergeben sich immer dann, wenn mit dem zu entwickelnden System Tätigkeiten ausgeführt werden sollen, die im betrachteten Anwendungsbereich bislang nicht ausgeführt werden. In diesem Fall können für diese neuen Tätigkeiten keine Materialien aus dem bisherigen Anwendungsbereich identifiziert werden, die bearbeitet werden können (z.B. um Informationen über Personen oder Geräte zu speichern).

  Diese Schwierigkeiten entstehen auch dann, wenn für neue Tätigkeiten, die mit Hilfe des zu entwickelnden Systems durchgeführt werden sollen, im betrachteten Anwendungsbereich bereits computergestützte Systeme bestehen, die selbst jedoch nicht auf der Basis der Metaphern Werkzeug und Material entwickelt wurden (siehe Abschnitt 3.4.1).

Implementierung (Abschnitt 3.5)

- Probleme ergeben sich bei der Übersetzung von Moduln, in deren Prozeduren persistente Variablen benutzt werden, wenn auf diese Moduln das closure-Konstrukt angewendet wird. Bei dem betrachteten Fallbeispiel kann immer nur ein solcher Modul gleichzeitig übersetzt werden. Andernfalls wird der Übersetzungsvorgang mit der Fehlermeldung internal assembler error: out of memory abgebrochen.

- Es wäre bei die Implementierung des Prototyps nützlich gewesen, wenn die Namen für benötigte P-File-Einträge erst zur Programmlaufzeit festgelegt werden könnten.

  Die Auflage, diese Namen bereits bei der Implementierung zu bestimmen, führt bei dem implementierten Prototyp dazu, daß die Daten aller Objekte einer Klasse im selben P-File-Eintrag gespeichert werden. So werden z.B. die Daten aller Bücher im P-File-Eintrag buch gespeichert. Dadurch ergibt sich der Nachteil, daß sich die Daten sämtlicher Objekte einer Klasse im Hauptspeicher befinden müssen, wenn auf die Daten eines Objektes dieser Klasse zugegriffen werden soll.

- Weil sich bei dem betrachteten Prototyp die Daten sämtlicher Objekte einer Klasse in einem P-File-Eintrag befinden müssen (siehe Abschnitt 3.5.1), sind die Möglichkeiten zum parallelen Datenzugriff eingeschränkt. Dies soll im folgenden erläutert werden: Wenn eine Instanz eines Programms schreibend auf die Daten eines Objektes zugreift, können keine Daten eines beliebigen anderen Objektes dieser Klasse mehr von einer zweiten Programminstanz gelesen werden, solange die erste Instanz die Schreibsperre behält. Ebenso kann eine Programminstanz erst dann schreibend auf die Daten eines Objektes einer Klasse zugreifen, wenn keine andere Instanz eines Programms, von dem aus bereits lesend auf ein beliebiges Objekt derselben Klasse zugegriffen wurde, eine Lesesperre für die Objekte dieser Klasse mehr besitzt. Dies soll anhand des P-File-Eintrags buch verdeutlicht werden:

  Beispiel: Im P-File Eintrag buch werden die Daten aller Bücher gespeichert. Wenn eine Programminstanz die Daten eines beliebigen Buches x verändert, kann keine weitere Instanz auf die Daten eines anderen Buches y zugreifen, solange die erste Instanz die Schreibsperre behält.
  Ein ähnliches Problem ergibt sich, wenn bereits Daten eines Buches gelesen wurden. Eine Programminstanz kann erst dann die Daten eines Buches x verändern, wenn keine Programminstanz, in der schon einmal lesend auf die Daten eines beliebigen anderen Buches y zugegriffen wurde, mehr eine Lesesperre für den betreffenden P-File-Eintrag besitzt.

- Unter den gegebenen Voraussetzungen erweist sich die bei der Implementierung des Prototyps vorgenommene Trennung von Daten und Zugriffsoperationen als unzweckmäßig. Die Geschwindigkeit beim Datenzugriff könnte gesteigert werden, wenn die Datenstrukturen für alle Daten einer Klasse und die benötigten Zugriffsoperationen in einem Modul gekapselt wären.

- ProSet-Sperren [DFH+95, S. 59] werden bei dem implementierten Prototyp später zurückgesetzt als ursprünglich angenommen. Wenn in einer Prozedur A eine ProSet-Lese oder Schreibsperre gesetzt wird, wird diese Sperre an die Prozedur B, durch die die Prozedur A aufgerufen wurde, weitergegeben, wenn in B bereits der ProSet-Persistenzmechanismus benutzt wurde. Dies gilt auch dann, wenn sich die Prozeduren A und B in verschiedenen Modulinstanzen befinden.

Zusätzlich ergaben sich beim Prototyping noch weitere Ergebnisse, die bisher noch nicht erwähnt wurden:

- In der Abbildung 3.7 im Abschnitt 3.4.2 ist ein Teil des Analysemodells skizziert. Dabei sind die Objekte zeitschrift, zeitschriftenheft, zeitschriftenband, heftdaten und banddaten dargestellt. Bei der Implementierung des Prototyps wurde deutlich, daß es zur Steigerung der Zugriffsgeschwindigkeit auf die zugehörigen Daten sinnvoll wäre, zusätzlich zu den in der Abbildung 3.7 dargestellten Beziehungen, Objectory-„acquaintance"-Beziehungen zwischen den Objekten Heftdaten und Zeitschrift sowie zwischen den Objekten Banddaten und Zeitschrift (jeweils Beziehungen in beiden Richtungen) einzufügen.

- Bei Versuchen, parallel auf dieselben persistenten Daten schreibend zuzugreifen, erwies sich der Persistenzmechanismus als anfällig für Deadlocks. Deadlocks verursachten nicht die erwartete Ausnahmemeldung p_deadlock_occured sondern die Ausnahmemeldung p_fatal_error.

- Die Behandlung von Ausnahmen, die im Zusammenhang mit der Persistenz auftreten, durch Ausnahme-Handler funktioniert nicht, wenn die persistenten Variablen oder Konstanten, auf die sich die Ausnahmen beziehen, innerhalb von Prozeduren deklariert werden. In diesem Fall wird die Ausführung eines Programms nach dem Auftreten einer Ausnahme abgebrochen.

Kapitel 4

Zweites Fallbeispiel: Modellierung eines E-Mail-Systems

In diesem Kapitel wird die Modellierung eines E-Mail-Systems als Fallbeispiel zum experimentellen Prototyping betrachtet. Es wird ein Prototyp als Labormuster (siehe Abschnitt 1.2) erzeugt, der die experimentelle Erprobung des Entwurfs eines E-Mail-Systems ermöglichen soll.

Mit einem E-Mail-System können Benutzer Nachrichten verschicken und empfangen. Die Aktivitäten der verschiedenen Benutzer sind weitgehend unabhängig voneinander und müssen evtl. parallel zueinander ausgeführt werden. Deshalb böte es sich an, bei der Implementierung des ProSet-Prototyps die Möglichkeiten zur Parallelprogrammierung, die von ProSet zur Verfügung gestellt werden [DFH+95], zu nutzen. Mit den ProSet-Compiler-Versionen, die bei dieser Diplomarbeit zur Verfügung stehen, ist es jedoch nicht möglich, den ProSet-Persistenzmechanismus und die Möglichkeiten zur Parallelprogrammierung gleichzeitig zu verwenden. Da in dieser Diplomarbeit das Prototyping mit persistenten Daten betrachtet wird, wird auf die Nutzung der Möglichkeiten zur Parallelprogrammierung verzichtet (siehe auch Abschnitt 2.3.2). Dies führt dazu, daß Daten bei dem zu erstellenden Prototyp häufig zwischen verschiedenen Programminstanzen ausgetauscht werden müssen.

Die grundsätzliche Idee, ein E-Mail-System als Beispiel für eine Anwendung, in der persistente Daten benutzt werden, zu betrachten, wurde von Gamerman et al. [GLV92] übernommen. Das in der Diplomarbeit betrachtete E-Mail-System unterscheidet sich jedoch von dem bei Gamerman et al. zugrunde gelegten System. Hierauf wird in den Abschnitten 4.3 und 4.7 noch genauer eingegangen.

Im Vergleich zum System zur Bibliotheksverwaltung (siehe Kapitel 3) werden beim E-Mail-System weniger verschiedene Daten benötigt. Die Daten des E-Mail-Systems haben meist eine geringere Lebensdauer. Zwischen den beim E-Mail-System benötigten Daten bestehen weniger Beziehungen, die zur Erhaltung der Konsistenz des Datenbestandes beachtet werden müssen, als beim System zur Bibliotheksverwaltung.

Die Modellierung des E-Mail-Systems wird in diesem Kapitel in sieben Abschnitten vorgestellt. Im Abschnitt 4.1 wird die Vorgehensweise im Entwicklungsprozeß bis zur Erstellung des Prototyps dargelegt. Einen Überblick über den Aufbau von E-Mail-Systemen und eine Beschreibung Anforderungen an das zu modellierende System und den Prototyp geben die Abschnitte 4.2 und 4.3. In den Abschnitten 4.4 und 4.5 werden das Anforderungsmodell und das Analysemodell beschrieben. Auf die Erstellung des Objectory-Entwurfsmodells und des Prototyps wird im Abschnitt 4.6 eingegangen. Abschnitt 4.7 enthält eine Zusammenfassung der Ergebnisse der durchgeführten Arbeiten.

4.1 Vorgehensweise

Die Vorgehensweise bei der Ist- und bei der Anforderungsanalyse ist mit der Vorgehensweise, die bei der Erstellung des Prototyps für ein System zur Bibliotheksverwaltung angewendet wurde (siehe Abschnitt 3.1), vergleichbar.

Eine Beschreibung einzelner Szenarien, wie sie von Gryczan und Züllighoven [GZ92] sowie von Bäumer et al. [BGZ95] für die Ist-Analyse vorgeschlagen wird (siehe Abschnitt 2.2), hätte mehr Zeit in Anspruch genommen, als bei der Bearbeitung dieses Fallbeispiels im Rahmen der Diplomarbeit für die Ist-Analyse zur Verfügung steht. Deshalb wird analog zum ersten Fallbeispiel bei der Ist-Analyse nur eine allgemeine Beschreibung bestehender E-Mail-Systeme gegeben. Danach werden die Anforderungen an das zu entwickelnde System und den zu implementierenden ProSet-Prototyp aufgeführt.

Anschließend wird das Objectory-Anforderungsmodell entwickelt. Parallel zur Ist-Analyse, der Beschreibung der Anforderungen und der Erstellung des Anforderungsmodells wird ein Glossar mit Definitionen verwendeter Begriffe erstellt. Dieses Glossar befindet sich im Anhang der Diplomarbeit.

Aufbauend auf dem Anforderungsmodell wird das Analysemodell für das betrachtete E-Mail-System entwickelt. Das Analysemodell bildet die Grundlage für die Erstellung des Entwurfsmodells und des Prototyps. Der Prototyp dient zur Validierung bestehender Teile des Entwurfsmodells und soll Aufschlüsse für die weitere Entwicklung des Entwurfsmodells liefern. Hierauf wird detailliert im Abschnitt 4.6 eingegangen.

4.2 Ist-Analyse

Ein E-Mail-System dient dazu, elektronisch Nachrichten zwischen Computerbenutzern zu versenden. Wenn ein Sender und ein Empfänger einer Nachricht verschiedene Computer benutzen, müssen die nötigen Informationen zum Nachrichtentransfer zwischen den entsprechenden Computern ausgetauscht werden. Nachrichten, die per E-Mail verschickt werden, werden für den Empfänger solange gespeichert, bis sie weiterverarbeitet werden.

Die folgende Beschreibung eines E-Mail-Systems orientiert sich an der vom *CCITT* 1984 entwickelten und 1988 überarbeiteten Empfehlung *X.400*, die sich nach Tanenbaum immer mehr zum Standard für alle elektronischen Postsysteme entwickelt [Tan92, S. 663]. CCITT ist eine Kurzform für *Comité Consultatif International de Télégraphique et Téléphonique*. Das CCITT erarbeitet als internationaler beratender Ausschuß Empfehlungen zur Telefon- und Datenkommunikation. Die Begriffe E-Mail und elektronische Post werden in dieser Diplomarbeit synonym verwendet. Der Text in diesem Abschnitt ist auf die Darstellung wesentlicher Eigenschaften und auf die Beschreibung von Leistungen, die auch von einfachen E-Mail-Systemen erbracht werden, beschränkt.

X.400 orientiert sich an der Funktionsweise der Briefpost („gelbe Post“). Bei den zu übertragenden Daten wird zwischen dem *Umschlag* (*envelope*) und dem *Inhalt* (*content*) einer Nachricht unterschieden. Für den Transport einer Nachricht vom Absender zum Empfänger sorgt das *Nachrichtenübertragungssystem*. Hierzu interpretiert das Nachrichtenübertragungssystem die Information des Umschlags. Neben den E-Mail-Adressen des Empfängers und des Absenders kann der Umschlag zusätzliche Informationen, wie z.B. Angaben über die Art des Nachrichteninhalts (Text, Grafik, digitalisierte Stimme, ...) oder Informationen über die Vertraulichkeitsstufe des Nachrichteninhalts (streng vertraulich, vertraulich, ...) enthalten. Einen Überblick über oft verwendete zusätzliche Angaben auf dem Umschlag gibt Tanenbaum [Tan92, S. 677].

Der Nachrichteninhalt kann in den *Kopf* (*heading*) und in den *Körper* (*body*) unterteilt werden. Der Kopf enthält in standardisierter Form alle wesentlichen Verwaltungsinformationen, die zur Einordnung und Weiterverarbeitung der Nachricht durch den Empfänger nötig sind. Hierzu gehören z.B. die Adresse des Absenders und die Angabe eines Betreffs. Eine Übersicht über einige mögliche Felder des Nachrichtenkopfes gibt wiederum Tanenbaum [Tan92, S. 673].

Der Körper enthält die Information, die der Sender dem Empfänger übermitteln möchte. Hierbei kann es sich um beliebige Daten (z.B. Text, Grafik, digitalisierte Stimme) handeln.

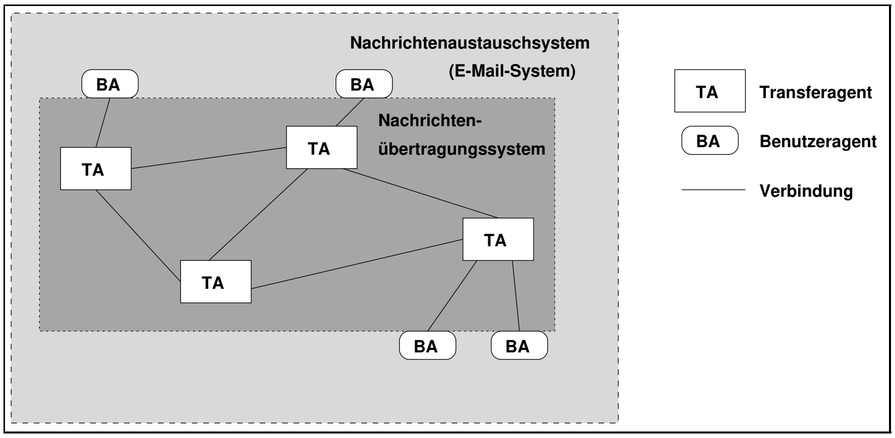

Abbildung 4.1: Modell eines Nachrichtenaustauschsystems.

Ein Beispiel für ein System, mit dem die Nachrichten verarbeitet und ausgetauscht werden können, ist in Abbildung 4.1 skizziert. Zur Beschreibung dieses Systems werden zunächst die Begriffe *Benutzeragent* (*user agent*) und *Transferagent* (*message transfer agent*) erläutert.

Benutzeragent: Ein Benutzeragent ist ein Programm, das die Schnittstelle zwischen dem Benutzer und dem Nachrichtenübertragungssystem bildet [Tan92, S. 667]. X.400 beschreibt die folgenden Mindestanforderungen an einen Benutzeragenten [BBP90, S. 27]:

- Der Benutzer kann mit Hilfe des Benutzeragenten einen Text so vorbereiten, daß er dem Nachrichtenübertragungssystem übergeben werden kann.
- Der Benutzer kann dem Benutzeragenten den gewünschten Empfänger seiner Nachricht und evtl. besondere Wünsche mitteilen. Besondere Wünsche können z.B. zusätzliche Angaben zur Versandart (Übertragungsgeschwindigkeit, Anforderung einer Quittung für die Auslieferung, ...) sein, falls das E-Mail-System in diesem Zusammenhang verschiedene Optionen anbietet.
- Der Benutzeragent kann den Nachrichteninhalt zusammen mit dem Umschlag an das Nachrichtenübertragungssystem zur Weiterleitung an den bzw. an die Empfänger übergeben.
- Der Benutzeragent kann den Umschlag und den Inhalt einer Nachricht vom Nachrichtenübertragungssystem entgegennehmen.
- Der Nachrichteninhalt und der Absender können dem Benutzer präsentiert werden.

Transferagent: Transferagenten können sowohl mit weiteren Transferagenten als auch mit mehreren Benutzeragenten verbunden sein. Die Transferagenten und die Verbindungen bilden das Nachrichtenübertragungssystem. Ein Transferagent übernimmt die Nachrichten von seinen jeweiligen Benutzeragenten oder von anderen Transferagenten.

Wenn ein Transferagent A eine Nachricht übernommen hat, wird überprüft, ob die Nachricht an einen anderen Transferagenten weitergeleitet werden muß oder ob der Benutzeragent des Empfängers der Nachricht dem Transferagenten A zugeordnet ist. Ist der Benutzeragent des Empfängers dem Transferagenten A zugeordnet, wird die Nachricht nicht weitergeleitet.

Um Nachrichten zu speichern, kann der Transferagent für jeden Benutzeragenten, der ihm zugrordnet ist, einen *Nachrichtenspeicher* besitzen. Die Nachrichtenspeicher sind in der Empfehlung X.400 optional. Wenn für einen Benutzer kein Nachrichtenspeicher existiert, müssen Nachrichten an diesen Benutzer sofort vom entsprechenden Benutzeragenten im Empfang genommen werden können, damit sie nicht verloren gehen. Für das betrachtete Fallbeispiel wird die Existenz der Nachrichtenspeicher für alle Benutzer des E-Mail-Systems vorausgesetzt. Auf

Anforderung des Benutzeragenten werden Nachrichten aus dem Nachrichtenspeicher an den Benutzeragenten übergeben oder gelöscht [Tan92, S. 667f.].

Allgemein ist jeder Benutzeragent genau einem Transferagenten zugeordnet. Ein Transferagent kann entweder gar keinen, genau einen oder mehrere Benutzeragenten bedienen. Eine übliche Abbildung des Modells auf reale Computersysteme besteht darin, die Transportagenten auf leistungsfähigen Großrechnern zu installieren, die von öffentlichen oder privaten Anbietern zur Verfügung gestellt werden, während die Benutzeragenten auf PCs oder Workstations eingerichtet sind, die sich in den Büros der Benutzer befinden. Dennoch ist diese Art der Abbildung nicht zwingend durch X.400 vorgeschrieben. Es sind beliebige weitere Konfigurationen möglich. Zum Beispiel können verschiedene Benutzeragenten zusammen mit einem Transferagenten auf demselben Computer installiert sein. Eine Übersicht über verschiedene Abbildungen des Modells nach X.400 auf reale Computersysteme geben z.B. Babatz et al. [BBP90, S. 30].

4.3 Beschreibung der Anforderungen an das System und an den Prototyp

Das zu modellierende E-Mail-System ist so aufgebaut, wie es im Abschnitt 4.2 beschrieben wurde. Nachrichten werden von einem Benutzeragenten an das Nachrichtenübertragungssystem übergeben. Von dort aus werden sie zum Empfänger transportiert. Das Nachrichtenübertragungssystem besteht aus Transferagenten, die miteinander verbunden sind.

Die im Abschnitt 4.2 beschriebenen Regeln für die Zuordnung von Benutzer- und Transferagenten werden beachtet. Ein Benutzeragent nimmt also die Leistungen genau eines Transferagenten in Anspruch, während ein Transferagent mehrere, genau einen oder gar keinen Benutzeragenten bedienen kann. An einen Transferagenten werden die folgenden Anforderungen gestellt:

- Die Transferagenten sorgen für die Weiterleitung von Nachrichten zum Empfänger.
- Ein Transferagent richtet für jeden ihm zugeordneten Benutzer einen Nachrichtenspeicher ein. Dort werden Nachrichten für die jeweiligen Benutzer abgelegt. Auf Anforderung eines Benutzeragenten übergibt oder löscht der Transferagent Nachrichten aus einem Nachrichtenspeicher. Außerdem bietet der Transferagent die Möglichkeit, ein Inhaltsverzeichnis eines Nachrichtenspeichers an einen Benutzeragenten zu übergeben.

Die Schnittstelle zwischen Nachrichtenübertragungssystem und den Benutzern wird durch die Benutzeragenten gebildet. Ein Benutzeragent soll die folgenden Anforderungen erfüllen:

- Ein Inhaltsverzeichnis des Nachrichtenspeichers eines Benutzers kann angezeigt werden.
- Der Inhalt von Nachrichten kann gelesen werden.
- Nachrichteninhalte können in einer Datei gespeichert werden.
- Auf Nachrichten kann direkt geantwortet werden. Die Adresse des Empfängers der Antwortnachricht ermittelt der Benutzeragent selbstständig.
- Nachrichten können erzeugt und an das Transportsystem weitergegeben werden.
- Eingetroffene Nachrichten können an andere Benutzer weitergeleitet werden.

Bei den in diesem Fallbeispiel betrachteten Nachrichten handelt es sich immer um Texte. Nebenbedingungen beim Versenden einer Nachricht vom Absender zum Empfänger, wie z.B. das Versenden einer Nachricht über den kürzesten Weg zum Empfänger, werden nicht vorgegeben.

Für den zu implementierenden Prototyp werden jeweils ein ProSet-Programm für einen Benutzeragenten und für einen Transfergenten erstellt. Durch die gleichzeitige Ausführung mehrerer

Instanzen des Benutzer- und des Transferagentenprogramms wird ein Modell eines Netzwerks aufgebaut. Die Anzahl der Benutzeragenten und Transferagenten sowie die bestehenden Verbindungen sind nicht fest vorgegeben. Das zu simulierende Netzwerk kann flexibel konfiguriert werden.

In der Einleitung zum Kapitel 4 wurde bereits erwähnt, daß die grundlegende Idee, ein E-Mail-System als Beispiel für eine Anwendung, bei der persistente Daten benutzt werden, zu betrachten, von Gamerman et al. übernommen wurde [GLV92]. Bei Gamerman et al. werden jedoch ausschließlich Komponenten implementiert, die in etwa die Funktionalität der hier betrachteten Benutzeragenten besitzen. Nachrichten werden von diesen Komponenten in einer Datenbank abgelegt und können aus dieser Datenbank wieder gelesen und gelöscht werden. Das in der Diplomarbeit betrachtete E-Mail-Modell ist wegen der zusätzlichen Modellierung eines aus mehreren Transferagenten bestehenden Nachrichtenübertragungssystems komplexer aufgebaut. Die Modellierung von Transferagenten im Rahmen des experimentellen Prototyping erscheint jedoch sinnvoll, weil die Transferagenten wesentliche Bestandteile des in der Ist-Analyse (siehe Abschnitt 4.2) zugrunde gelegten E-Mail-Systems sind.

4.4 Das Anforderungsmodell

Das Anforderungsmodell besteht analog zum Anforderungsmodell des ersten Fallbeispiels aus dem Anwendungsfallmodell und aus dem Problembereichsmodell (vgl. Abschnitt 3.4.1). Benutzerschnittstellen spielen bei dem zu erstellenden Prototyp nur eine untergeordnete Rolle. Außerdem finden keine Interaktionen mit fremden System statt. Das E-Mail-System besteht zwar aus verschiedenen Prozessen, die miteinander kommunizieren. Diese Prozesse werden jedoch als ein zusammenhängendes System betrachtet. Schnittstellenbeschreibungen werden in dem hier betrachteten Anforderungsmodell deshalb nicht erstellt.

4.4.1 Das Anwendungsfallmodell

Bei der Erstellung des Anwendungsfallmodells wird auf die Begriffe zurückgegriffen, die bereits im Glossar (siehe Zusatzdokument zur Diplomarbeit), das parallel zur Ist-Analyse und zur Anforderungsdefinition erstellt wurde, definiert sind. Bei der Erstellung des Anwendungsfallmodells wird das Glossar um weitere Begriffe ergänzt. Ein Beispiel für einen Begriff, der erst bei der Erstellung des Anwendungsfallmodells im Glossar definiert wird, ist der Begriff *Bezugsnummer*:

> „Die Bezugsnummer kennzeichnet eine Nachricht im Nachrichtenspeicher eines Benutzers des Nachrichtenaustauschsystems eindeutig.“

Das Anwendungsfallmodell des E-Mail-Systems besteht aus zwei Teilen. Im ersten Teilmodell werden die Anwendungsfälle für den Benutzeragenten beschrieben. Die Beschreibung der Anwendungsfälle für die Transferagenten befindet sich im zweiten Teil des Anwendungsfallmodells.

Auf die Konfigurierung des Benutzeragenten und des Transferagenten wird im Anwendungsfallmodell nicht eingegangen. Hierdurch unterscheidet sich das Anwendungsfallmodell des E-Mail-Systems vom Anwendungsfallmodell des Biliotheksverwaltungssystems. Beim Bibliotheksverwaltungssystem wurde die Einrichtung des Systems im Anwendungsfallmodell berücksichtigt (Anwendungsfälle Beschreibung erreichbarer Bücherkataloge und Beschreibung erreichbarer Zeitschriftenkataloge, siehe Abbildung 3.2 im Abschnitt 3.4.1).

Der Einfluß einer Beschreibung von Anwendungsfällen, in denen die Einrichtung des zu entwickelnden Systems dargestellt wird, auf die weitere Systementwicklung wird nach den Erfahrungen, die beim ersten Fallbeispiel gemacht wurden, als so gering eingeschätzt, daß die Betrachtung dieser Anwendungsfälle vernachlässigt werden kann, ohne die weitere Software-Entwicklung dadurch negativ zu beeinflussen.

Im folgenden wird das Anwendungsfall des E-Mail-Systems, getrennt nach Benutzeragent und Transferagent, vorgestellt.

Der Benutzeragent: Das Teilmodell des Benutzeragenten ist in den Abbildungen 4.2 und 4.3 skizziert. In der Abbildung 4.2 sind der Benutzer des E-Mail-Systems als einziger Akteur und die noch nicht erweiterten Anwendungsfälle skizziert. In der Abbildung 4.3 wird die Erweiterung des Anwendungsfalls Nachricht versenden durch den Anwendungsfall Nachricht erzeugen veranschaulicht.

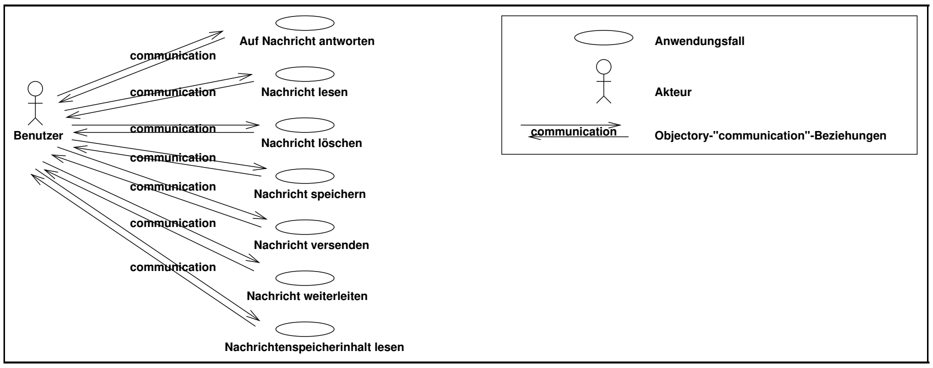

Abbildung 4.2: Akteur und Anwendungsfälle für den Benutzeragenten.

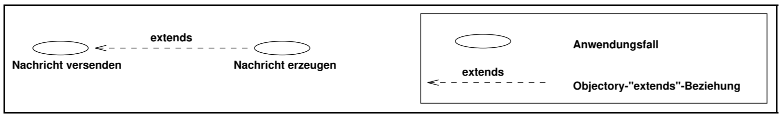

Abbildung 4.3: Erweiterung des Anwendungsfalls Nachricht versenden

Im folgenden ist die Beschreibung des Anwendungsfalls Nachricht versenden aufgeführt.

Der Anwendungsfall Nachricht versenden

Ereignisfluß

Start
Im Menü des Benutzeragenten wird der Befehl *Nachricht versenden* gewählt.

Eingabe der Empfängeradressen
Der Benutzer gibt die Adressen der Empfänger der Nachricht ein. Anschließend wählt er den Befehl *Nachricht erzeugen*. Alternativ kann er die Ausführung durch Auswahl des Befehls *Abbruch* vorzeitig abbrechen.

Eingabe des Betreffs
Der Benutzer gibt den Betreff der Nachricht ein.

Erzeugen und Verschicken der Nachricht
Der Benutzer gibt den Namen der Datei ein, in der der Text des Nachrichtenkörpers gespeichert ist. Alternativ kann der Benutzer den Befehl *Abbruch* wählen, um zum Hauptmenü zurückzukehren, ohne eine Nachricht zu versenden.

Wenn nicht *Abbruch* gewählt wurde, wird die Nachricht durch den Benutzeragenten erzeugt und an die Empfänger verschickt.

Ausnahmen

Keine Verbindung zum Transferagenten
Wenn keine Verbindung zu einem Transferagenten existiert, wird die Fehlermeldung *Transferagent nicht gefunden* ausgegeben.

Datei kann nicht gelesen werden
Wenn die ausgewählte Datei mit dem Text des Nachrichtenkörpers nicht geöffnet werden kann, wird die Fehlermeldung *Datei kann nicht geöffnet werden* ausgegeben.

Nachricht kann nicht zugestellt werden
Wenn die Nachricht an einen angegebenen Empfänger nicht zugestellt werden konnte, wird die Fehlermeldung *Nachricht konnte nicht zugestellt werden* zusammen mit der Adresse, an die die Nachricht nicht ausgeliefert werden konnte, ausgegeben.

Überschreitung einer maximalen Wartezeit
Wenn nach dem Ablauf einer maximalen Wartezeit noch keine Quittung vom Nachrichtenempfänger eingegangen ist, wird die Fehlermeldung *Timeout* ausgegeben.

Beim Anwendungsfall Nachricht versenden wird zunächst davon ausgegangen, daß der Text des Nachrichtenkörpers einer zu versendenden Nachricht bereits in einer Datei gespeichert ist. Durch die „extends“-Beziehung [JCJÖ93, S. 163ff.], die den Anwendungsfall Nachricht erzeugen mit dem Anwendungsfall Nachricht versenden verbindet (siehe Abbildung 4.3), wird bei der Ausführung des Anwendungsfalls Nachricht versenden die Option geboten, den Anwendungsfall Nachricht erzeugen zusätzlich auszuführen. Dadurch hat der Benutzer die Möglichkeit, einen Text für den Nachrichtenkörper direkt einzugeben, anstatt ihn aus einer Datei zu laden.

Sowohl bei dem Anwendungsfall Nachricht versenden als auch bei dem Anwendungsfall Nachricht erzeugen handelt es sich um Anwendungsfälle, die in einem anderen Kontext auch unabhängig voneinander verwendet werden könnten.

Der Anwendungsfall Nachricht erzeugen ist folgendermaßen beschrieben:

Der Anwendungsfall Nachricht erzeugen

Ereignisfluß

Start
Der Benutzer drückt auf die Frage nach dem Namen einer Datei mit einem Text für einen Nachrichtenkörper die Taste *Enter*, ohne einen Dateinamen einzugeben.

Eingabe des Nachrichtenkörpers
Der Benutzer gibt den Text des Nachrichtenkörpers ein. Wenn die Eingabe beendet ist, wählt er den Befehl *Ende*.

Ausnahme

Kein Nachrichtenkörper eingegeben
Wenn der Befehl *Ende* ausgewählt wird, ohne daß ein Text für den Nachrichtenkörper eingegeben wurde, springt das Programm sofort zum Hauptmenü zurück.

Die Aufgaben der Anwendungsfälle, die für den Benutzeragenten spezifiziert wurden, werden in der Tabelle 4.1 kurz beschrieben. Die ausführlichen Beschreibungen sämtlicher Anwendungsfälle wurden aus Platzgründen nicht in dieses Kapitel aufgenommen. Sie befinden sich im Zusatzdokument zu dieser Diplomarbeit.

Anwendungsfall	Beschreibung
Auf Nachricht antworten	Der Benutzer antwortet auf eine eingegangene Nachricht direkt.
Nachricht erzeugen	Der Benutzer erzeugt einen Nachrichtenkörper.
Nachricht lesen	Eine Nachricht wird ausgewählt. Der Inhalt der ausgewählten Nachricht wird angezeigt.
Nachricht löschen	Der Benutzer wählt eine Nachricht, die gelöscht werden soll, aus seinem Nachrichtenspeicher aus.
Nachricht speichern	Der Inhalt einer Nachricht wird in einer Datei gespeichert.
Nachricht versenden	Eine Nachricht wird versendet.
Nachricht weiterleiten	Eine eingegangene Nachricht wird an andere Benutzer weitergeleitet.
Nachrichtenspeicherinhalt lesen	Es wird angezeigt, welche Nachrichten sich im Nachrichtenspeicher des Benutzers befinden.

Tabelle 4.1: Beschreibung der Anwendungsfälle für den Benutzeragenten.

Der Transferagent: Während im Teilmodell des Benutzeragenten nur der Benutzer als einziger Akteur existiert, besitzt das Teilmodell des Transferagenten die beiden Akteure Benutzeragent und Transferagent, da der Transferagent sowohl mit Benutzeragenten als auch mit fremden Transferagenten Informationen austauscht. Die Anwendungsfälle und die Akteure des Transferagenten sind in der Abbildung 4.4 dargestellt.

Exemplarisch soll hier nur der Anwendungsfall Nachricht übergeben ausführlich beschrieben werden. Detaillierte Beschreibungen sämtlicher Anwendungsfälle des Transferagenten befinden sich im Zusatzdokument zur Diplomarbeit. Ein kurze Übersicht über die Anwendungsfälle des Transferagenten gibt die Tabelle 4.2.

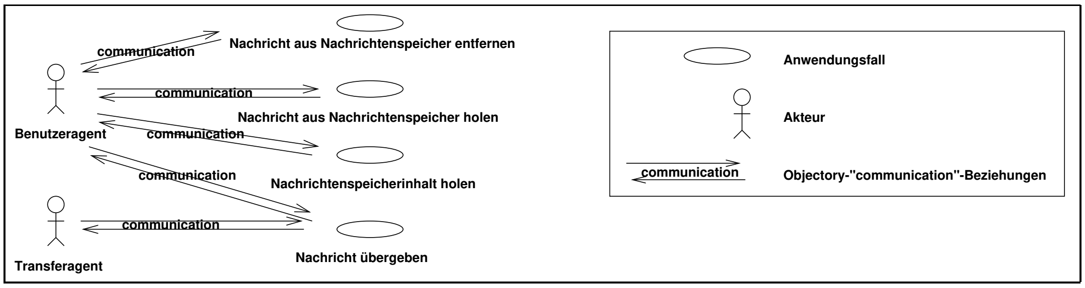

Abbildung 4.4: Akteure und Anwendungsfälle für den Transferagenten.

Der Anwendungsfall Nachricht übergeben

Ereignisfluß

Auftrag
Ein Benutzeragent oder ein anderer Transferagent übergibt eine Nachricht oder eine Quittung an den Transferagenten.

Reaktion
Wenn für den Empfänger der Nachricht bzw. Quittung ein Nachrichtenspeicher verwaltet wird, wird kontrolliert, ob es sich bei den empfangenen Informationen um eine Quittung für eine andere Nachricht oder um eine Nachricht zum Informationsaustausch zwischen Benutzern handelt. Handelt es sich um eine Quittung, wird die Quittung an den Benutzeragenten des Quittungsempfängers weitergegeben. Wenn es sich um eine Nachricht zum Informationsaustausch zwischen Benutzern handelt, wird die empfangene Nachricht im Nachrichtenspeicher des Benutzers abgelegt. Zusätzlich wird eine positive Quittung erzeugt, die an den Absender der im Nachrichtenspeicher abgelegten Nachricht zurückgeschickt wird.

Wenn für den Empfänger der Nachricht bzw. Quittung kein Nachrichtenspeicher verwaltet wird, wird ein Transferagent gesucht, an den die Nachricht bzw. Quittung weitergeleitet werden kann. Anschließend wird die Nachricht bzw. Quittung an den gefundenen Transferagenten weitergeleitet.

Bei dem Versenden einer Nachricht vom Absender zum Empfänger sind keine Nebenbedingungen, wie z.B. ein Versenden der Nachricht über den kürzesten Weg, vorgegeben.

Ausnahmen

Falsches Format
Wenn das Format der Nachricht bzw. Quittung nicht dem erwarteten Format entspricht, wird die Fehlermeldung *Falsches Format* an den Benutzeragenten bzw. Transferagenten, von dem die Nachricht bzw. Quittung empfangen wurde, zurückgeschickt. Anschließend wird die Ausführung dieses Anwendungsfalls abgebrochen.

Weiterleiten nicht möglich
Wenn die empfangenen Informationen nicht an den angegebenen Empfänger weitergeleitet werden können, wird überprüft, ob es sich bei diesen Informationen um eine Quittung oder um eine Nachricht zum Informationsaustausch zwischen Benutzern handelt. Handelt es sich um eine Quittung, wird die Ausführung dieses Anwendungsfalls abgebrochen. Handelt es sich um eine Nachricht zum Informationsaustausch zwischen Benutzern, wird eine negative Quittung erzeugt und an den Absender der Nachricht, die nicht weitergeschickt werden kann, zurückgesendet

Anwendungsfall	Beschreibung
Nachrichtenspeicherinhalt holen	Informationen über den Inhalt des Nachrichtenspeichers eines Benutzers werden an den Benutzeragenten übergeben.
Nachricht aus Nachrichtenspeicher entfernen	Eine Nachricht wird aus dem Nachrichtenspeicher eines Benutzers entfernt.
Nachricht aus Nachrichtenspeicher holen	Der Transferagent übergibt eine Nachricht an den Benutzeragenten.
Nachricht übergeben	Ein Benutzeragent oder ein fremder Transferagent übergibt eine Nachricht oder eine Quittung an einen Transferagenten, um sie an einen Empfänger weitersenden zu lassen.

Tabelle 4.2: Beschreibung der Anwendungsfälle für den Transferagenten.

4.4.2 Das Problembereichsmodell

Im Problembereichsmodell werden Materialien und Werkzeuge aus dem Problembereich des E-Mail-Systems und ihre Beziehungen zueinander beschrieben. Weil die Problembereiche des Benutzeragenten und des Transferagenten identisch sind, wird das Problembereichsmodell im Gegensatz zum Anwendungsfallmodell (siehe Abschnitt 4.4.1) nicht in ein Teilmodell für den Benutzeragenten und in ein zweites Teilmodell für den Transferagenten aufgeteilt. Es existiert nur ein Problembereichsmodell, das gleichermaßen für den Benutzeragenten und für den Transferagenten gültig ist.

Während im Problembereichsmodell des Systems zur Bibliotheksverwaltung die Beziehungen zwischen Objekten für jeden einzelnen Anwendungsfall separat dargestellt wurden (siehe Abschnitt 3.4.1), wird bei der Darstellung der Beziehungen zwischen den Problembereichsmodellobjekten des E-Mail-Systems nicht mehr zwischen verschiedenen Anwendungsfällen unterschieden. Die Gründe für die Erstellung unterschiedlich ausführlicher Problembereichsmodelle in den beiden Fallbeispielen werden im folgenden vorgestellt.

- Beim Bibliotheksverwaltungssystem wurde der Prototyp in einer sehr frühen Phase des Software-Entwicklungsprozesses erstellt. Von den Objectory-Modellen wurden nur das Anwendungsfallmodell und das Problembereichsmodell vollständig entwickelt. Das Analysemodell wurde nur teilweise erstellt. Durch ein detailliertes Problembereichsmodell sollte, soweit möglich, ein Ausgleich für nicht erstellte Objectory-Modelle bzw. Modellteile geschaffen werden.

  Der Prototyp des E-Mail-Systems wird hingegen in einer relativ späten Phase des Software-Entwicklungsprozesses implementiert. Das Entwurfsmodell existiert bei der Implementierung des Prototyps bereits teilweise. Das Anwendungsfallmodell, das Problembereichsmodell und das Analysemodell existieren vollständig. Ein Ausgleich für fehlende Modellteile wird bei der Erstellung des Prototyps für das E-Mail-System deshalb als nicht nötig angesehen.

- Bei dem System zur Bibliotheksverwaltung handelt es sich um eine Anwendung, bei der viele unterschiedliche Daten, zwischen denen vielfältige Beziehungen bestehen, anfallen. Diese Beziehungen müssen bei einer Manipulation von Daten beachtet werden, um die Konsistenz des Datenbestandes zu erhalten. Die separate Darstellung von Beziehungen zwischen Objekten für jeden Anwendungsfall dient dazu, die Anzahl der Beziehungen, die gleichzeitig beachtet werden müssen, zu begrenzen und so die Übersichtlichkeit des Problembereichsmodells zu verbessern.

  Beim E-Mail-System werden zum einen wesentlich weniger Daten benötigt, zum anderen bestehen zwischen den Daten wesentlich weniger Beziehungen als beim System zur Bibliotheksverwaltung (auf die Beziehungen zwischen Objekten des Problembereichsmodells des E-Mail-Systems wird weiter unten in diesem Abschnitt noch genauer eingegangen). Die Beziehungen zwischen den Daten lassen sich bei dem in diesem Kapitel betrachteten Fallbeispiel auch dann noch übersichtlich darstellen, wenn nicht zwischen verschiedenen Anwendungsfällen unterschieden wird.

Über einen fachlichen Klassenentwurf (siehe Abschnitt 2.2) geht auch das bei diesem Fallbeispiel erstellte Problembereichsmodell hinaus. So werden neben einer Darstellung der Beziehungen zwischen Klassen auch statische Beziehungen zwischen Instanzen aufgeführt. Beziehungen zwischen Instanzen werden beim fachlichen Klassenentwurf nicht berücksichtigt.

Bei den dargestellten statischen Beziehungen zwischen Instanzen handelt es sich um Objectory-„acquaintance"-Beziehungen und Objectory-„consists of"-Beziehungen. Durch die Darstellung dieser Beziehungen sollen Zusammenhänge zwischen den modellierten Objekten verdeutlicht werden.

Des weiteren unterscheidet sich das Problembereichsmodell des E-Mail-Systems durch die Aufnahme von Werkzeugen vom Problembereichsmodell des Systems zur Bibliotheksverwaltung. Das Problembereichsmodell des Systems zur Bibliotheksverwaltung enthält keine Werkzeuge, weil sie auch in der Ist-Analyse, die aus Zeitgründen nur oberflächlich durchgeführt werden konnte, nicht aufgeführt sind. Die Ist-Analyse des E-Mail-Systems konnte aus Zeitgründen zwar ebenfalls nur oberflächlich durchgeführt werden. Hier werden jedoch z.B. Benutzeragenten beschrieben, die im folgenden als Werkzeuge angesehen werden. Eine Begründung dafür, daß Werkzeuge in der Ist-Analyse des E-Mail-Systems aufgeführt sind, während sie in der Ist-Analyse des Systems zur Bibliotheksverwaltung nicht enthalten sind, wird im folgenden aufgeführt:

- In der Ist-Analyse des E-Mail-Systems werden viele Grundlagen von der CCITT-Empfehlung X.400 übernommen. Die Empfehlung X.400 beschreibt den Aufbau eines E-Mail-Systems aus verschiedenen Komponenten bereits sehr detailliert. Dabei sind auch bereits Werkzeuge, wie z.B. Benutzeragenten, als wesentliche Bestandteile eines E-Mail-Systems beschrieben. Werkzeuge, die in der Empfehlung X.400 aufgeführt sind, können direkt in das zu entwickelnde System übernommen werden. Sie werden deshalb im Problembereichsmodell berücksichtigt.

  Im Gegensatz dazu konnte bei der Ist-Analyse im ersten Fallbeispiel nicht auf eine ähnlich detaillierte Empfehlung für den Aufbau eines Systems zur Bibliotheksverwaltung zurückgegriffen werden, in der auch bereits Werkzeuge aufgeführt sind, die für das zu entwickelndes Software-System hätten übernommen werden können.

Im folgenden wird auf die Materialien und auf die Werkzeuge des E-Mail-Systems eingegangen. Im Gegensatz zum Abschnitt über das Problembereichsmodell des Systems zur Bibliotheksverwaltung (Abschnitt 3.4.1) beziehen sich die Skizzen, die zum Problembereichsmodell des E-Mail-Systems aufgeführt werden, immer auf das gesamte System und nicht nur auf einzelne Anwendungsfälle, da im Problembereichsmodell des E-Mail-Systems die einzelnen Anwendungsfälle nicht mehr isoliert voneinander betrachtet werden (Begründung: siehe oben).

Materialien: Eine Übersicht über die Materialobjekte und ihre Beziehungen gibt die Abbildung 4.5.

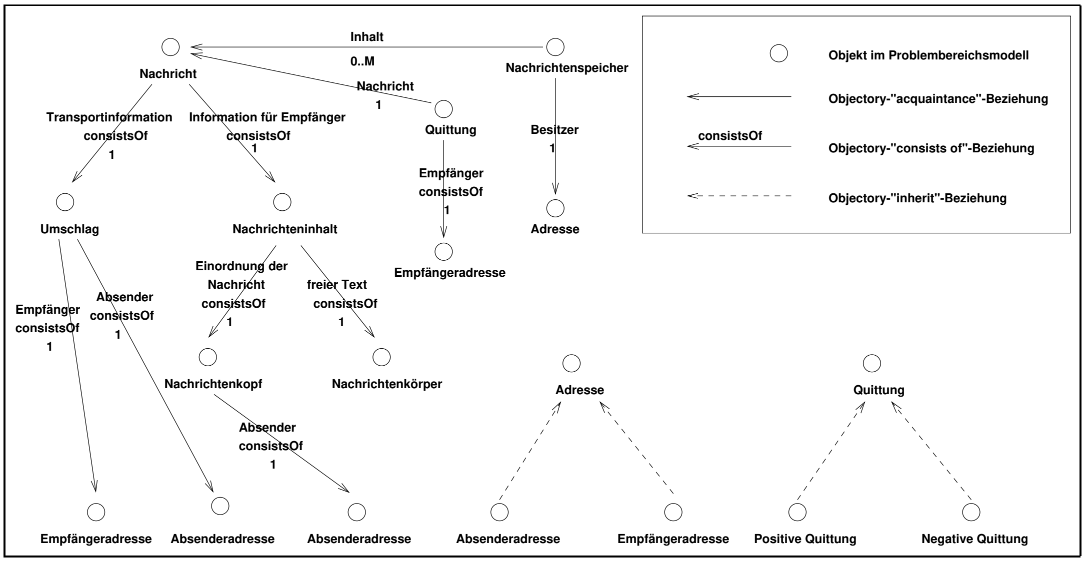

Abbildung 4.5: Materialien im Problembereichsmodell.

Gleiche Bezeichnungen für Beziehungen und Objekte werden bei Jacobson et al. ausdrücklich zugelassen (vgl. Abbildung 3.7 im Abschnitt 3.4.2).

Ein zentrales Objekt ist hier das Objekt Nachrichtenspeicher. Ein Nachrichtenspeicher wird als Material angesehen, das von einem Transferagenten bearbeitet wird. In einem Nachrichtenspeicher können beliebig viele Nachrichten für einen Benutzer gespeichert sein. Dies wird in der Abbildung 4.5 durch die 0..M-„acquaintance"-Beziehung Inhalt dargestellt. Anstelle einer „acquaintance"-Beziehung zwischen dem Objekt Nachrichtenspeicher und dem Objekt

Nachricht hätte auch eine „consists of“-Beziehung modelliert werden können. Die „acquaintance“-Beziehung wurde gewählt, weil die Nachrichten als eigenständige Objekte angesehen werden, die zwar in einem Nachrichtenspeicher gespeichert werden können, aber kein direkter Bestandteil eines Nachrichtenspeichers sind, wie es z.B. Speicherbereiche im Hauptspeicher oder Festplattensektoren wären. Speicherbereiche im Hauptspeicher und Festplattensektoren sind nicht im Problembereichsmodell aufgeführt, weil sie keinen Einfluß auf die weitere Entwicklung des E-Mail-Systems haben.

Jeder Nachrichtenspeicher ist einem Benutzer des E-Mail-Systems fest zugeordnet. Dies wird durch die Beziehung Besitzer zwischen dem Objekt Nachrichtenspeicher und dem Objekt Adresse dargestellt (der Begriff Material umfaßt nicht ausschließlich materielle Dinge, eine Adresse wird daher bei diesem Fallbeispiel als Material angesehen, siehe Abschnitt 2.2).

Die „consists of“-Beziehungen Transportinformation und Information für den Empfänger beschreiben, daß eine Nachricht aus einem Umschlag (Objekt Umschlag) und aus einem Nachrichteninhalt (Objekt Nachrichteninhalt) besteht. Der Umschlag enthält Informationen, die das Nachrichtenübertragungssystem benötigt, um eine Nachricht vom Absender zum Empfänger zu schicken. Deshalb ist die Empfängeradresse ein Bestandteil des Umschlags. Damit eine Quittung an den Absender einer Nachricht zurückgeschickt werden kann, enthält der Umschlag auch die Adresse des Absenders.

Der Nachrichteninhalt spaltet sich in den Nachrichtenkopf und in den Nachrichtenkörper auf. Der Nachrichtenkopf enthält ebenfalls eine Absenderadresse, um dem Nachrichtenempfänger das Einordnen einer Nachricht zu erleichtern. Die Absenderangaben des Umschlags und des Nachrichtenkopfes müssen nicht in jedem Fall identisch sein. Zum Beispiel hat ein Benutzer die Möglichkeit, eine an ihn adressierte Nachricht zusätzlich an einen oder an mehrere andere Benutzer weiterzuleiten (siehe Tabelle 4.1 im Abschnitt 4.4.1). Beim Weiterleiten bleibt der Nachrichteninhalt einschließlich der Absenderangabe im Nachrichtenkopf unverändert. Im Gegensatz dazu wird die Absenderadresse auf dem Nachrichtenumschlag vor dem Weiterleiten der Nachricht ausgetauscht. Der Nachrichtenumschlag einer weitergeleiteten Nachricht enthält als Absender die Adresse des Benutzers, der die Nachricht weiterleitet. Der Grund, im Nachrichtenkopf und auf dem Umschlag unterschiedliche Absenderadressen zuzulassen, soll im folgenden anhand eines Beispiels veranschaulicht werden.

Beispiel: In diesem Beispiel erzeugt ein Benutzer A des E-Mail-Systems eine Nachricht und verschickt sie an einen Benutzer B. Der Benutzer B möchte die Nachricht an einen anderen Benutzer C weiterleiten, gibt jedoch eine falsche Adresse ein. Die eingegebene Adresse existiert im E-Mail-System nicht.

Das Nachrichtenübertragungssystem reagiert auf die fehlerhafte Empfängeradresse mit dem Verschicken einer negativen Quittung an den Absender, dessen Adresse auf dem Umschlag der Nachricht angegeben ist (siehe Beschreibung des Anwendungsfalls Nachricht übergeben im Abschnitt 4.4.1). Wäre auf dem Umschlag noch Benutzer A als Absender angegeben, erhielte der Benutzer A eine negative Quittung, obwohl die von A versendete Nachricht beim gewünschten Empfänger B angekommen ist. Empfänger B erhielte hingegegen keine Rückmeldung, daß die Nachricht, die er an Benutzer C weiterleiten wollte, ihr Ziel nicht erreicht hat.

Die Adresse des Benutzers A im Nachrichtenkopf gibt an, daß Benutzer A (und nicht Benutzer B) der Verfasser des Textes des Nachrichtenkörpers ist. Da der Nachrichtenkörper beim Weiterleiten der Nachricht nicht verändert wird, bleibt die Adresse im Nachrichtenkopf erhalten.

Bei dem hier vorgestellten Problembereichsmodell wird zwischen einer positiven und einer negativen Quittung unterschieden (Objekte Positive Quittung und Negative Quittung). Eine positive Quittung zeigt an, daß eine Nachricht beim Empfänger angekommen ist. Eine negative Quittung wird verschickt, wenn eine Nachricht nicht an den gewünschten Empfänger ausgeliefert werden konnte. In der Abbildung 4.5 soll durch die 1-Beziehung Nachricht zwischen dem Objekt Quittung und dem Objekt Nachricht angedeutet werden, daß sich eine Quittung immer auf eine Nachricht bezieht. Jede Quittung enthält die Adresse des Benutzers, für den die Quittung bestimmt ist. Dies wird durch die Beziehung Empfänger zwischen den Objekten Quittung und Empfängeradresse symbolisiert.

Die hier vorgestellte Modelllierung von Materialien ist nur eine von vielen Möglichkeiten. Zusätzlich zu den oben beschriebenen Objekten könnten weitere Materialobjekte aufgeführt werden. Zum Beispiel hätte ein Materialobjekt Betreff für den Betreff einer Nachricht in das Problembereichsmodell aufgenommen und mit dem Objekt Nachrichtenkopf durch eine „consists of"-Beziehung verbunden werden können, wie es in der Abbildung 4.6 dargestellt ist.

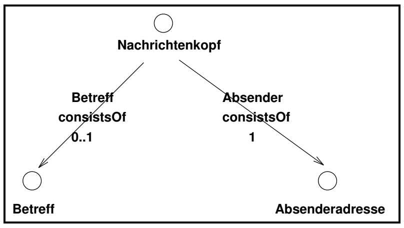

Abbildung 4.6: Modellierung eines zusätzlichen Objektes im Problembereichsmodell.

Die Beschränkung der im Problembereichsmodell modellierten Materialobjekte auf die in der Abbildung 4.5 dargestellten Objekte soll im folgenden beschrieben und motiviert werden.

Ein Material A, das Bestandteil eines anderen Materials B ist, wird nur dann in das Problembereichsmodell aufgenommen, wenn das Material A auch unabhängig vom Material B benötigt wird. Dies ist bei dem Betreff einer Nachricht nicht der Fall. Der Betreff einer Nachricht wird bei dem betrachteten E-Mail-System immer im Zusammenhang mit den restlichen Informationen des Nachrichtenkopfes, aber niemals allein, benutzt. Im Gegensatz dazu wird z.B. eine Adresse nicht ausschließlich als Bestandteil einer Nachricht benötigt. Adressen werden beispielsweise als eigenständige Informationen benötigt, wenn ein Benutzeragent gestartet wird. Der Benutzeragent muß die Adresse des Benutzers, der ihn gestartet hat, erfragen, um dem Transferagenten mitteilen zu können, auf welchen Nachrichtenspeicher sich die vergebenen Aufträge (z.B. das Anfordern von Informationen über den Inhalt eines Nachrichtenspeichers) beziehen sollen. Informationen, die nicht unabhängig sind, werden erst im Analysemodell als Attribute modelliert. Das Analysemodell wird im Abschnitt 4.5 beschrieben.

Durch diese Regelung für die Aufnahme von Materialobjekten soll verhindert werden, daß das Problembereichsmodell mit Objekten überladen wird, die keine Informationen liefern, die nicht bereits in anderen Objekten enthalten sind. So soll die Übersichtlichkeit des Problembereichsmodells erhalten bleiben. Eine ähnliche Beschränkung für die Aufnahme von Objekten in das Problembereichsmodell wurde bereits bei der Modellierung des Systems zur Bibliotheksverwaltung vorgenommen. Die Gründe für die Beschränkung der Aufnahme von Objekten in das Problembereichsmodell des Bibliotheksverwaltungssystems sind mit den Gründen, die für das E-Mail-System angegeben wurden, vergleichbar. Sie wurden bei der Beschreibung der Vorgehensweise zur Modellierung des Bibliotheksverwaltungssystems vorgestellt (siehe Abschnitt 3.1).

Es läßt sich nicht in allen Fällen eindeutig entscheiden, ob Informationen eines Objektes unabhängig von den Informationen eines anderen Objektes benötigt werden. Diese Problematik wird bei den Objekten Umschlag, Nachrichteninhalt, Nachrichtenkopf und Nachrichtenkörper deutlich. Auf diese Objekte wird im folgenden genauer eingegangen:

Die Objekte Umschlag und Nachrichteninhalt: Ein Benutzeragent übergibt nur vollständige Nachrichten an einen Transferagenten, und ein Transferagent leitet nur vollständige Nachrichten an einen anderen Transferagenten weiter. Dies spräche dafür, die Objekte Umschlag und Nachrichteninhalt nicht als Materialobjekte in das Problembereichsmodell aufzunehmen.

Andererseits liest das Nachrichtenübertragungssystem beim Weiterleiten einer Nachricht ausschließlich die Informationen des Umschlags, während die Informationen des

Nachrichteninhalts durch das Nachrichtenübertragungssystem nicht interpretiert werden. Der Benutzer des Nachrichtenaustauschsystems benötigt hingegen die Informationen des Nachrichteninhalts, während die Informationen des Umschlags für ihn unwichtig sind. Die Trennung von Umschlag und Nachrichteninhalt soll auch im Problembereichsmodell dargestellt werden. Deshalb werden die Materialobjekte Umschlag und Nachrichteninhalt aufgenommen.

Die Objekte Nachrichtenkopf und Nachrichtenkörper: Die Entscheidung, ob für den Nachrichtenkopf und für den Nachrichtenkörper eigene Materialobjekte erzeugt werden sollen, ist noch schwieriger als bei den Objekten Umschlag und Nachrichteninhalt. Die Objekte Nachrichtenkopf und Nachrichtenkörper wurden aufgenommen, weil die Aufteilung des Nachrichteninhalts in den Nachrichtenkopf und in den Nachrichtenkörper in der CCITT-Empfehlung X.400 stark betont wird (siehe Abschnitt 4.2). Der Nachrichtenkopf enthält Informationen, die durch den Benutzeragenten zum Teil selbständig erzeugt werden können (z.B. die Adresse des Absenders), während der Nachrichtenkörper ausschließlich durch den Benutzer erzeugt wird. Ob die Aufnahme des Nachrichtenkopfes und des Nachrichtenkörpers als Materialobjekte im Problembereichsmodell für die Implementierung notwendig ist, läßt sich in dieser Phase des Software-Entwicklungsprozesses noch nicht eindeutig entscheiden.

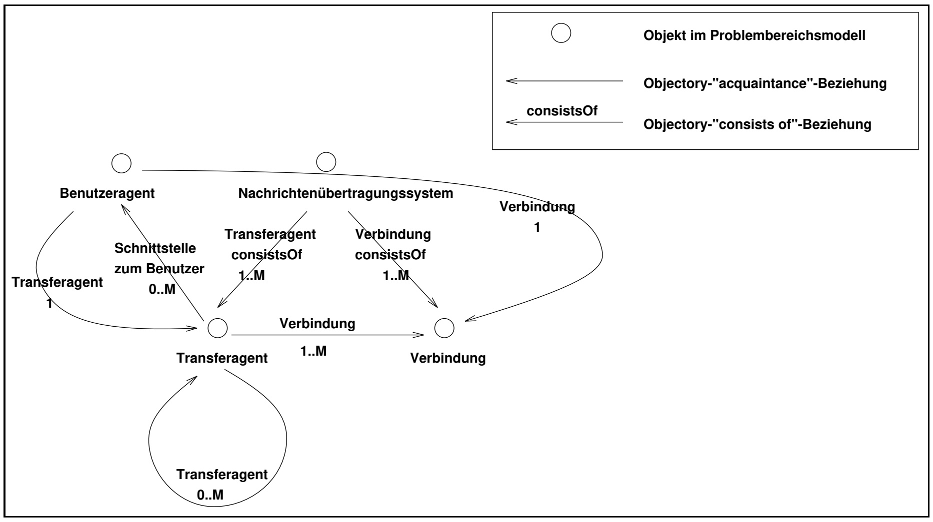

Abbildung 4.7: Werkzeuge im betrachteten Problembereich.

Werkzeuge: Zur Vorstellung der im Problembereichsmodell dargestellten Werkzeuge soll zunächst die Abbildung 4.7 erläutert werden. Bei der Erläuterung werden die in dieser Abbildung dargestellten Objekte zunächst nur als Problembereichsmodellobjekte, aber noch nicht speziell als Repräsentation von Werkzeugen, angesehen. Der Werkzeugbegriff im Zusammenhang mit den dargestellten Objekten wird im Anschluß an die Erläuterung der Abbildung diskutiert.

Erläuterung der Abbildung: Das System zum Nachrichtenaustausch besteht aus (mehreren) Benutzeragenten und dem Nachrichtenübertragungssystem. Das Nachrichtenübertragungssystem setzt sich aus Transferagenten und aus Verbindungen zusammen. Das Objekt Verbindung repräsentiert die physikalischen Verbindungen (z.B. elektrische Leitungen) zwischen einem Benutzeragenten und einem Transferagenten bzw. zwischen zwei Transferagenten. Diese physikalischen Verbindungen zwischen zwei Orten werden nicht durch die eingezeichneten Kanten symbolisiert. Die Kanten zeigen ausschließlich an, daß ein Objekt Kenntnis davon besitzt, daß ein anderes Objekt existiert.

So muß beispielsweise ein Benutzeragent, der eine Nachricht an einen Transferagenten übergeben soll, wissen, daß dieser Transferagent existiert. Dies wird durch die „acquaintance"-Beziehung zwischen dem Benutzeragenten und dem Transferagenten ausgedrückt. Damit die Nachricht versendet werden kann, muß noch eine physikalische Verbindung existieren. Diese Verbindung wird durch das Objekt Verbindung repräsentiert.

Die Objekte Nachrichtenübertragungssystem und Verbindung wurden zusätzlich zu den Objekten Benutzeragent und Transferagent in diese Abbildung aufgenommen, weil sowohl das Nachrichtenübertragungssystem als auch Verbindungen gemäß Abbildung 4.1 (siehe Abschnitt 4.2) Bestandteile des Problembereichs eines E-Mail-Systems sind.

Diskussion des Werkzeugbegriffs: Sämtliche in der Abbildung 4.7 aufgeführten Objekte werden im folgenden als *Werkzeugobjekte* angesehen. Der Begriff Werkzeugobjekt umfaßt dabei sowohl Objekte, die einzelne Werkzeuge darstellen, als auch ein Objekt, das eine Sammlung von Werkzeugen repräsentiert. Das Objekt Nachrichtenübertragungssystem repräsentiert eine Sammlung mehrerer Werkzeuge. Die Objekte Benutzeragent, Transferagent und Verbindung stellen einzelne Werkzeuge dar.

Während ein Benutzeragent offensichtlich ein Werkzeug zum Bearbeiten von Nachrichten ist, ist die Benutzung der Metapher Werkzeug für die Transferagenten und für die Verbindungen jedoch problematisch. Dies wird im folgenden diskutiert:

- Mit einem Transferagenten werden Nachrichtenspeicher bearbeitet und Nachrichten zum Weitersenden vorbereitet. Eine Verbindung dient dazu, eine Nachricht von einem Ort zu einem anderen Ort zu transportieren. Weil bei dieser Interpretation sowohl Transferagenten als auch Verbindungen dazu dienen, Nachrichtenspeicher bzw. Nachrichten zu bearbeiten, sind Transferagenten und Verbindungen keine Materialien.

  Andererseits sind Transferagenten und Verbindungen strenggenommen auch keine Werkzeuge, weil Aktionen der Transferagenten und der Verbindungen nicht direkt von Menschen veranlaßt werden. Bäumer et al. beschreiben, daß Werkzeuge „ohne menschliche Benutzung ‚herumliegen' " [BGZ95, S. 48], d.h. keine Materialien bearbeiten können.

  Neben Werkzeugen und Materialen erwähnen Bäumer et al. noch Automaten, mit denen Prozesse, die immer nach einem gleichen Schema ablaufen, abgearbeitet werden (siehe Abschnitt 2.2). Automaten können nach Bäumer et al. menschliche Handlungsweisen ersetzen. Die Beförderung einer Nachricht mit Hilfe von Verbindungen und Transferagenten könnte in diesem Zusammenhang als Ersatz für die manuelle Beförderung einer Nachricht von einem Ort zu einem anderen Ort angesehen werden. Andererseits handelte es sich nicht mehr um ein E-Mail-System, wenn Nachrichten nicht elektronisch verschickt werden könnten, sondern manuell überbracht werden müßten.

  Da bereits der Ist-Analyse ein E-Mail-System, und kein traditionelles Postsystem, zugrunde liegt, können die Verbindungen und die Transferagenten bei dem betrachteten Fallbeispiel jedoch keine menschlichen Handlungsweisen ersetzen. Sie sind deshalb auch keine Automaten.

  Weitere Metaphern stehen bei dem hier betrachteten WAM-Ansatz nicht zur Verfügung. Um Transferagenten und Verbindungen trotzdem mit den Metaphern, die durch den betrachteten WAM-Ansatz zur Verfügung gestellt werden, beschreiben zu können, werden sie den Werkzeugen zugeordnet. Ein Transferagent wird als Werkzeug angesehen, mit dem Nachrichtenspeicher bearbeitet und Nachrichten zum Weitersenden vorbereitet werden. Eine Verbindung wird als Werkzeug angesehen, mit dem Nachrichten zwischen zwei Orten transportiert werden können.

Die beschriebene Aufteilung der Objekte aus der Abbildung 4.7 in die Werkzeugsammlung Nachrichtenübertragungssystem und in die Werkzeuge Benutzeragent, Transferagent und Verbindung ist nicht die einzige Möglichkeit zur Strukturierung der gefundenen

Objekte. Alternativ könnte z.B. auch das Nachrichtenübertragungssystem als Werkzeug zum Versenden von Nachrichten betrachtet werden, das aus Transferagenten und Verbindungen besteht. Diese Aufteilung hängt weitgehend von subjektiven Kriterien ab. Bei dem hier betrachteten Beispiel erscheinen die Aufgaben, die durch Benutzeragenten, Transferagenten und Verbindungen bewältigt werden, jeweils ausreichend unterschiedlich zu sein, um Benutzeragenten, Transferagenten und Verbindungen als eigenständige Werkzeuge anzusehen.

Die Objekte aus der Abbildung 4.7 und ihre Beziehungen werden im folgenden noch einmal detailliert vorgestellt:

Das Objekt Benutzeragent

Der Benutzeragent bildet die Schnittstelle zwischen den Benutzern des E-Mail-Systems und dem Nachrichtenübertragungssystem.

1-Beziehung Transferagent: Ein Benutzeragent ist genau einem Transferagenten zugeordnet.

1-Beziehung Verbindung: Der Benutzeragent kann über eine Verbindung Daten mit seinem Transferagenten austauschen.

Das Objekt Nachrichtenübertragunsgssystem

Das Nachrichtenübertragungssystem sorgt für die Weiterleitung einer Nachricht vom Absender zum Empfänger.

1..M-Beziehung Transferagent: Transferagenten sind Bestandteile des Nachrichtenübertragungssystems.

1..M-Beziehung Verbindung: Verbindungen sind Bestandteile des Nachrichtenübertragungssystems. Es muß mindestens eine Verbindung zwischen einem Benutzeragenten und einem Transferagenten bestehen.

Das Objekt Transferagent

Die Transferagenten bilden zusammen mit den Verbindungen das Nachrichtenübertragungssystem. Sie übernehmen Nachrichten von Benutzeragenten oder von anderen Transferagenten und leiten sie zum Empfänger weiter.

0..M-Beziehung Schnittstelle zum Benutzer: Der Transferagent kann mit mehreren Benutzeragenten verbunden sein.

0..M-Beziehung Transferagent: Ein Transferagent kann mit anderen Transferagenten verbunden sein, an die Nachrichten weitergeleitet werden können.

1..M-Beziehung Verbindung Über Verbindungen verschickt der Transferagent Nachrichten.

Das Objekt Verbindung

Eine Verbindung transportiert Daten zwischen zwei Transferagenten oder zwischen einem Benutzeragenten und einem Transferagenten.

Zusätzlich zu den bisher beschriebenen Beziehungen existiert noch die in der Abbildung 4.8 dargestellte „acquaintance“-Beziehung zwischen dem Werkzeugobjekt Transferagent und dem Materialobjekt Nachrichtenspeicher.

Analog zum ersten Fallbeispiel stellen die Objekte des Problembereichsmodells des E-Mail-Systems eine Teilmenge der im Glossar definierten Begriffe dar (vgl. Abschnitt 3.1). So ist z.B. die bei der Ist-Analyse verwendete Bezeichnung X.400 (siehe Abschnitt 4.2) im Glossar erklärt. Sie wird jedoch nicht als Objekt in das Problembereichsmodell aufgenommen.

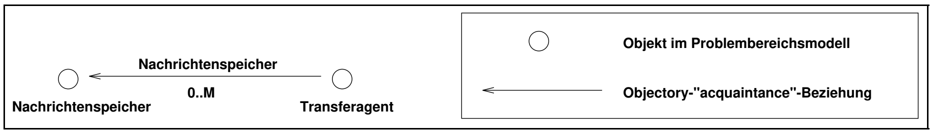

Abbildung 4.8: Beziehung zwischen den Objekten Transferagent und Nachrichtenspeicher.

4.5 Das Analysemodel

Im Gegensatz zum ersten Fallbeispiel (siehe Kapitel 3) wird bei der Entwicklung des Prototyps für das E-Mail-System ein vollständiges Analysemodell mit Entitätsobjekten (siehe Abschnitt 3.4.2), Schnittstellenobjekten [JCJÖ93, S. 176] und Kontrollobjekten [JCJÖ93, S. 190] erstellt.

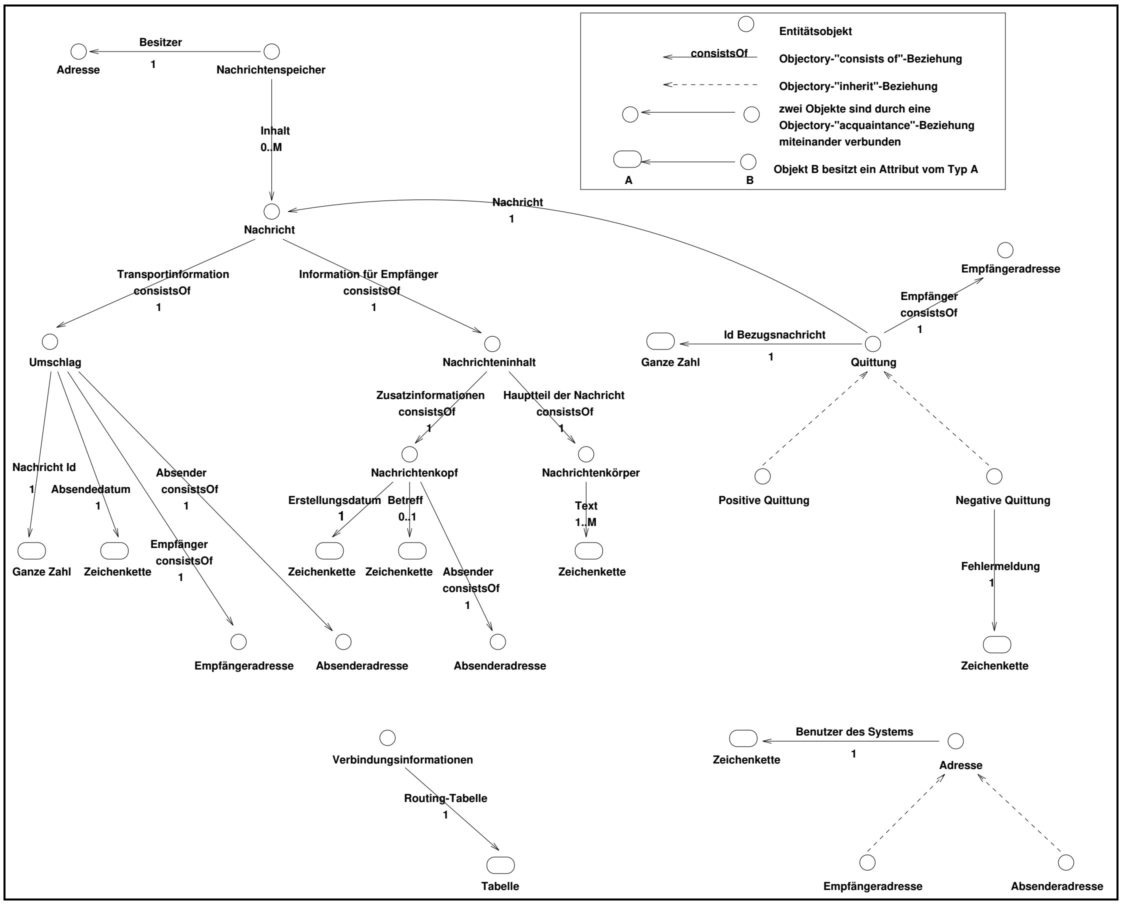

Abbildung 4.9: Materialien im Analysemodell.

Die Entitätsobjekte sind in der Abbildung 4.9 dargestellt. Sie entsprechen weitgehend den im Problembereichsmodell dargestellten Materialobjekten (siehe Abbildung 4.5 im Abschnitt 4.4.2). Zusätzlich zu den Materialobjekten des Problembereichsmodells wird das Objekt Verbindungsinformationen aufgenommen. Dieses Objekt beinhaltet Informationen, die ein Transferagent benötigt, um Nachrichten oder Quittungen weiterzuleiten. Im Gegensatz zum Problembereichsmodell werden im Analysemodell auch Attribute von Objekten aufgeführt. Die Darstellung von Attributen und die Modellierung des Objektes Verbindungsinformationen wird im folgenden erläutert:

Darstellung von Attributen: Attribute werden von Objekten benutzt, um Informationen zu speichern (siehe Abschnitt 3.1).

Bei dem hier vorgestellten Analysemodell wäre es in den meisten Fällen prinzipiell möglich gewesen, anstelle von Attributen Entitätsobjekte zu modellieren, die selbst eigene Attribute haben.

Beispiel: Das Objekt Umschlag besitzt ein Attribut vom Typ Ganze Zahl (siehe Abbildung 4.9). Dieses Attribut dient dazu, eine Zahl zu speichern, durch die eine Nachricht eindeutig gekennzeichnet ist. Anstelle eines Attributes hätte für diese *Nachrichtenidentifikationsnummer* auch ein Entitätsobjekt erzeugt werden können. Dies gilt analog für das Absendedatum, das in einem Atttribut vom Typ Zeichenkette gespeichert ist. Die hier beschriebene alternative Modellierung der Nachrichtenidentifikationsnummer und des Absendedatums ist in der Abbildung 4.10 skizziert.

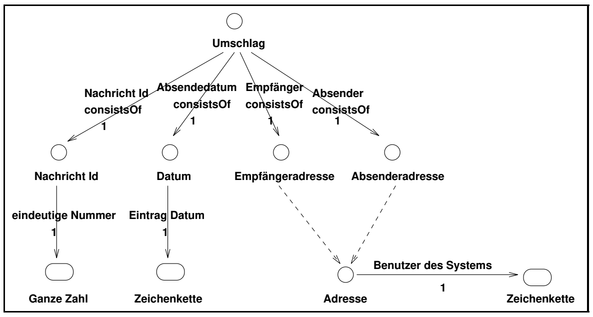

Abbildung 4.10: Alternative Realisierung.

Bei der Entscheidung, für welche Informationen Objekte im Analysemodell erzeugt werden und welche Informationen in Attributen zu Objekten gespeichert werden, geben die im Problembereichsmodell enthaltenen Materialobjekte eine Hilfe. Bei diesem Fallbeispiel werden alle Materialobjekte des Problembereichsmodells im Analysemodell als Entitätsobjekte modelliert. Zusätzliche Informationen werden in Attributen gespeichert. Die einzige Ausnahme hiervon stellt das Entitätsobjekt Verbindungsinformationen, für das kein entsprechendes Materialobjekt im Problembereichsmodell existiert, dar. Dieses Objekt wird im folgenden beschrieben.

Modellierung des Objektes Verbindungsinformationen: Das Objekt Verbindungsinformationen ist Bestandteil eines Transferagenten. In diesem Objekt werden Informationen, die zum Weiterleiten einer Nachricht oder einer Quittung benötigt werden, verwaltet.

Weil sich das Objekt Verbindungsinformationen bereits auf den internen Aufbau eines Transferagenten bezieht, aber z.B. im Gegensatz zum Nachrichtenspeicher kein allgemeines Konzept eines E-Mail-Systems darstellt, ist dieses Objekt nicht im Problembereichsmodell aufgeführt. Die im Objekt Verbindungsinformationen gespeicherten Daten sind im Problembereichsmodell ein Bestandteil des Werkzeugobjektes Transferagent (siehe Abbildung 4.7 im Abschnitt 4.4.2).

Das Objekt Verbindungsinformationen verwaltet eine Tabelle, in der alle Benutzer aufgeführt sind, deren Nachrichtenspeicher fremden Transferagenten zugeordnet sind. In dieser Tabelle wird zu den aufgeführten Benutzern jeweils die nächste Zwischenstation angegeben, an die Nachrichten und Quittungen weitergeleitet werden sollen. Bei den Zwischenstationen handelt es sich immer um fremde Transferagenten. Zu den angegebenen Transferagenten muß eine Verbindung bestehen. Die hier beschriebene Tabelle wird im folgenden als *Routing-Tabelle* bezeichnet. Der Aufbau und der Zweck der Routing-Tabelle soll anhand eines Beispiels verdeutlicht werden:

Beispiel: In der Abbildung 4.11 ist ein einfaches Netzwerk skizziert. Dieses Netzwerk besteht aus den drei Transferagenten TA_a, TA_b und TA_c. Mit Hilfe dieses Netzwerks können die Benutzer Ben_X, Ben_Y und Ben_Z Nachrichten austauschen. Der Nachrichtenspeicher des Benutzers Ben_X ist dem Transferagenten TA_a zugeordnet. Die Nachrichtenspeicher der Benutzer Ben_Y und Ben_Z sind dem Transferagenten TA_b zugeordnet. Der Transferagent TA_c verwaltet keinen Nachrichtenspeicher. Die Transferagenten TA_a und TA_c sowie die Transferagenten TA_b und TA_c sind miteinander verbunden. Zwischen den Transferagenten TA_a und TA_b besteht keine Verbindung. Die Routing-Tabellen, die von den Objekten Verbindungsinformation der drei Transferagenten verwaltet werden, sind in der Tabelle 4.3 dargestellt.

Wenn z.B. der Benutzer Ben_X eine Nachricht an den Benutzer Ben_Z senden möchte, wird diese Nachricht zunächst vom Benutzeragenten des Benutzers Ben_X an den Transferagenten TA_a übergeben. Dieser Transferagent sendet die Nachricht an den Transferagenten TA_c weiter. TA_c entnimmt aus seiner Routing-Tabelle, daß er die Nachricht an den Transferagenten TA_b übergeben muß. TA_b nimmt die Nachricht in Empfang und legt sie im Nachrichtenspeicher des Benutzers Ben_Z ab.

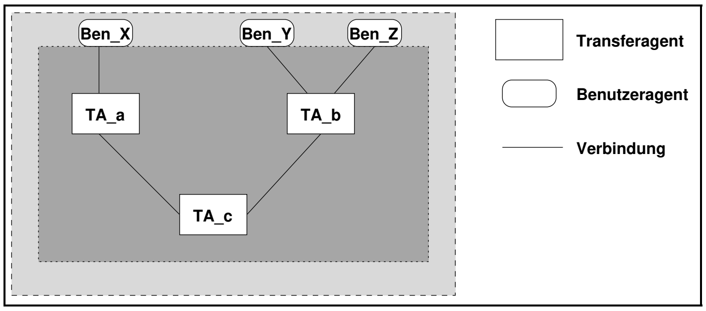

Abbildung 4.11: Beispiel für ein einfaches Netzwerk.

Transferagent TA_a		Transferagent TA_b		Transferagent TA_c	
Empfänger	weiterleiten an	Empfänger	weiterleiten an	Empfänger	weiterleiten an
Ben_Y	TA_c	Ben_X	TA_c	Ben_X	TA_a
Ben_Z	TA_c			Ben_Y	TA_b
				Ben_Z	TA_b

Tabelle 4.3: Routing-Tabellen für die Transferagenten aus Abbildung 4.11.

Ein Transferagent benötigt die Routing-Tabelle nur dann, wenn er für den Empfänger einer Nachricht oder Quittung keinen Nachrichtenspeicher verwaltet. Die Adressen der Benutzer, für die Nachrichtenspeicher verwaltet werden, sind in der Routing-Tabelle eines Transferagenten deshalb nicht aufgeführt.

Während das Objekt Verbindungsinformationen das einzige Entitätsobjekt ist, dem kein Materialobjekt im Problembereichsmodell zugeordnet ist, zeigen sich bei der Modellierung der Werkzeuge wesentlich größere Unterschiede zwischen dem Analysemodell und dem Problembereichsmodell. So geben die im Problembereichsmodell aufgeführten Werkzeugobjekte (siehe Abbildung 4.7 im Abschnitt 4.4.2) zwar zusammen mit den modellierten Beziehungen eine grobe Übersicht über den Aufbau eines E-Mail-Systems. Für eine direkte Übernahme von Objekten und Beziehungen in das Analysemodell ist diese Übersicht jedoch nicht detailliert genug.

Bei der weiteren Beschreibung des Analysemodells wird zunächst der Aufbau eines Benutzeragenten vorgestellt. Anschließend wird der Aufbau eines Transferagenten erläutert. Im letzten Teil werden die Beziehungen betrachtet, die zwischen Benutzer- und Transferagenten bestehen müssen, um ein Nachrichtenaustauschsystem aufzubauen. Verbindungen zwischen Agenten, die im Problembereichsmodell noch als Werkzeuge modelliert wurden, werden bei der weiteren Systementwicklung nicht mehr als Komponenten des Software-Systems berücksichtigt. Verbindungen sind ein Teil der Hardware, die für die Einrichtung des zu modellierenden E-Mail-Systems vorausgesetzt wird.

Aufbau des Benutzeragenten: Abbildung 4.12 beschreibt den Aufbau eines Benutzeragenten des zu modellierenden E-Mail-Systems. Dabei wird für jeden Anwendungsfall (siehe Abschnitt 4.4.1) ein Kontrollobjekt modelliert (siehe auch Jacobson et al. [JCJÖ93, S. 191]). Die Zuordnung von Kontrollobjekten zu Anwendungsfällen ist in der Tabelle 4.4 dargestellt. Zusätzlich besitzt ein Benutzeragent noch die beiden Schnittstellenobjekte BA_Benutzerschnittstelle und BA_Client_Kommunikation.

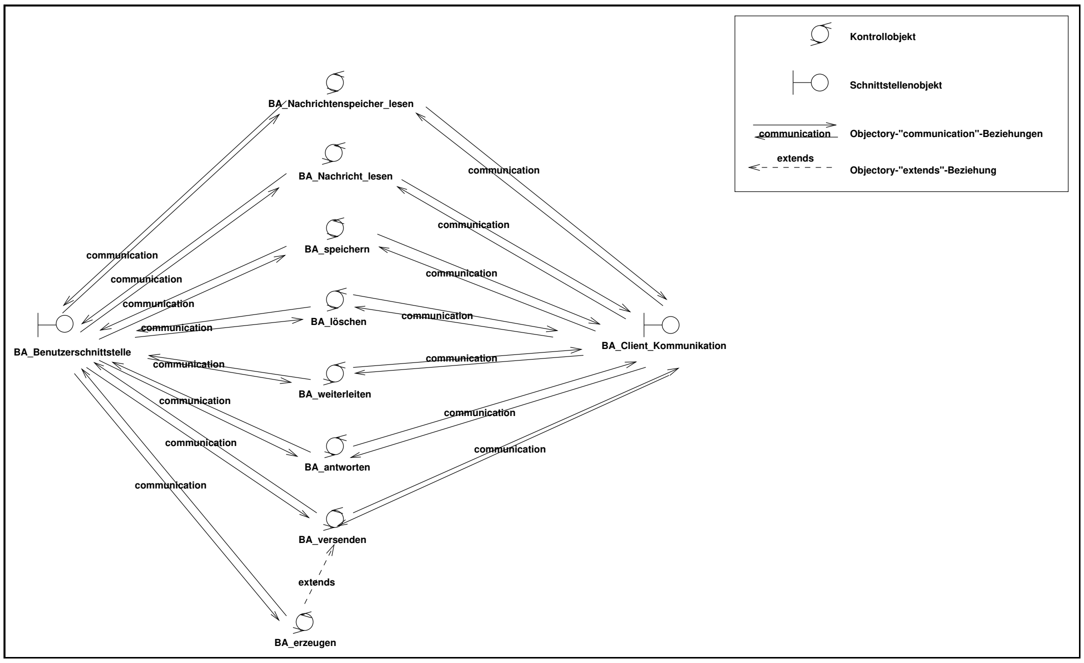

Abbildung 4.12: Der Benutzeragent im Analysemodell.

Anwendungsfall	Kontrollobjekt
Nachrichtenspeicherinhalt lesen	BA_Nachrichtenspeicher_lesen
Nachricht lesen	BA_Nachricht_lesen
Nachricht speichern	BA_speichern
Nachricht loeschen	BA_löschen
Nachricht weiterleiten	BA_weiterleiten
Auf Nachricht antworten	BA_antworten
Nachricht versenden	BA_versenden
Nachricht erzeugen	BA_erzeugen

Tabelle 4.4: Anwendungsfälle für den Benutzeragenten und zugeordnete Kontrollobjekte.

Mit Hilfe von „communication"-Beziehungen wird dargestellt, welche Objekte Stimuli austauschen können. Ein Objekt, von dem eine Kante zur Darstellung einer „communication"-Beziehung ausgeht, kann Stimuli an das Objekt senden, auf das die Kante zeigt. Bei „communication"-Beziehungen zwischen Objekten stimmt die in der Objectory-Dokumentation benutzte Terminologie wieder mit der Terminologie überein, die Jacobson et al. verwenden [JCJÖ93, S. 190] (im Gegensatz zur Bezeichnung von Beziehungen, die den Austausch von Stimuli zwischen Anwendungsfallinstanzen und Instanzen von Akteuren anzeigen, siehe Abschnitt 3.4.1). Eine Kante, die eine „communication"-Beziehung symbolisiert, geht immer von dem Objekt aus, das Stimuli [JCJÖ93, S. 47] an ein anderes Objekt senden kann, um dort die Ausführung von Operationen zu veranlassen. Die Kante zeigt auf das Objekt, das den Stimulus empfängt und evtl. eine Reaktion zurückschickt. Für ein mögliches Zurückschicken von Informationen als Reaktion auf einen Stimulus wird keine neue „communication"-Beziehung eingetragen [Ory94a, S. 39]. Zur Unterscheidung, bei welchem Informationstransfer es sich um einen eigenständigen Stimulus handelt bzw. welcher Informationstransfer als Reaktion auf einen empfangenen Stimulus gilt, werden in der Dokumentation zu Objectory die folgenden Regeln angegeben [Ory94c, S.22]:

- Informationen, die von Entitätsobjekten an Kontrollobjekte oder an Schnittstellenobjekte gesendet werden, sind immer Reaktionen auf Stimuli. Es existiert also keine „communication"-Beziehung, die von einem Entitätsobjekt ausgeht und bei einem Kontrollobjekt oder bei einem Schnittstellenobjekt endet.
- Informationen die von Schnittstellenobjekten oder von Kontrollobjekten an andere Objekte gesendet werden, werden immer als eigenständige Stimuli versendet. Dies gilt auch dann, wenn ein Objekt A Informationen an ein Objekt B sendet, die vorher durch einen Stimulus, der vom Objekt B an das Objekt A gesendet wurde, angefordert wurden.

  Nach den Ausführungen in der Objectory-Dokumentation werden die von Schnittstellenobjekten bzw. von Kontrollobjekten versendeten Informationen nicht als Reaktionen auf Stimuli angesehen, weil Schnittstellenobjekte und Kontrollobjekte im Gegensatz zu Entitätsobjekten, die Möglichkeit haben, den Ereignisfluß eigenmächtig zu bestimmen. Es kann nicht vorausgesehen werden, ob ein Kontrollobjekt oder ein Schnittstellenobjekt, das einen Stimulus empfängt, so auf diesen Stimulus reagiert, wie es erwartet wird.

Eine Kante, mit der eine „communication"-Beziehung dargestellt wird, ist nicht exklusiv einem bestimmten Stimulus zugeordnet. Auch dann, wenn von einem Objekt A verschiedenartige Stimuli an ein Objekt B gesendet werden können, wird nur genau eine Kante zwischen dem Objekt A und dem Objekt B eingetragen.

Die Anzahl der Instanzen, die im Rahmen einer „communication"-Beziehung Stimuli vom Initiator einer Kommunikation empfangen können, wird als Kardinalität der entsprechenden „communication"-Beziehung zwischen Objekten bezeichnet. In dieser Diplomarbeit wird die Kardinalität von „communication"-Beziehungen in Anlehnung an Jacobson et al. nicht explizit in den aufgeführten Abbildungen angegeben. Falls die explizite Angabe der Anzahl der Instanzen, die im Rahmen einer „communication"-Beziehung Stimuli von Initiator einer Kommunikation empfangen können, für das Verständnis einer Abbildung erforderlich ist, wird diese Angabe in dem Textabschnitt gemacht, in dem die betroffene Abbildung beschrieben wird. Zusätzlich sind alle Kardinalitäten in den Beschreibungen der Beziehungen zu den erstellten Modellen im Zusatzdokument zu dieser Diplomarbeit aufgeführt. Im folgenden soll nun auf den Austausch von Informationen des in der Abbildung 4.12 dargestellten Benutzeragenten eingegangen werden.

Für die Kommunikation mit dem Benutzer des Nachrichtenaustauschsystems ist das Objekt BA_Benutzerschnittstelle zuständig. Wenn ein Benutzer des E-Mail-Systems einen Anwendungsfall ausgewählt hat, sendet dieses Objekt einen Stimulus an das Kontrollobjekt, das die Ausführung des gewählten Anwendungsfalls kontrolliert. Das Kontrollobjekt BA_erzeugen ist bei der Auswahl eines Anwendungsfalls durch den Benutzer noch als potentieller Empfänger eines Stimulus vom Objekt BA_Benutzerschnittstelle ausgenommen, da der Anwendungsfall Nachricht erzeugen, dem das Objekt BA_erzeugen zugeordnet ist, nur als Erweiterung des Anwendungsfalls Nachricht versenden ausgeführt werden kann (siehe Abbildung 4.3 im Abschnitt 4.4.1).

Die Kontrollobjekte benötigen zur Kontrolle der Ausführung der Anwendungsfälle immer zusätzliche Eingaben des Benutzers, wie z.B. den Namens einer Datei, in der der Text eines Nachrichtenkörpers gespeichert ist (siehe Beschreibung des Anwendungsfalls Nachricht versenden im Abschnitt 4.4.1). Operationen für die hierfür nötige Kommunikation mit dem Benutzer werden ebenfalls vom Objekt BA_Benutzerschnittstelle zur Verfügung gestellt. Damit eine benötigte Kommunikationsoperation ausgeführt wird, sendet ein Kontrollobjekt einen Stimulus an das Objekt BA_Benutzerschnittstelle. Die hierfür benötigte „communication"-Beziehung wird durch die Kante dargestellt, die von dem Kontrollobjekt ausgeht und beim Objekt

BA_Benutzerschnittstelle endet. Das Objekt BA_Benutzerschnittstelle sendet Eingaben des Benutzers an das Kontrollobjekt zurück. Da es sich bei dem Objekt BA_Benutzerschnittstelle um ein Schnittstellenobjekt, und nicht um ein Entitätsobjekt, handelt, wird für das Zurückschicken der Benutzereingaben an das betroffene Kontrollobjekt ein eigenständiger Stimulus versendet (siehe oben).

Operationen für die Kommunikation mit dem Transferagenten stellt das Objekt BA_Client_Kommunikation bereit. Dieses Objekt empfängt Stimuli von den Kontrollobjekten, die die Ausführung der Anwendungsfälle kontrollieren. Diese Stimuli werden für die Kommunikation mit dem Transferagenten aufbereitet und an den Transferagenten weitergeschickt. Antworten des Transferagenten werden vom Schnittstellenobjekt BA_Client_Kommunikation empfangen und an die Kontrollobjekte zurückgegeben. Falls der Transferagent nicht innerhalb einer maximalen Wartezeit auf einen weitergegebenen Stimulus geantwortet hat, wird die Fehlermeldung *Timeout* an das entsprechende Kontrollobjekt zurückgegeben.

Aufbau des Transferagenten: Der Aufbau eines Transferagenten ist in der Abbildung 4.13 skizziert. Analog zum Benutzeragenten wird für jeden möglichen Anwendungsfall ein Kontrollobjekt modelliert. Die Zuordnung von Kontrollobjekten zu Anwendungsfällen ist in der Tabelle 4.5 dargestellt.

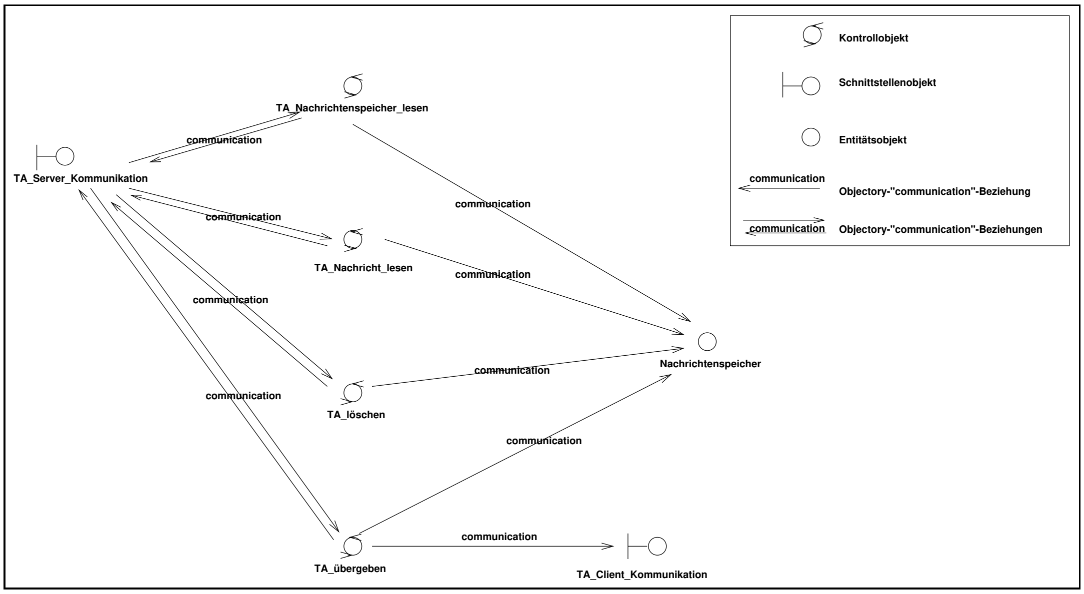

Abbildung 4.13: Der Transferagent im Analysemodell.

Anwendungsfall	Kontrollobjekt
Nachrichtenspeicherinhalt holen	TA_Nachrichtenspeicher_lesen
Nachricht aus Nachrichtenspeicher holen	TA_Nachricht_lesen
Nachricht aus Nachrichtenspeicher entfernen	TA_löschen
Nachricht übergeben	TA_übergeben

Tabelle 4.5: Anwendungsfälle für den Transferagenten und zugeordnete Kontrollobjekte.

Zusätzlich zu den Kontrollobjekten sind in der Abbildung 4.13 noch die Objekte Nachrichtenspeicher, TA_Server_Kommunikation und TA_Client_Kommunikation aufgeführt. Das bei der Erläuterung der Abbildung 4.9 beschriebene Objekt Verbindungsinformationen wird erst bei der Beschreibung der „communication“-Beziehung zwischen verschiedenen Agenten benötigt.

Das Entitätsobjekt Nachrichtenspeicher wurde bereits bei der Erklärung der Abbildung 4.9 vorgestellt. Ein Transferagent besitzt für jeden Benutzeragenten, der ihm zugeordnet ist, jeweils ein Objekt Nachrichtenspeicher. Es ist also möglich, daß ein Transferagent genau einen, mehrere oder gar keinen Nachrichtenspeicher verwaltet. Gemäß den Festlegungen in der Objectory-Dokumentation werden alle Informationen, die vom Entitätsobjekt Nachrichtenspeicher an eines der Kontrollobjekte versendet werden, als Reaktionen auf Stimuli der Kontrollobjekte angesehen (siehe oben). Deshalb werden keine „communication"-Beziehungen modelliert, die vom Objekt Nachrichtenspeicher ausgehen.

Die Objekte TA_Server_Kommunikation und TA_Client_Kommunikation dienen zur Kommunikation mit Benutzeragenten und mit anderen Transferagenten. Durch das Objekt TA_Server_Kommunikation werden Aufträge von Benutzeragenten oder von anderen Transferagenten empfangen. Bei Aufträgen eines Benutzeragenten sorgt das Objekt TA_Server_Kommunikation zusätzlich für die Rückgabe der Ergebnisse des bearbeiteten Auftrags an den auftraggebenden Benutzeragenten. Das Objekt TA_Client_Kommunikation ist für die Weiterleitung von Nachrichten und Quittungen an fremde Transferagenten zuständig.

Wenn ein Benutzeragent einen Auftrag empfängt, wird ein Stimulus an eines der vier Kontrollobjekte gesendet. Das Kontrollobjekt, das den Stimulus empfängt, sorgt für die Ausführung des Anwendungsfalls, dem es nach der Tabelle 4.5 zugeordnet ist. Die Kontrollobjekte TA_Nachrichtenspeicher_lesen, TA_Nachricht_lesen und TA_löschen senden bei der Ausführung der jeweiligen Anwendungsfälle Stimuli an ein Objekt Nachrichtenspeicher und empfangen eine Reaktion von diesem Objekt. Diese Reaktion wird anschließend mit Hilfe eines neuen Stimulus an das Objekt TA_Server_Kommunikation weitergegeben. Das Objekt TA_Server_Kommunikation schickt darauf eine Antwort an den Benutzeragenten oder Transferagenten, der den Auftrag aufgegeben hat, zurück.

Beispiel: Wenn das Objekt TA_Server_Kommunikation von einem Benutzeragenten den Auftrag erhält, ein Inhaltsverzeichnis des Nachrichtenspeichers eines Benutzers anzuzeigen, wird ein Stimulus an das Objekt TA_Nachrichtenspeicher_lesen gesendet. Das Objekt TA_Nachrichtenspeicher_lesen leitet den Stimulus an das Objekt Nachrichtenspeicher, das den Nachrichtenspeicher des Benutzers enthält, weiter, sofern dieser Nachrichtenspeicher existiert. Existiert der gesuchte Nachrichtenspeicher, werden die Informationen, die sich auf den Umschlägen der im Nachrichtenspeicher gesammelten Nachrichten befinden, zusammengefaßt und als Reaktion an das Objekt TA_Nachrichtenspeicher_lesen zurückgeschickt. Das Objekt TA_Nachrichtenspeicher_lesen sendet anschließend einen Stimulus an das Objekt TA_Server_Kommunikation und gibt dabei die empfangenen Informationen weiter. Vom Objekt TA_Server_Kommunikation aus werden die erhaltenen Informationen an den Benutzeragenten verschickt.

Existiert der gesuchte Nachrichtenspeicher nicht, übergibt das Objekt TA_Nachrichtenspeicher_lesen eine Fehlermeldung an das Objekt TA_Server_Kommunikation. Das Objekt TA_Server_Kommunikation versendet die Fehlermeldung anschließend an den Benutzeragenten, der Auftrag aufgegeben hat.

Das Kontrollobjekt TA_übergeben, das die Ausführung des Anwendungsfalls Nachricht übergeben kontrolliert, ist zusätzlich noch durch eine „communication"-Beziehung mit dem Objekt TA_Client_Kommunikation verbunden. An das Objekt TA_Client_Kommunikation werden Nachrichten oder Quittungen übergeben, die an fremde Transferagenten weitergeleitet werden sollen.

Der Anwendungsfall Nachricht übergeben (siehe Abschnitt 4.4.1) wird ausgeführt, wenn das Objekt TA_Server_Kommunikation eine Quittung oder eine Nachricht empfängt. Das Objekt TA_Client_Kommunikation sorgt für die Weiterleitung der Nachricht bzw. Quittung an einen anderen Transferagenten, falls die Weiterleitung nötig ist. Auf den Informationsfluß zwischen Objekten und auf die Behandlung von Ausnahmen wird detailliert bei der Beschreibung des Entwurfsmodells im Abschnitt 4.6 eingegangen.

Beziehungen zwischen Agenten: Für die Kommunikation des Benutzeragenten mit einem Transferagenten ist das Objekt BA_Client_Kommunikation zuständig (siehe Abbildung 4.12). Für die Kommunikation eines Transferagenten sorgen die Objekte TA_Server_Kommunikation und TA_Client_Kommunikation (siehe Abbildung 4.13). Die Beziehungen zwischen Benutzer- und Transferagenten bzw. zwischen zwei Transferagenten sind in der Abbildung 4.14 skizziert.

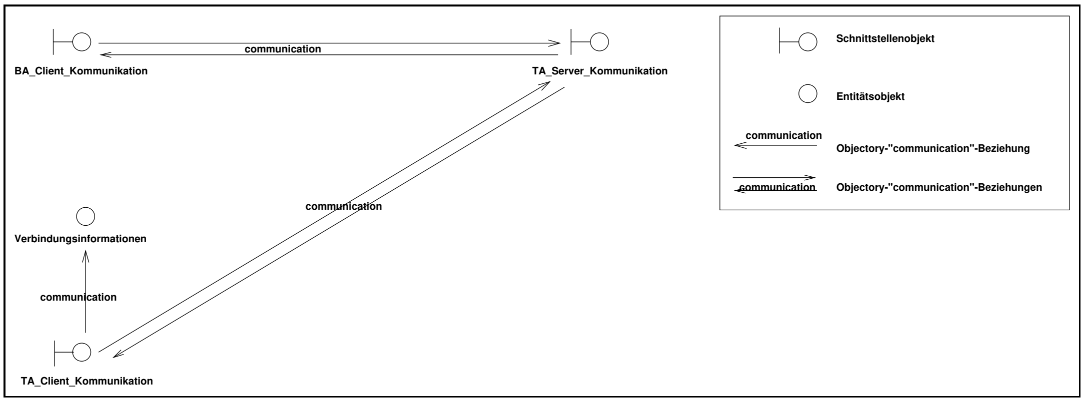

Abbildung 4.14: Beziehungen zwischen den Agenten.

Diese Abbildung wird im folgenden anhand der skizzierten Objekte erläutert:

Das Objekt BA_Client_Kommunikation: Ein Benutzeragent vergibt über das Objekt BA_Client_Kommunikation Aufträge an einen Transferagenten und empfängt Informationen des Transferagenten. Dies wird durch die „communication"-Beziehungen zwischen den Objekten BA_Client_Kommunikation und TA_Server_Kommunikation dargestellt.

Das Objekt TA_Server_Kommunikation: Über das Objekt TA_Server_Kommunikation nimmt ein Transferagent Aufträge entgegen. Diese Aufträge können von einem Benutzeragenten oder von einem anderen Transferagenten stammen. Bei den Aufträgen kann es sich um Befehle zum Weiterleiten von Nachrichten oder Quittungen oder um Befehle zum Lesen bzw. zur Bearbeitung des Nachrichtenspeichers eines Benutzers handeln. Das Objekt TA_Server_Kommunikation kann auch Informationen an die Objekte TA_Client_Kommunikation und BA_Client_Kommunikation senden (z.B. zur Angabe einer Fehlermeldung, wenn das Format einer übergegebenen Nachricht nicht gelesen werden konnte).

Das Objekt TA_Server_Kommunikation unterhält „communication"-Beziehungen zu keinem, genau einem oder mehreren Objekten TA_Client_Kommunikation und BA_Client_Kommunikation.

Das Objekt TA_Client_Kommunikation: Ein Transferagent leitet über das Objekt TA_Client_Kommunikation Nachrichten oder Quittungen an fremde Transferagenten weiter. Dies wird durch die „communication"-Beziehung zwischen den Objekten TA_Client_Kommunikation und TA_Server_Kommunikation verdeutlicht. Die Objekte TA_Client_Kommunikation und TA_Server_Kommunikation stammen von verschiedenen Transferagenten. Die Information, an welchen Transferagent eine Nachricht oder eine Quittung weitergeleitet werden muß, erhält das Objekt TA_Client_Kommunikation durch das Objekt Verbindungsinformationen.

Das Objekt TA_Client_Kommunikation unterhält „communication"-Beziehung zu keinem, genau einem oder mehreren Objekten TA_Server_Kommunikation.

Das Objekt Verbindungsinformationen: Dieses Entitätsobjekt enthält die Routing-Tabelle, in der angegeben wird, wohin eine Nachricht bzw. Quittung weitergeleitet werden muß (siehe Seite 71f.).

4.6 Erstellung des Entwurfsmodells und des Prototyps

Durch den ProSet-Prototyp soll die Erstellung des Objectory-Entwurfsmodells unterstützt werden. Die Interaktionen, die zwischen den einzelnen *Systemkomponenten* stattfinden, sollen mit Hilfe des zu erstellenden Prototyps simuliert werden können, wie es von Bischofberger und Pomberger für das experimentelle Prototyping gefordert wird [BP92, S. 17]. Die Systemkomponenten sind bei dem betrachteten Fallbeispiel die Objekte im Entwurfsmodell.

Die Simulation der Interaktionen zwischen Entwurfsmodellobjekten soll zum einen zur Validierung erstellter Teile des Entwurfsmodells dienen, zum anderen werden Aufschlüsse für nötige Änderungen der Systemarchitektur und für die Weiterentwicklung des Entwurfsmodells erwartet. Für Objekte im Entwurfsmodell schlagen Jacobson et al. die Bezeichnung *Block* (*block*) vor [JCJÖ93, S. 146]. Die Bezeichnungen Block und Objekt werden im folgenden bei der Beschreibung des Entwurfsmodells synonym benutzt.

Damit eine Validierung und Weiterentwicklung des Entwurfsmodells mit Hilfe eines ProSet-Prototyps sinnvoll ist, wird vorausgesetzt, daß es sich bei der Programmiersprache, in der das System für die Endanwender implementiert wird, um eine imperative Programmiersprache handelt, die ein ähnliches Modulkonzept anbietet wie ProSet. Die Entwicklung des Prototyps und des Entwurfsmodells erfolgt bei dem betrachteten Fallbeispiel in drei Teilschritten:

- Zunächst wird ein Entwurfsmodell erstellt, in dem nur der Benutzeragent berücksichtigt wird. Transferagenten und Beziehungen zwischen Benutzer- und Transferagenten werden nicht beachtet. Anschließend wird ein Prototyp für den modellierten Benutzeragenten erstellt. Dieser Prototyp liefert Aufschlüsse über nötige Änderungen des bis zu diesem Zeitpunkt erstellten Entwurfsmodells. Die nötigen Änderungen des Entwurfsmodells werden nach der Erstellung des Prototyps durchgeführt.

- Im zweiten Schritt wird ein separates Entwurfsmodell für den Transferagenten erstellt. Dabei werden nur die Abläufe innerhalb eines Transferagenten betrachtet. Die Kommunikation mit anderen Transferagenten und mit Benutzeragenten bleibt noch unberücksichtigt. Auch für dieses Modell wird ein Prototyp erstellt. Durch das Prototyping gewonnene Erkenntnisse können zu einer Überarbeitung des betrachteten Entwurfsmodells führen.

- Im letzten Schritt werden die Beziehungen, die zwischen Benutzeragenten und Transferagenten, sowie zwischen verschiedenen Transferagenten bestehen, betrachtet. Dabei wird auf die in den ersten beiden Phasen erstellten Entwurfsmodelle und Prototypen zurückgegriffen. Sobald ein Prototyp eines E-Mail-Systems mit Benutzer- und Transferagenten implementiert ist, wird die Arbeit an diesem Fallbeispiel mit einem Ausblick auf weitere nötige Arbeiten am Entwurfsmodell beendet.

Die aufgeführten Teilschritte werden im folgenden in separaten Abschnitten vorgestellt.

4.6.1 Modellierung des Benutzeragenten

Bei der Modellierung des Benutzeragenten werden die folgenden Schritte durchgeführt:

Erster Schritt: Zuerst werden die Blöcke (Entwurfsmodellobjekte), aus denen der Benutzeragent besteht, sowie ihre Beziehungen modelliert.

Zweiter Schritt: Für die einzelnen Anwendungsfälle werden *Interaktionsdiagramme* (*interaction diagrams*) erstellt (siehe Jacobson et al. [JCJÖ93, S. 215ff.]). In einem Interaktionsdiagramm wird beschrieben, welche Arten von Stimuli in welcher Reihenfolge zwischen den Blöcken des zu entwickelnden Systems bei der Ausführung eines Anwendungsfalls versendet werden und welche Aktivitäten in einzelnen Blöcken ausgeführt werden.

Dritter Schritt: Der Prototyp des Benutzeragenten wird auf der Basis der entworfenen Systemstruktur und der Interaktionsdiagramme implementiert.

Vierter Schritt: Der Entwurf der Struktur des Benutzeragenten und die Interaktionsdiagramme werden unter Berücksichtigung der Ergebnisse des Prototyping überarbeitet.

Der Aufbau eines Benutzeragenten, der sich nach der Durchführung des oben vorgestellten vierten Schrittes ergibt, ist in der Abbildung 4.15 dargestellt. Gemäß der Forderung von Jacobson et al. bleibt die Systemstruktur, die bereits im Analysemodell modelliert wurde (siehe Abbildung 4.12 im Abschnitt 4.5), dabei weitgehend erhalten. Änderungen gegenüber der im Analysemodell vorgestellten Systemstruktur ergeben sich nur durch die Vorgaben, die durch die im Analysemodell noch nicht berücksichtigte Implementierungsumgebung gemacht werden [JCJÖ93, S. 144].

Zur Modellierung der „communication"-Beziehungen werden die gleichen Regeln, die bereits bei der Modellierung dieser Beziehungen im Analysemodell angewendet wurden, benutzt. Um das Versenden von Stimuli zu ermöglichen, werden also „communication"-Beziehungen modelliert, während für Informationen, die als Reaktionen auf Stimuli versendet werden, keine „communication"-Beziehungen modelliert werden. Objekte, die für die Kontrolle von Anwendungsfällen oder für die Kontrolle des Programmablaufs zuständig sind und Objekte, die für die Kommunikation mit der Außenwelt zuständig sind, versenden Informationen immer als eigenständige Stimuli. Im Gegensatz dazu versenden Objekte, die zur Speicherung von Daten dienen (solche Objekte entsprechen Entitätsobjekten im Analysemodell), Informationen, die an Objekte, die zur Kontrolle von Anwendungsfällen bzw. zur Kontrolle des Programmablaufs oder für die Kommunikation mit der Außenwelt modelliert wurden, gesendet werden sollen, immer als Reaktionen auf Stimuli dieser Objekte.

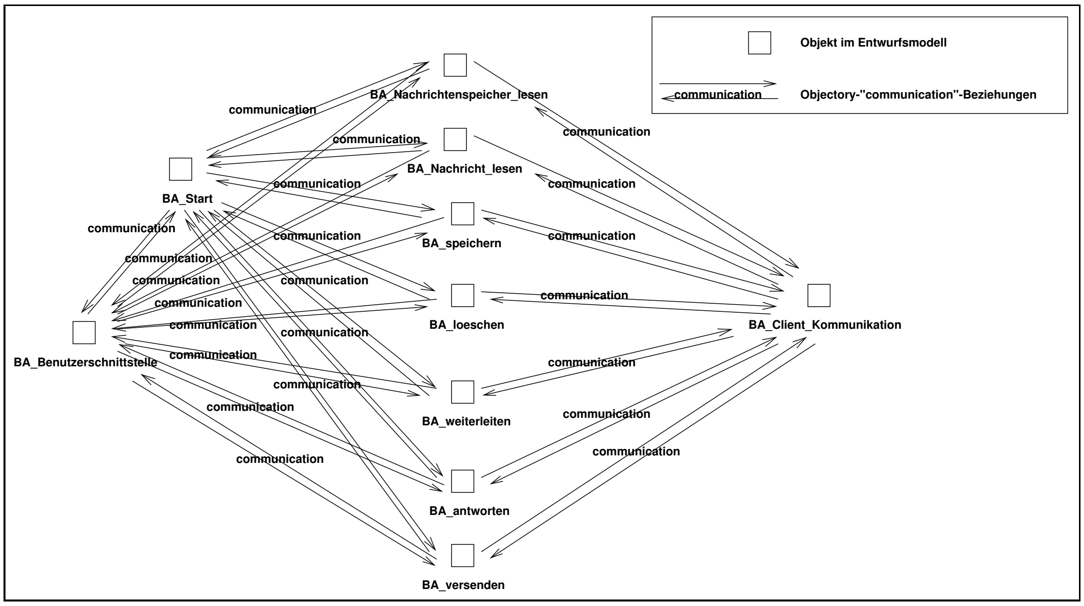

Abbildung 4.15: Der Benutzeragent im Entwurfsmodell.

Die Abweichungen der in der Abbildung 4.15 vorgestellten Modellierung des Benutzeragenten von der Modellierung des Benutzeragenten im Analysemodell werden im folgenden erläutert:

- Wenn im Namen eines Analysemodellobjektes ein Umlaut (ä, ö, ü) benutzt wird, wird dieser Umlaut im Namen des Entwurfsmodellobjektes durch ae, oe oder ue ersetzt, weil auf der der Tastatur der Computer, die zur Programmierung zur Verfügung stehen, keine Umlaute vorhanden sind. Hiervon ist beim Benutzeragenten das Objekt BA_loeschen (Schreibweise für das Entwurfsmodell) betroffen. Die Änderung der Objektnamen wurde bereits vor der Erstellung des Prototyps durchgeführt.

- Zusätzlich zu den Objekten des Anlysemodells wird das Objekt BA_Start aufgenommen. Wenn der Benutzer des E-Mail-Systems aufgefordert werden soll, einen Anwendungsfall auszuwählen, sendet das Objekt BA_Start zunächst einen Stimulus an das Objekt BA_Benutzerschnittstelle, um abfragen zu lassen, welcher Anwendungsfall ausgeführt werden soll. Das Objekt BA_Benutzerschnittstelle sendet die Eingabe des Benutzers anschließend an das Objekt BA_Start zurück. Abhängig von der Auswahl des Benutzers wird ein Stimulus an eines der Objekte, die für die Kontrolle der Anwendungsfälle zuständig sind, verschickt. Bei den Objekten, die für die Kontrolle der Anwendungsfälle zuständig sind, handelt es sich um die Objekte BA_Nachrichtenspeicher_lesen, BA_Nachricht_lesen, BA_speichern, BA_loeschen, BA_weiterleiten, BA_antworten und BA_versenden. Wenn die Ausführung eines Anwendungsfalls beendet ist, sendet das Objekt zur Kontrolle des Anwendungsfalls einen Stimulus an das Objekt BA_Start, um die Auswahl eines neuen Anwendungsfalls zu ermöglichen.

  Die Modellierung des Objektes BA_Start beruht auf Erkenntnissen, die bei der Konstruktion des Prototyps gewonnen wurden. Der Grund für die Abweichung vom Analysemodell wird im folgenden erläutert. Exemplarisch sollen dabei die Objekte BA_Benutzerschnittstelle und BA_versenden betrachtet werden. Das Objekt BA_versenden könnte bei dieser Beschreibung jedoch beliebig durch eines der anderen betroffenen Objekte BA_Nachrichtenspeicher_lesen, BA_Nachricht_lesen, BA_speichern, BA_loeschen, BA_weiterleiten oder BA_antworten ersetzt werden.

  Im ProSet-Programm wird sowohl ein Modul BA_Benutzerschnittstelle als auch ein Modul BA_versenden implementiert. Das Versenden eines Stimulus wird im ProSet-Programm auf eine der beiden folgenden Arten realisiert:

  - Das Versenden eines Stimulus wird durch einen Prozeduraufruf simuliert. Dies ist beispielsweise der Fall, wenn ein Objekt zur Kontrolle eines Anwendungsfalls eine Operation aufruft, die vom Objekt BA_Benutzerschnittstelle zur Verfügung gestellt wird, um eine Eingabe eines Benutzers zu lesen (z.B. zur Eingabe eines Dateinamens).
  - Das Versenden eines Stimulus wird durch einen ProSet-return-Befehl [DFH+95, S. 16f.] am Ende einer Prozedur simuliert. Dies ist beispielsweise der Fall, wenn das Objekt BA_Benutzerschnittstelle die Eingabe des Benutzers an das Objekt zur Anwendungsfallkontrolle zurücksenden soll. Da es sich bei dem Objekt BA_Benutzerschnittstelle um ein Objekt zur Kommunikation mit der Außenwelt handelt, wird zum Verschicken dieser Informationen immer ein neuer Stimulus benötigt (siehe oben).

  Wenn das Objekt BA_versenden einen Stimulus an das Objekt BA_Benutzerschnittstelle verschickt, um Eingaben eines Benutzers einlesen zu lassen, wird in einer Prozedur einer Modulinstanz des Moduls BA_versenden eine Prozedur einer Modulinstanz des Moduls BA_Benutzerschnittstelle aufgerufen. Dort werden die Benutzereingaben gelesen und anschließend an das Objekt BA_versenden mit Hilfe des return-Befehls zurückgegeben.

  Wenn das Objekt BA_Start nicht neu aufgenommen und stattdessen die im Analysemodell dargestellte Struktur übernommen worden wäre, ergäben sich dadurch Probleme, daß die Moduln BA_Benutzerschnittstelle und BA_versenden sich gegenseitig instanziieren müßten. Hierauf wird im folgenden genauer eingegangen:

  - Zum Start des Anwendungsfalls Nachricht versenden müßte eine Prozedur in einer Modulinstanz des Moduls BA_Benutzerschnittstelle eine Prozedur in einer Modulinstanz des Moduls BA_versenden aufrufen.
  - Wenn bei der Ausführung des Anwendungsfalls Nachricht versenden Eingaben des Benutzers benötigt werden, müßte von einer Prozedur einer Instanz des Moduls BA_versenden eine Prozedur in einer Instanz des Moduls BA_Benutzerschnittstelle aufgerufen werden.

  Damit die Instanz des Moduls BA_Benutzerschnittstelle benötigte Prozeduren in einer Instanz des Moduls BA_versenden aufrufen kann, muß die Instanz des Moduls BA_Benutzerschnittstelle eine Instanz des Moduls BA_versenden kennen. Dies kann dadurch erreicht werden, daß bei der Instanziierung des Moduls BA_Benutzerschnittstelle eine Instanz des Moduls BA_versenden exportiert wird. Die Instanziierung des Moduls BA_Benutzerschnittstelle sähe dann folgendermaßen aus:

```
program Hauptprogramm;
   ...
   ba_benutzerschnittstelle :=
      instantiate (closure BA_Benutzerschnittstelle)
   export
      instanz_modul_ba_versenden,   -- Instanz des Moduls BA_versenden
      ...;                          -- weitere Exporte
   import
      ...;                          -- Importe
   end instantiate;
   ...
end Hauptprogramm;
```

Problematisch ist jedoch, daß nicht nur von Prozeduren einer Instanz des Moduls BA_Benutzerschnittstelle Prozeduren in einer Instanz des Moduls BA_versenden aufgerufen werden, sondern daß Prozeduraufrufe in umgekehrter Richtung stattfinden. Um dies zu ermöglichen, müßte die exportierte Instanz instanz_modul_ba_versenden selbst bereits eine Instanz des Moduls BA_Benutzerschnittstelle kennen. Dies ließe sich folgendermaßen realisieren:

Erster Schritt: Zunächst wird eine Instanz des Moduls BA_Benutzerschnittstelle erzeugt. An diese Instanz wird noch keine Instanz des Moduls BA_versenden exportiert (anstelle einer Instanz des Moduls BA_versenden kann eine Instanz eines beliebigen anderen Moduls exportiert werden).

Zweiter Schritt: Eine Instanz des Moduls BA_versenden wird erzeugt. Die im ersten Schritt erstellte Instanz des Moduls BA_Benutzerschnittstelle wird an die erzeugte Instanz des Moduls BA_versenden exportiert.

Dritter Schritt: Eine neue Instanz des Moduls BA_Benutzerschnittstelle wird erzeugt. Die im zweiten Schritt erstellte Instanz des Moduls BA_versenden wird an die neue Instanz des Moduls BA_Benutzerschnittstelle exportiert.

Die dargestellten Ergebnisse lassen sich für die anderen betroffenen Objekte (BA_Nachrichtenspeicher_lesen, BA_Nachricht_lesen, BA_speichern, BA_loeschen, BA_weiterleiten, BA_antworten) verallgemeinern. Die hier beschriebene Art der Implementierung ist zwar nicht falsch, jedoch erscheint die doppelte Instanziierung des Moduls BA_Benutzerschnittstelle unsauber, weil ursprünglich vorgesehen war, jedem Objekt, das Bestandteil eines Benutzeragenten ist, genau eine Instanz eines ProSet-Moduls zuzuordnen. Dieses Ziel wird durch die zusätzliche Einführung des Objektes BA_Start erreicht. Jedem in der Abbildung 4.15 dargestellten Objekt wird so bei der Implementierung genau eine Modulinstanz zugeordnet.

- Für das im Analysemodell aufgeführte Kontrollobjekt BA_erzeugen existiert im Entwurfsmodell kein entsprechendes Objekt mehr. Die Aufgabe dieses Objektes war die Kontrolle der Durchführung des Anwendungsfalls Nachricht erzeugen (siehe Abschnitt 4.4.1).

  Die Klasse des Objektes BA_erzeugen erweitert im Analysemodell die Klasse des Objektes BA_versenden. Dies wird im Analysemodell durch die „extends“-Beziehung zwischen dem Objekt BA_erzeugen und dem Objekt BA_versenden beschrieben. Im Entwurfsmodell wird hingegen bereits berücksichtigt, daß bei den hier betrachteten Programmiersprachen für den Prototyp und für das endgültige System die „extends“-Beziehung nicht direkt implementiert werden kann.

  Wenn für die Implementierung ein Vererbungsmechanismus zur Verfügung stünde, hätte die „extends“-Beziehung im Entwurfsmodell und in der Implementierung durch eine Vererbungsbeziehung ersetzt werden können (die Richtung des Pfeils, der die „extends“-Beziehung symbolisiert, hätte dabei umgedreht werden müssen). Die Klasse (bzw. der Modul) BA_versenden wäre ein Nachkomme der Klasse (bzw. des Moduls) BA_erzeugen.

Da ein Vererbungsmechanismus für die Implementierung nicht zur Verfügung steht, wäre es naheliegend, die „extends“-Beziehung im Analysemodell durch eine „communication“-Beziehung im Entwurfsmodell zu ersetzen, wie es in der Abbildung 4.16 dargestellt wird.

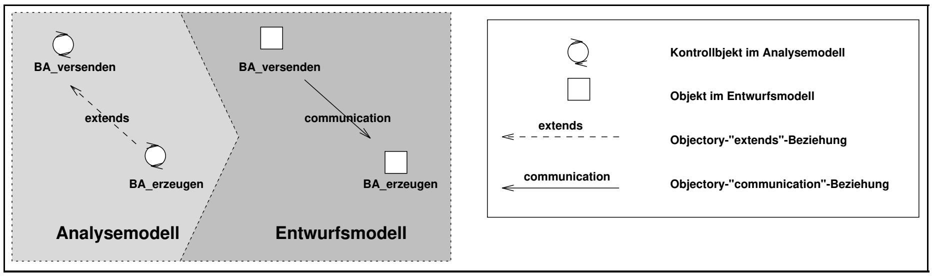

Abbildung 4.16: Ersetzen einer „extends“-Beziehung durch eine „communication“-Beziehung. Diese Skizze soll nur das Prinzip verdeutlichen. Da in dem betrachteten Fallbeispiel das Objekt BA_erzeugen auch Stimuli an das Objekt BA_versenden senden kann, müßte im endgültigen Modell zusätzlich noch eine „communication“-Beziehung vom Objekt BA_erzeugen zum Objekt BA_versenden hinzugefügt werden.

Eine Beziehung zwischen Klassen („extends“-Beziehung) wird so durch eine Beziehung zwischen Instanzen („communication“-Beziehung) ersetzt. Wenn im Objekt BA_versenden auf Routinen zugegriffen werden soll, die im Objekt BA_erzeugen enthalten sind, sendet das Objekt BA_versenden einen Stimulus an das Objekt BA_erzeugen, der die Ausführung der benötigten Routinen veranlaßt. Diese Vorgehensweise schlagen Jacobson et al. vor [JCJÖ93, S. 212].

Die Umwandlung der „extends“-Beziehung im Analysemodell in eine „communication“-Beziehung im Entwurfsmodell wurde bei der Erstellung des Entwurfsmodells auch zunächst durchgeführt. Beim Prototyping hat sich jedoch gezeigt, daß ein eigenes Modul BA_erzeugen überflüssig ist.

Um dies zu verdeutlichen, ist im folgenden die Implementierung des Moduls BA_erzeugen für den zuerst erstellten Prototyp aufgeführt:

```
module BA_erzeugen
   import ba_benutzerschnittstelle;
   export start;

   procedure start();
   begin
      return ba_benutzerschnittstelle.eingabe_nachrichtenkoerper();
         ------------------------------------------------
         -- ba_benutzerschnittstelle ist eine Instanz des
         -- Moduls BA_Benutzerschnittstelle
         ------------------------------------------------
   end start;

end BA_erzeugen;
```

Bei dieser Implementierung wird deutlich, daß die einzige Aufgabe der Instanz des Moduls BA_erzeugen darin besteht, die Prozedur eingabe_nachrichtenkoerper in einer Instanz des Moduls BA_Benutzerschnittstelle aufzurufen. Die Implementierung eines eigenständigen Moduls BA_erzeugen, um diese relativ einfache Aufgabe auszuführen, wird als unnötiger Aufwand angesehen. Der Aufruf der Prozedur eingabe_nachrichtenkoerper wird daher bei einer überarbeiteten Prototyp-Implementierung von einer Instanz des Moduls BA_versenden durchgeführt.

- Die Modellierung der für den Benutzeragenten benötigten Materialien weicht von der Modellierung der Materialien im Analysemodell (siehe Abbildung 4.9 im Abschnitt 4.5) ab. Benötigte Materialien sind in der hier betrachteten Phase der Entwurfsmodellerstellung sämtliche Materialien, die in der Abbildung 4.9 im Abschnitt 4.5 aufgeführt sind mit Ausnahme des Nachrichtenspeichers (Analysemodellobjekt Nachrichtenspeicher) und der Informationen zum Weiterleiten von Nachrichten bzw. Quittungen (Analysemodellobjekt Verbindungsinformationen). Die Objekte Nachrichtenspeicher und Verbindungsinformationen werden erst beim Entwurf des Transferagenten (Abschnitt 4.6.2) bzw. beim Entwurf des vollständigen E-Mail-Systems (Abschnitt 4.6.3) berücksichtigt.

  Die Materialien, die beim Entwurf des Benutzeragenten benötigt werden, sind im Analysemodell noch als Objekte modelliert. Im Entwurfsmodell sind sie hingegen nicht mehr aufgeführt. Dies liegt daran, daß eine Modellierung der vom Benutzeragenten bearbeiteten Materialien (z.B. Nachrichten oder Quittungen) als eigenständige Objekte im Entwurfsmodell eine Repräsentation dieser Materialien als Modulinstanzen in der Implementierung des Prototyps und des Anwendungssystems zur Folge gehabt hätte. Dies hätte jedoch zu Problemen geführt, weil die Materialien oft zwischen verschiedenen Prozessen ausgetauscht werden müssen. Der Austausch von Daten zwischen Prozessen wird in der hier beschriebenen Phase der Entwurfsmodellerstellung und des Prototyping zwar noch nicht im Detail betrachtet, jedoch wird hier bereits berücksichtigt, daß zur Interprozeßkommunikation auf Funktionen zurückgegriffen werden muß, die in der Programmiersprache *C* [KR70] implementiert sind (hierauf wird im Abschnitt 4.6.3 genauer eingegangen). Es ist daher nicht möglich, Modulinstanzen direkt zwischen verschiedenen Prozessen auszutauschen. Eine Umwandlung von Modulinstanzen in Zeichenketten, die zwischen verschiedenen Prozessen verschickt werden könnten und eine anschließende Wiederherstellung der Modulinstanzen wird für eine Implementierung als zu aufwendig eingeschätzt.

  Stattdessen werden für benötigte Materialien Datenstrukturen innerhalb von Moduln bzw. innerhalb von Prozeduren benutzt, die sich möglichst einfach in Zeichenketten umwandeln lassen und die möglichst einfach aus Zeichenketten wiederhergestellt werden können.

Beispiel: Im Modul BA_antworten ist die Prozedur start implementiert. Durch diese Prozedur wird die Ausführung des Anwendungsfalls Auf Nachricht antworten gestartet. Bei der Ausführung dieses Anwendungsfalls wird eine Antwort für eine empfangene Nachricht erzeugt und verschickt (siehe Tabelle 4.1 im Abschnitt 4.4.1).

Das Analysemodellobjekt Nachrichteninhalt wird in dieser Prozedur in einem Tupel gespeichert. Dieses Tupel besitzt Einträge für den Nachrichtenkopf und für den Nachrichtenkörper.

Der Nachrichtenkopf wird durch ein Tupel dargestellt, das aus Einträgen für die Absenderadressen, den Betreff und das Erstellungsdatum besteht.

Der Nachrichtenkörper wird durch ein Tupel dargestellt, in dem für jede Zeile der Nachricht eine Zeichenkette gespeichert ist.

Die beschriebene Datenstruktur ist in der Abbildung 4.17 skizziert.

Diese Datenstruktur eines Nachrichteninhalts, kann zwar, ähnlich wie Modulinstanzen, nicht direkt an C-Funktionen zum Versenden an einen anderen Prozeß übergeben werden, jedoch wird die Umwandlung in eine Zeichenkette und eine Wiederherstellung der alten Struktur aus einer Zeichenkette bei einer solchen Datenstruktur als erheblich einfacher eingeschätzt als bei einer Modulinstanz.

Die Speicherung des Nachrichteninhalts in einem Tupel hat gegenüber einer Speicherung in einer Zeichenkette den Vorteil, daß einfacher auf einzelne Bestandteile, wie z.B. die Absenderadresse, zugegriffen werden kann.

Abbildung 4.17: Beispiel für eine Datenstruktur zur Speicherung des Nachrichteninhalts.

> Die beschriebenen Änderungen bei der Modellierung der Materialien sind kein Ergebnis des Prototyping. Sie wurden bereits vor der Erstellung des Prototyps vorgenommen.

Neben einer Skizze der Objekte des Benutzeragenten und einer Beschreibung der Beziehungen zwischen den Objekten enthält das Entwurfsmodell des E-Mail-Systems noch Interaktionsdiagramme [JCJÖ93, S. 215ff.]. Mit Interaktionsdiagrammen können Abläufe innerhalb der Objekte des Benutzeragenten und insbesondere der Austausch von Stimuli zwischen den Objekten dargestellt werden. Die Entwicklung der Interaktionsdiagramme ist der letzte Schritt vor der Implementierung des Prototyps für den Benutzeragenten. Nach dem Prototyping werden die Interaktionsdiagramme überarbeitet. Das Entwurfsmodell des Benutzeragenten enthält für jeden Anwendungsfall ein Interaktionsdiagramm. Die einzige Ausnahme hiervon bilden die Anwendungsfälle Nachricht versenden und Nachricht erzeugen.

Ursprünglich wurde für jeden dieser beiden Anwendungsfälle ein eigenständiges Interaktionsdiagramm erzeugt. Nach der Entscheidung, aufgrund der Ergebnisse des Prototyping die Kontrolle des Anwendungsfalls Nachricht erzeugen dem Objekt BA_versenden zu übertragen, wurde die Zusammenlegung der Anwendungsfälle Nachricht erzeugen und Nachricht versenden auch bei den Interaktionsdiagrammen berücksichtigt. Die Anwendungsfälle Nachricht versenden und Nachricht erzeugen werden gemeinsam in einem Interaktionsdiagramm beschrieben. Dieses Interaktionsdiagramm ist in der Abbildung 4.18 beispielhaft für alle Interaktionsdiagramme dargestellt. Die restlichen Interaktionsdiagramme befinden sich im Zusatzdokument zur Diplomarbeit.

Jedes Entwurfsmodellobjekt, das an der Ausführung der Anwendungsfälle Nachricht versenden und Nachricht erzeugen beteiligt ist, wird durch einen senkrechten *Balken* (*bar*) repräsentiert. Stimuli werden durch Kanten dargestellt, die vom Initiator einer Kommunikation zum Empfänger gerichtet sind. Bei den eingetragenen Stimuli wird gemäß der Empfehlung von Jacobson et al. zwischen *Intraprozeßkommunikation* (Kommunikation innerhalb eines Prozesses) und *Interprozeßkommunikation* (Kommunikation zwischen verschiedenen Prozessen) unterschieden [JCJÖ93, S. 221f.].

Für Informationen, die als Reaktion auf einen Stimulus an einen anderen Block zurückgesendet werden, werden keine eigenen Stimuli modelliert. Grundlage für die Unterscheidung, wann ein Informationsaustausch als Reaktion gilt und wann für einen Informationsaustausch ein eigener Stimulus modelliert wird, sind wieder die bereits bei der Beschreibung der „communication"-Beziehungen des Analysemodells erwähnten Ausführungen in der Objectory-Dokumentation (siehe Abschnitt 4.5). Für Informationen, die von einem Entwurfsmodellobjekt, das zur Speicherung von Daten dient (ein solches Objekt wäre im Analysemodell als Entitätsobjekt modelliert), an ein Objekt zur Kontrolle eines Anwendungsfalls bzw. zur Kontrolle des Programmablaufs oder an ein Objekt zur Durchführung der Kommunikation mit der Außenwelt geschickt werden, werden grundsätzlich keine Stimuli modelliert. Solche Informationen werden immer als direkte Reaktionen auf vorausgegangene Stimuli versendet. Ansonsten wird für jeden Informationsaustausch ein eigener Stimulus modelliert.

Bei der Intraprozeßkommunikation werden die eingezeichneten Stimuli im ProSet-Programm durch Prozeduraufrufe oder return-Befehle simuliert (hierauf wurde bereits bei der Begründung der Einführung des Objektes BA_Start eingegangen). Die Rückgabe von Informationen als direkte Reaktion auf einen Stimulus wird in ProSet immer durch einen return-Befehl simuliert.

Wenn in einem Entwurfsmodellobjekt Aktionen ausgeführt werden sollen oder wenn auf eine direkte Reaktion eines Objektes, das ein Stimulus empfangen hat, gewartet wird, wird der entsprechende Balken weiß gezeichntet, ansonsten ist er schwarz. Ein zusätzlicher Balken wird für die *Systemgrenze* gezeichnet. Die Systemgrenze repräsentiert alles, was außerhalb des modellierten Software-Systems liegt, wie z.B. die Ein-/Ausgabe-Operationen des Betriebssystems.

Die Aktionen, die in den Blöcken ausgeführt werden, sind am linken Rand beschrieben. Die Beschreibung besteht aus strukturiertem Text. Alternativ hätte hier auch bereits ein Pseudo-Code der Programmiersprache, in der das System implementiert wird, verwendet werden können. Der Pseudo-Programmiersprachen-Code wird bei dem betrachteten Fallbeispiel nicht verwendet, weil die entworfenen Interaktionsdiagramme sowohl als Grundlage für den ProSet-Prototyp als auch als Grundlage für die Implementierung eines Systems, mit dem die Endanwender arbeiten, dienen

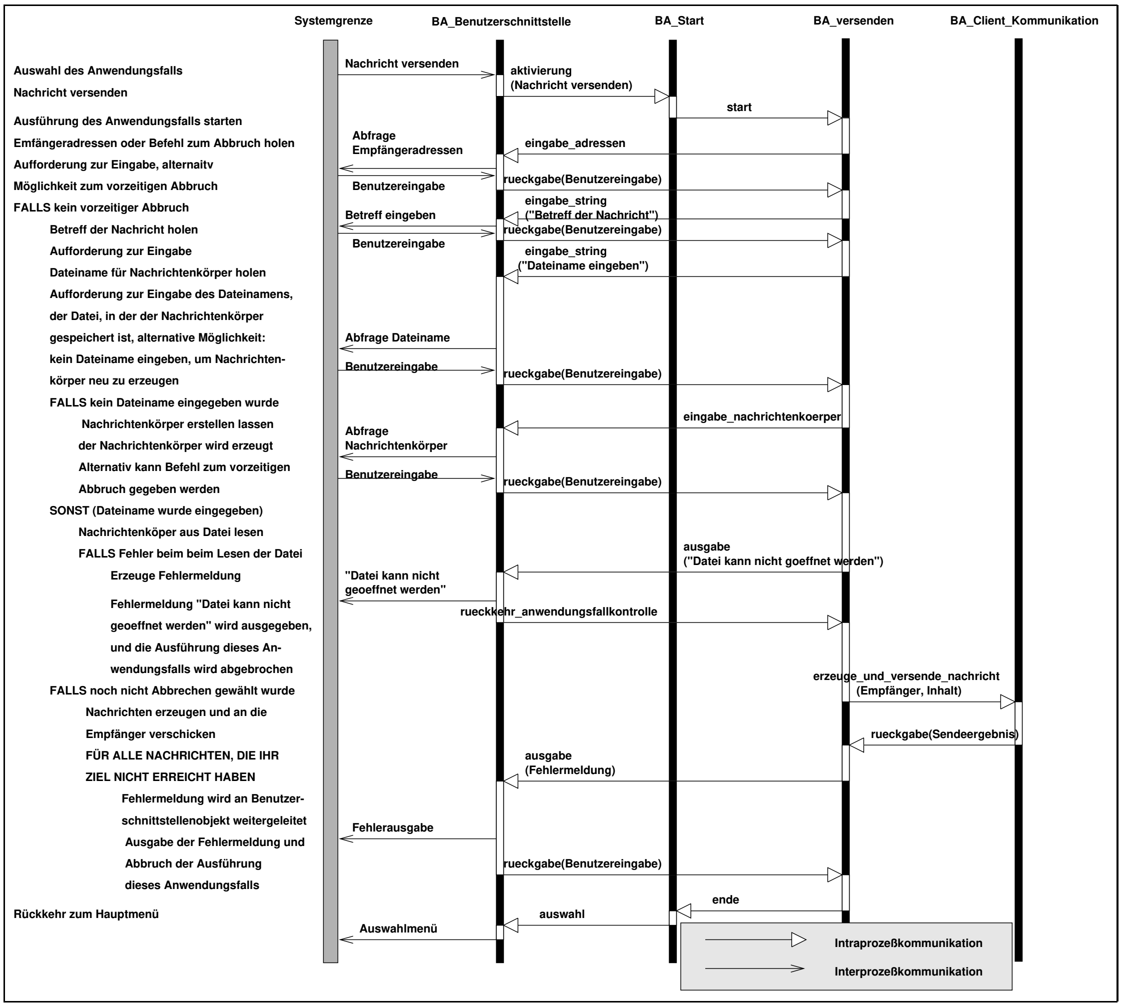

Abbildung 4.18: Interaktionsdiagramm für die Anwendungsfälle Nachricht versenden und Nachricht erzeugen.

können sollen. Bei der Programmiersprache, in der ein System für die Anwender implementiert wird, wird es sich in der Regel nicht um die Sprache ProSet handeln, die für die Erstellung des Prototyps verwendet wird.

Bei der ProSet-Implementierung des hier betrachteten Prototyps wird für jede Klasse eines Entwurfsmodellobjektes eine Modulinstanz erstellt. Die Entwurfsmodellobjekte werden in der ProSet-Implementierung durch Instanzen dieser Moduln dargestellt. Die Stimuli, die zwischen den Objekten des modellierten Systems ausgetauscht werden, entsprechen immer Prozeduraufrufen oder return-Befehlen (siehe hierzu Begründung der Einführung des Objektes BA_Start).

Es ist daher einfach möglich, auf der Basis der Interaktionsdiagramme, den ProSet-Prototyp zu implementieren. Bei der Implementierung von Prototypen werden Fehler im Interaktionsdiagramm deutlich. Bei diesen Fehlern kann es sich sowohl um fehlende oder falsche Stimuli als auch um Aktionen handeln, die im Interaktionsdiagramm nicht oder falsch beschrieben wurden.

Die Kommunikation mit dem Transferagenten wird beim vollständigen Prototyp des E-Mail-Systems durch eine Instanz des Objektes BA_Client_Kommunikation durchgeführt. Die Kommunikation eines Benutzeragenten mit einem Transferagenten wird jedoch erst in einer späteren Phase der Entwurfsmodellerstellung und des Prototyping detailliert betrachtet. Zunächst wird die Kommunikation mit dem Transferagenten bei dem erstellten Prototyp durch das Objekt

BA_Client_Kommunikation nur simuliert.

4.6.2 Modellierung des Transferagenten

Bei der Modellierung des Transferagenten wird die gleiche Vorgehensweise angewendet wie bei der Modellierung des Benutzeragenten (siehe Abschnitt 4.6.1). Im einzelnen werden die folgenden Phasen durchlaufen:

Erster Schritt: Zuerst werden die Blöcke, aus denen der Transferagent besteht, sowie ihre Beziehungen modelliert.

Zweiter Schritt: Für die Anwendungsfälle werden Interaktionsdiagramme erstellt.

Dritter Schritt: Der Prototyp des Transferagenten wird implementiert.

Vierter Schritt: Der Entwurf der Struktur des Transferagenten und die Interaktionsdiagramme werden unter Berücksichtigung der Ergebnisse des Prototyping überarbeitet.

Der Aufbau eines Transferagenten ist in der Abbildung 4.19 dargestellt. Im Gegensatz zum Benutzeragenten haben sich durch das Prototyping keine Änderungen für den Aufbau des Transferagenten ergeben. Der skizzierte Aufbau entspricht also sowohl dem Entwurf des Transferagenten vor der Implementierung des Prototyps als auch dem Entwurf des Transferagenten nach dem Prototyping.

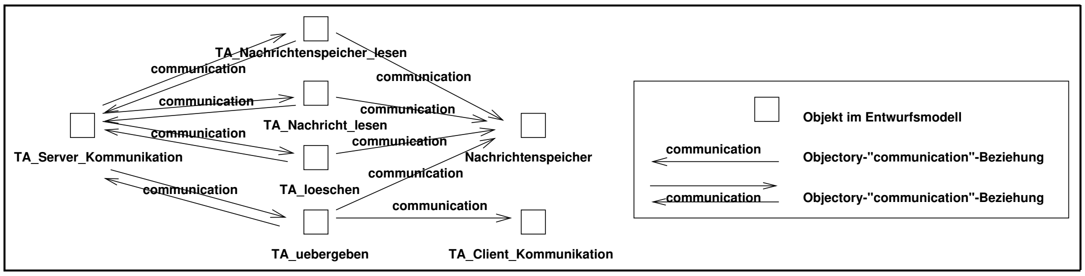

Abbildung 4.19: Der Transferagent im Entwurfsmodell.

Im folgenden werden zunächst die Abweichungen des hier modellierten Transferagenten gegenüber dem im Analysemodell dargestellten Transferagenten beschrieben.

- Analog zur Benennung der Objekte des Benutzeragenten werden die Umlaute ä, ö und ü in den Objektnamen durch ae, oe und ue ersetzt (Beispiel: Objekt TA_loeschen).
- Nach der Modellierung im Analysemodell besitzt ein Transferagent für jeden Benutzeragenten, der ihm zugeordnet ist, jeweils ein Objekt Nachrichtenspeicher (siehe Abbildung 4.9 im Abschnitt 4.5). Ein Transferagent, dem N Benutzeragenten zugeordnet sind, besitzt also gemäß dem Analysemodell auch N Objekte Nachrichtenspeicher (auf die Zuordnung von Benutzeragenten zu Transferagenten wird im Abschnitt 4.6.3 genauer eingegangen). Im Entwurfsmodell besitzt jeder Transferagent genau ein Objekt Nachrichtenspeicher. Durch dieses Objekt werden sämtliche Nachrichtenspeicher des entsprechenden Transferagenten repräsentiert. Diese Änderung gegenüber dem Analysemodell wurde vorgenommen, um die Implementierung zu vereinfachen. Eine Verwaltung einer eigenständigen Instanz des Moduls Nachrichtenspeicher für jeden zugeordneten Benutzeragenten wäre problematisch, weil der Aufbau des zu simulierenden Netzwerks nicht starr vorgegeben ist (siehe Abschnitt 4.3). Zum Zeitpunkt der Übersetzung des Transferagentenprogramms sind weder die benötigte Anzahl der Nachrichtenspeicher noch die Adressen der Benutzer, für die Nachrichtenspeicher eingerichtet werden müssen, bekannt.

Wenn für jeden Nachrichtenspeicher eine eigenständige Modulinstanz erzeugt werden müßte, wäre es nicht möglich, für jede benötigte Modulinstanz eine eigene Variable im Transferagentenprogramm vorzusehen, da bei der Implementierung noch nicht bekannt ist, wieviele Instanzen dieses Moduls Nachrichtenspeicher — und wieviele verschiedene Variablen zur Speicherung der Modulinstanzen — benötigt werden.

Es wäre zwar prinzipiell möglich, im Transferagentenprogramm genau eine Variable zu definieren, die eine endliche Abbildung enthält. Hier könnten zur Laufzeit des Programms bei Bedarf neu erzeugte Instanzen des Moduls Nachrichtenspeicher eingefügt und eindeutig einem bestimmten Benutzer zugeordnet werden. Jedoch erscheint diese Form der Implementierung umständlicher als die Verwaltung genau einer Modulinstanz, in der alle Nachrichtenspeicher eines Transferagenten verwaltet werden.

Da das Entwurfsmodellobjekt Nachrichtenspeicher analog zum Analysemodellobjekt Nachrichtenspeicher zur Speicherung von Daten dient, werden keine „communication“-Beziehungen modelliert, die beim Entwurfsmodellobjekt Nachrichtenspeicher beginnen und bei einem der Objekte zur Kontrolle der Anwendungsfälle enden (zur Beschreibung der Modellierung von „communication“-Beziehungen im Entwurfsmodell siehe Abschnitt 4.6.1).

- Bei den Materialien, die zusätzlich zum Nachrichtenspeicher zur Modellierung des Transferagenten benötigt werden, handelt es sich um dieselben Materialien, die auch durch den Benutzeragenten bearbeitet werden (z.B. Nachrichten und Quittungen). Weil Objekte (bzw. Modulinstanzen) nicht direkt versendet werden können und eine Umwandlung von Modulinstanzen in Zeichenketten nur schwer möglich ist, werden die restlichen benötigten Materialien analog zum Entwurf des Benutzeragenten nicht als Objekte im Entwurfsmodell dargestellt (siehe hierzu Beschreibung der Materialien für den Benutzeragenten, Abschnitt 4.6.1, Seite 83).

  Das im Analysemodell aufgeführte Entitätsobjekt Verbindungsinformationen (siehe Abbildung 4.9 im Abschnitt 4.5) wird bei der hier betrachteten Phase der Erstellung des Entwurfsmodells noch nicht berücksichtigt.

Vor der Implementierung des Prototyps werden noch die Interaktionsdiagramme für die Anwendungsfälle des Transferagenten entworfen. Die Interaktionsdiagramme bilden eine wesentliche Grundlage für die Implementierung des Prototyps. Nach dem Prototyping werden Fehler in den Interaktionsdiagrammen (z.B. fehlende oder falsche Stimuli) beseitigt. Die Interaktionsdiagramme zum Transferagenten sind im Zusatzdokument zur Diplomarbeit aufgeführt Ein Unterschied zwischen dem modellierten Benutzeragenten und dem modellierten Transferagenten soll noch explizit hervorgehoben werden:

Ein Benutzeragent, der bei dem hier modellierten E-Mail-System eine Nachricht an einen Transferagenten übergibt, ist solange blockiert, bis er eine positive oder negative Quittung erhält, die angibt, ob die verschickte Nachricht beim Empfänger angekomment ist oder bis eine maximale Wartezeit abgelaufen ist (Timeout). So kann der Benutzer sofort kontrollieren, ob eine von ihm verschickte Nachricht ihr Ziel erreicht hat.

Der Transferagent wartet hingegen nach dem Weitergeben einer Nachricht nicht auf eine Quittung, die anzeigt, ob eine Nachricht beim Nachrichtenspeicher des Empfängers angekommen ist, um die Abarbeitung des Empfangspuffers nicht unnötig zu verzögern. Quittungen werden vom Transferagenten wie Nachrichten durch das Objekt TA_Server_Kommunikation empfangen. Wenn der Benutzeragent des Quittungsempfängers dem Transferagenten zugeordnet ist, wird die Quittung über das Objekt TA_Server_Kommunikation an den wartenden Benutzeragenten übergeben. Ansonsten wird die Quittung über das Objekt TA_Client_Kommunikation an den nächsten Transferagenten weitergeleitet.

Bei der Implementierung des ProSet-Prototyps für den Transferagenten wird analog zur Implementierung des Prototyps für den Benutzeragenten jedes Entwurfsmodellobjekt durch eine ProSet-Modulinstanz repräsentiert. Das Versenden der in den Interaktionsdiagrammen dargestellten Stimuli wird durch Aufrufe von Prozeduren oder durch return-Befehle simuliert. Direkte Reaktionen auf Stimuli werden immer durch return-Befehle simuliert.

Die Moduln TA_Server_Kommunikation und TA_Client_Kommunikation des ProSet-Prototyps haben bei dem hier betrachteten Prototyp noch nicht ihre endgültige Form. In der hier betrachteten Phase des Prototyping simulieren sie nur die Moduln, die bei der Erstellung des Protoyps für das vollständige E-Mail-System (siehe Abschnitt 4.6.3) benötigt werden. Benutzeragenten bzw. fremde Transferagenten, von denen Nachrichten oder Quittungen empfangen werden, werden in der hier betrachteten Prototyping-Phase durch den Entwickler, der mit dem erstellten Prototyp arbeitet, simuliert. Das Objekt TA_Server_Kommunikation, das in einer späteren Entwicklungsphase Nachrichten, Quittungen oder Befehle zur Verwaltung der Nachrichtenspeicher von fremden Transferagenten oder von Benutzeragenten empfängt, liest bei dem hier erstellten Prototyp die Nachrichten, Quittungen oder Befehle zur Nachrichtenspeicherverwaltung, die der Entwickler über die Tastatur eingibt. Des weiteren kann noch nicht flexibel gewählt werden, für welche Benutzer Nachrichtenspeicher verwaltet werden sollen. Die Namen der Benutzer, für die durch die Instanz des Moduls Nachrichtenspeicher ein Nachrichtenspeicher eingerichtet wird, sind noch fest vorgegeben.

Bei der Implementierung des Prototyps böte es sich an, die Nachrichten, die im Objekt Nachrichtenspeicher für die verschiedenen Benutzer verwaltet werden, in persistenten Variablen zu speichern. Dadurch blieben die Nachrichtenspeicherinhalte auch dann erhalten, wenn ein Transferagent vorübergehend ausfällt. Das Erhalten der Nachrichtenspeicherinhalte bei einem Ausfall eines Transferagenten ist für ein System, mit dem die Anwender arbeiten sollen, unbedingt notwendig. Bei der Implementierung des Prototyps wurde auf die Nutzung des Persistenzmechanismus verzichtet. Auf die Gründe wird im folgenden eingegangen:

Die Namen der Variablen für die Datenstrukturen, in denen Nachrichten gespeichert werden sollen, hätten bereits vor der Übersetzung des ProSet-Transferagentenprogramms festgelegt werden müssen, weil Variablennamen in ProSet nicht zur Programmlaufzeit definiert oder geändert werden können. Solange nur mit einem Transferagenten gearbeitet wird, könnte eine Datenstruktur definiert werden, in der die zu speichernden Nachrichten nach Nachrichtenspeichern getrennt abgelegt werden können. Eine solche Datenstruktur könnte relativ einfach mit Hilfe der von ProSet angebotenen zusammengesetzten Datentypen Tuple und Set — und mit daraus erzeugten endlichen Abbildungen (*maps*) — definiert werden. Für diese Datenstruktur könnte eine persistente Variable erzeugt werden.

Das E-Mail-System, das durch den zu erstellenden Prototypen simuliert werden soll, soll jedoch mehrere Transferagenten besitzen können. Die verschiedenen Transferagenten werden bei der endgültigen Prototypimplementierung durch verschiedene Instanzen desselben Programms repräsentiert. Es wäre daher sehr einfach möglich, genau eine eine persistente Variable zu erzeugen, in der die Inhalte sämtlicher Nachrichtenspeicher aller simulierten Transferagenten abgelegt werden. Sämtliche Instanzen des Transferagentenprogramms griffen dann auf dieselbe persistente Variable zu. Hier träten jedoch die folgenden Probleme auf:

- Solange ein Transferagent die persistente Variable, die alle Nachrichtenspeicher enthält, manipuliert, kann kein weiterer Transferagent mehr auf seinen Nachrichtenspeicher zugreifen, weil dies durch eine ProSet-Schreibsperre (siehe Doberkat et al. [DFH+95, S. 59ff.]) verhindert wird.
- Die Erfahrungen mit dem ersten Fallbeispiel haben gezeigt, daß bei einer Nutzung des Persistenzmechanismus Deadlocks auftreten können, wenn mehrere Programminstanzen auf dieselbe persistente Variable zugreifen (siehe Abschnitt 3.6). Da die Ausnahmebehandlung bei der verwendeten ProSet-Compiler-Version 0.6 nicht innerhalb von Prozeduren funktioniert, wäre es nur sehr umständlich möglich gewesen, einen unerwünschten Programmabruch beim Auftreten eines Deadlocks zu vermeiden.

Alternativ könnten verschiedene persistente Variablen für die Nachrichtenspeicher der verschiedenen Transferagenten definiert werden. Jeder simulierte Transferagent besäße dann für seine Nachrichtenspeicher eine eigene persistente Variable. Hierzu sollen zwei verschiedene Varianten betrachet werden:

- Die persistenten Variablen verschiedener Transferagenten werden im selben P-File gespeichert.

- Die persistenten Variablen verschiedener Transferagenten werden in verschiedenen P-Files gespeichert.

Verschiedene Variablen im selben P-File: Für jeden zu simulierenden Transferagenten wird eine Instanz desselben Transferagentenprogramms gestartet. Damit jede gestartete Instanz eines Transferagentenprogramms eine eigene persistente Variable für seine Nachrichtenspeicher besitzen kann, müßten diese Variablen für die verschiedenen Instanzen unterschiedliche Namen haben. Diese Namen müßten bereits bei der Implementierung des Transferagentenprogramms festgelegt werden, weil Variablennamen im ProSet-Programm nicht dynamisch zur Laufzeit eines Programms definiert oder geändert werden können. Zum Zeitpunkt der Implementierung des Transferagentenprogramms ist jedoch noch nicht bekannt, wieviele verschiedene Instanzen dieses Programms zur Simulation eines Netzwerks gestartet werden müssen. Deshalb ist auch nicht bekannt, wieviele verschiedene persistente Variablen zur Speicherung der Nachrichtenspeicherinhalte im Transferagentenprogramm berücksichtigt werden müssen.

Verschiedene Variablen in verschiedenen P-Files: Hier könnten die Namen der verschiedenen persistenten Variablen zur Aufnahme der Nachrichtenspeicherinhalte der verschiedenen Transferagenten gleich sein. Der Name des P-Files, das einer Instanz des Transferagentenprogramms zugeordnet ist, könnte beim Start dieser Instanz als Argument übergeben werden. Gegen eine Realisierung dieser Variante sprechen die folgenden Argumente:

- Die Arbeit mit dem Prototyp wird komplizierter. Weil zum Anlegen eines P-Files externe Werkzeuge, wie z.B. der xpft, nötig sind, ist kann eine Erweiterung der zu simulierenden Netzwerkstruktur nicht mehr durch den erstellten Prototyp allein durchgeführt werden. Es muß immer auf externe Werkzeuge zurückgegriffen werden.
- Auf einen Nachrichtenspeicher müßten die simulierten Transferagenten auch schreibend zugreifen. Hierfür müßten in den entsprechenden Prozeduren in den Moduln des ProSet-Programms persistente Variablen definiert werden. Dies führte bei der Übersetzung des Transferagentenprogramms zu der Fehlermeldung internal assembler error: out of memory, die bereits bei der Betrachtung des ersten Fallbeispiels beschrieben wurde (siehe Abschnitt 3.5.1). Es wäre zwar möglich, diese Fehlermeldung dadurch zu umgehen, daß Moduln getrennt voneinander übersetzt, persistent gemacht und anschließend vom persistenten Speicher geladen werden, wie dies bereits beim ersten Fallbeispiel gezeigt wurde. Die Erstellung des Prototyps und insbesondere das nachträgliche Verbessern von Fehlern in den Moduln, die persistent gemacht werden müßten, würde dadurch jedoch erschwert.

  Diese Schwierigkeit träte auch bei allen anderen bereits vorgestellten Realisierungsideen für persistente Nachrichtenspeicher im Prototyp zusätzlich zu den dort bereits beschriebenen Problemen auf.

Um zu entscheiden, ob es sinnvoll ist, die Nachrichtenspeicher trotz der beschriebenen Schwierigkeiten persistent zu machen, sollen noch einmal das Ziele betrachtet werden, die hier mit dem Prototyping verfolgt werden: Durch den erstellten Prototyp sollen die Interaktionen zwischen Entwurfsmodellobjekten simuliert werden können. Hierdurch soll eine Validierung erstellter Teile des Entwurfsmodells ermöglicht und Aufschlüsse für nötige Änderungen und mögliche Weiterentwicklungen des Entwurfsmodells gegeben werden (siehe Seite 78).

Das Erreichen dieser Ziele wird nicht behindert, wenn die Nachrichtenspeicher beim erstellten Prototyp nicht persistent sind. Deshalb werden die Nachrichtenspeicher nur in internen Variablen der Objekte Nachrichtenspeicher, die zu den gestarteten Transferagenten gehören verwaltet. Sie werden wegen der beschriebenen Schwierigkeiten nicht persistent gemacht.

4.6.3 Modellierung des vollständigen E-Mail-Systems

Im letzten Schritt werden die in den Abschnitten 4.6.1 und 4.6.2 beschriebenen Modelle zusammengefaßt und ergänzt, um ein vollständiges E-Mail-System zu modellieren. Dabei wird folgendermaßen vorgegangen:

- Zunächst werden die Objekte, die für die Benutzer- und die Transferagenten benötigt werden, sowie ihre Beziehungen zueinander beschrieben. Dabei wird auf die bereits in den Abschnitten 4.6.1 und 4.6.2 vorgestellten Modelle zurückgegriffen.
- Anschließend wird der Datenaustausch zwischen den Benutzeragenten und den Transferagenten bzw. zwischen verschiedenen Transferagenten für den Prototyp beschrieben.
- Im nächsten Schritt wird der Prototyp implementiert.
- Nach der Implementierung des Prototyps wird die Arbeit an diesem Fallbeispiel mit einem Ausblick auf auf weitere Arbeiten, die zu Erstellung eines vollständigen Entwurfsmodells für das E-Mail-System nötig sind, beendet.

Die Ergebnisse dieser Phasen werden nun im einzelnen betrachtet.

Objekte und Beziehungen im E-Mail-System

Bei der Darstellung der Objekte und der Beziehungen zwischen Objekten des vollständigen E-Mail-Systems werden die in den Abbildungen 4.15 (Abschnitt 4.6.1) und 4.19 (Abschnitt 4.6.2) dargestellten Objekte und Beziehungen übernommen. Zusätzlich wird der Transferagent um das Objekt Verbindungsinformationen erweitert. Das Objekt Verbindungsinformationen enthält die Routing-Tabelle für den zugehörigen Transferagenten (siehe Abschnitt 4.5). Auf die Erstellung der Routing-Tabellen wird bei der Beschreibung der Implementierung des Prototyps näher eingegangen. Wenn ein Transferagent eine Nachricht oder eine Quittung an einen anderen Transferagenten weiterleiten muß, liefert das Objekt Verbindungsinformationen auf Anfrage die Adresse des nächsten Transferagenten. Die Objekte des vollständigen E-Mail-Systems und ihre Beziehungen zueinander sind in den Abbildungen 4.20 und 4.21 dargestellt. Der Benutzeragent kommuniziert mit einem Transferagenten grundsätzlich über das Objekt BA_Client_Kommunikation. Ein Transferagent kommuniziert mit den Benutzeragenten, die ihm zugeordnet sind, grundsätzlich über das Objekt TA_Server_Kommunikation. Des weiteren empfängt ein Transferagent mit Hilfe des Objektes TA_Server_Kommunikation Nachrichten und Quittungen, die ihm von fremden Transferagenten zugesendet werden. Wenn Nachrichten oder Quittungen an andere Transferagenten weitergeleitet werden müssen, geschieht dies mit Hilfe des Objektes TA_Client_Kommunikation.

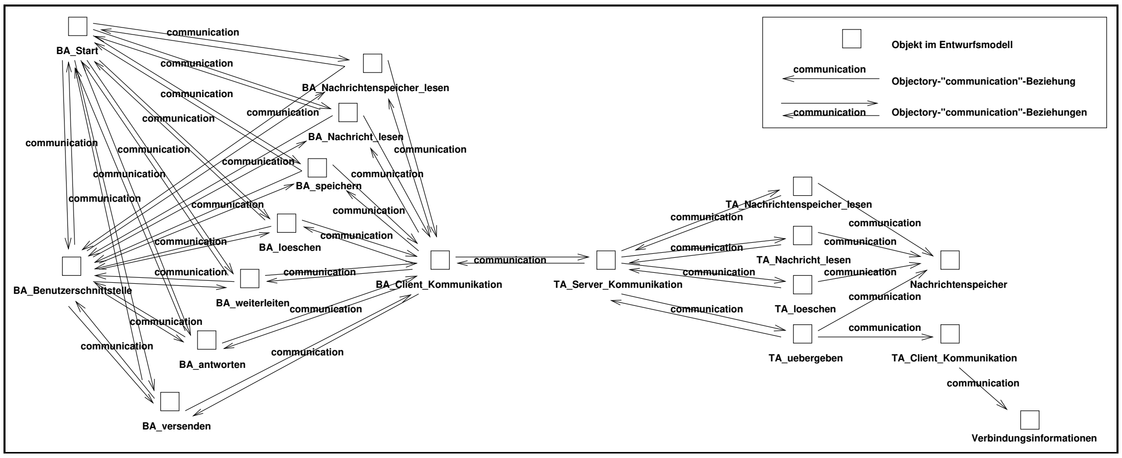

Abbildung 4.20: Benutzeragent und Transferagent.

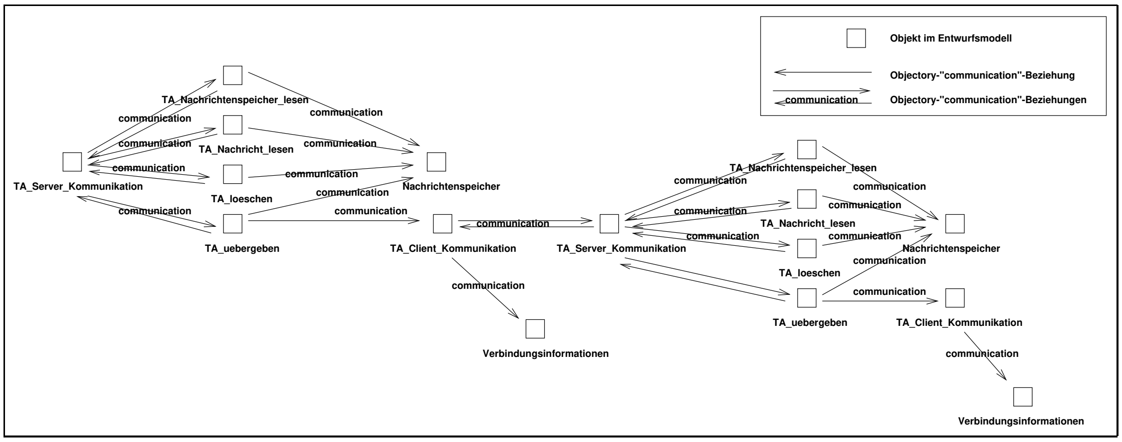

Abbildung 4.21: Zwei Transferagenten.

Beschreibung des Datenaustauschs beim Prototyp

Bei den durch den Prototyp simulierten Benutzeragenten und den Transferagenten handelt es sich immer um eigenständige Prozesse. Deshalb ist zur Simulation des Informationsaustauschs zwischen den Netzwerkkomponenten Interprozeßkommunikation erforderlich.

Da das Prototyping mit persistenten Daten ein Schwerpunkt dieser Diplomarbeit ist, erscheint es zunächst naheliegend, die Interprozeßkommunikation mit Hilfe des ProSet-Persistenzmechanismus zu realisieren. Dabei wäre es möglich, eine persistente Variable zu erzeugen, auf die sämtliche Benutzeragentenprozesse und Transferagentenprozesse zugreifen können. Diese Variable könnte genutzt werden, um Informationen zwischen verschiedenen Prozessen auszutauschen.

Alternativ könnte für jeden Benutzeragentenprozeß und für jeden Transferagentenprozeß eine eigene persistente Variable erzeugt werden, die als Empfangspuffer für eintreffende Informationen von fremden Prozessen dient. Die Realisierung der Interprozeßkommunikation mit Hilfe des Persistenzmechanismus hätte jedoch zu zu den gleichen Problemen geführt, die bereits bei der Diskussion über die Probleme, die Nachrichtenspeicher des Prototyps persistent zu machen, aufgeführt wurden (siehe Abschnitt 4.6.2, Seite 88f.). Es erscheint für daher einfacher, für die Interprozeßkommunikation C-Funktionen zu implementieren, die in ein ProSet-Programm eingebunden werden und von dort aus aufgerufen werden können.

Bei der Kommunikation zwischen den Komponenten des E-Mail-Systems kann zwischen der Kommunikation zwischen einem Benutzeragenten und einem Transferagenten und der Kommunikation zwischen zwei Transferagenten unterschieden werden. Der hier vorgestellte Ablauf des Datenaustauschs bezieht sich ausschließlich auf den Prototyp des E-Mail-Systems. Für das System, mit dem die Endanwender arbeiten, wäre ein umfangreicheres Protokoll nötig, das einen größeren Schutz vor Fehlern bei der Datenübertragung bietet.

Kommunikation zwischen Benutzer- und Transferagent: Ein Benutzeragent kann Aufträge an einen Transferagenten senden. Nach jedem vergebenen Auftrag wartet der Benutzeragent auf die Bearbeitung des Auftrags durch den Transferagenten. Wenn innerhalb einer vorgegeben maximalen Wartezeit keine Rückmeldung vom Transferagenten empfangen wird, die die erfolgreiche Bearbeitung des Auftrags signalisiert, bricht der Benutzeragent das Warten ab.

Es kann zwischen Aufträgen, die sich auf die Verwaltung des Nachrichtenspeichers eines Benutzers beziehen, und Aufträgen, bei denen Nachrichten versendet werden sollen, unterschieden werden.

Aufträge zur Verwaltung eines Nachrichtenspeichers: Die folgenden Aufträge sind möglich:

- Inhaltsverzeichnis eines Nachrichtenspeichers anzeigen,
- Nachricht lesen und
- Nachricht löschen.

Die Struktur der Befehle, die an den Transferagenten verschickt werden, ist bei diesen drei Aufträgen immer gleich. Ein Befehl besteht aus vier Feldern, in denen sich Informationen befinden, die der Transferagent zur Ausführung des übergebenen Auftrages benötigt. Diese Felder sind in der Abbildung 4.22 dargestellt.

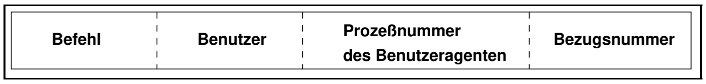

Abbildung 4.22: Aufbau eines Befehls zur Verwaltung eines Nachrichtenspeichers.

Die einzelnen Felder eines Befehls werden im folgenden beschrieben:

Befehl: Hier wird festgelegt welchen Auftrag der Transferagent ausführen soll. Die folgenden Einträge sind zulässig:

- "Nachrichtenspeicherinhalt holen",
- "Nachricht aus Nachrichtenspeicher holen",
- "Nachricht aus Nachrichtenspeicher entfernen".

Benutzer: Hier wird die Adresse des Benutzers angegeben, auf dessen Nachrichtenspeicher zugegriffen werden soll.

Prozeßnummer des Benutzeragenten: Jedem Benutzeragentenprozeß und jedem Transferagentenprozeß ist eine Nummer eindeutig zugeordnet, durch die der entsprechende Prozeß identifiziert werden kann. Diese Nummer wird als *Prozeßnummer* bezeichnet. Das Feld Prozeßnummer des Benutzeragenten enthält die Prozeßnummer des Benutzeragenten, der den Auftrag an einen Transferagenten übergeben hat.

Bezugsnummer: In einem Nachrichtenspeicher können sich mehrere Nachrichten befinden. Damit es möglich ist, sich auf spezielle Nachrichten in einem Nachrichtenspeicher zu beziehen, wird jeder Nachricht in einem Nachrichtenspeicher eine Zahl eindeutig zugeordnet. Diese Zahl wird als *Bezugsnummer* bezeichnet. Durch die Bezugsnummer kann sich der Benutzeragent bei den Aufträgen Nachricht aus Nachrichtenspeicher holen und Nachricht aus Nachrichtenspeicher entfernen eindeutig auf eine bestimmte Nachricht beziehen. Wenn das Inhaltsverzeichnis eines Nachrichtenspeichers ausgegeben werden soll, enthält dieses Feld den Zahlenwert Null. Um die Struktur der Befehle, die sich auf die Verwaltung des Nachrichtenspeichers beziehen, zu vereinheitlichen, wird dieses Feld auch dann an den Transferagenten übergeben, wenn das Inhaltsverzeichnis eines Nachrichtenspeichers an den Benutzeragenten zurückgesendet werden soll.

Versenden von Nachrichten: Wenn eine Nachricht versendet werden soll, übergibt der Benutzeragent die zu versendende Nachricht an den Transferagenten. Die Felder einer Nachricht sind in der Abbildung 4.23 dargestellt.

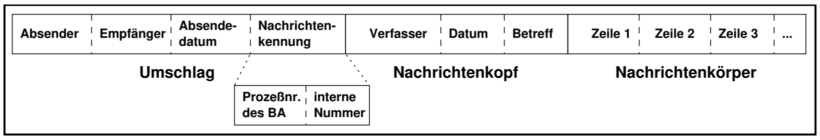

Abbildung 4.23: Aufbau einer Nachricht.

Eine Nachricht enthält die Informationen, die bereits bei der Beschreibung der Materialien im Analysemodell für eine Nachricht vorgesehen waren (siehe Abbildung 4.9 im Abschnitt 4.5). Bestandteile einer Nachricht sind der Umschlag, der Nachrichtenkopf und der Nachrichtenkörper. Der Umschlag enthält die Adressen des Absenders und des Empfängers, sowie das Absendedatum und eine Kennzeichnung für die Nachricht (Feld Nachrichtenkennung). Der Nachrichtenkörper enthält die Adresse des Verfassers, das Datum und den Betreff der Nachricht (Absender und Verfasser einer Nachricht müssen nicht identisch sein, siehe Abschnitt 4.4.2, Beispiel auf Seite 65). Im Nachrichtenkörper sind die Textzeilen der Nachricht gespeichert. Da die Anzahl der Textzeilen bei den Nachrichten variabel ist, kann die die Anzahl der Felder des Nachrichtenkörpers bei verschiedenen Nachrichten variieren.

Die Angabe des Absendedatums auf dem Umschlag wird bei dem modellierten E-Mail-System zwar nicht benötigt. Dieses Feld könnte jedoch bei Erweiterungen des modellierten Systems sinnvoll sein. Durch die Angabe des Absendedatums könnte z.B. kontrolliert werden, ob eine Nachricht vor einer festgelegten Frist abgesendet wurde.

Für die Kennzeichnung einer Nachricht war im Analysemodell eine ganze Zahl vorgesehen. Das Umschlagsfeld Nachrichtenkennung besteht bei der in der Abbildung 4.23 vorgeschlagenen Nachrichtenstruktur im Gegensatz dazu aus zwei Teilen. Im ersten Teil wird die Prozeßnummer des Benutzeragenten, der die Nachricht verschickt gespeichert. Der zweite Teil des Feldes Nachrichtenkennung erlaubt dem Benutzeragenten eine eindeutige interne Kennzeichnung seiner Nachrichten. Die Speicherung der Prozeßnummer ist nötig, damit eine Quittung für eine Nachricht an den richtigen Benutzeragenten zurückgeleitet werden kann. Eine interne Nachrichtenkennzeichnung des Benutzeragenten wäre für den hier modellierten Benutzeragenten nicht nötig. Sie könnte jedoch bei einer Erweiterung des Systems erforderlich sein, wenn mehrere Nachrichten hintereinander verschickt werden sollen, ohne daß der Benutzeragenten nach jedem Senden bis zum Eintreffen einer Quittung blockiert ist. Durch die interne Nachrichtenkennzeichnung könnten Quittungen eindeutig verschiedenen Nachrichten zugeordnet werden.

Wenn kein Fehler aufgetreten ist, haben die Reaktionen des Transferagenten, die an den Benutzeragenten zurückgesendet werden, die in der Abbildung 4.24 dargestellten Formate. Falls ein Fehler aufgetreten ist, sendet der Transferagent eine der in der Abbildung 4.25 dargestellten Fehlermeldungen an den Benutzeragenten.

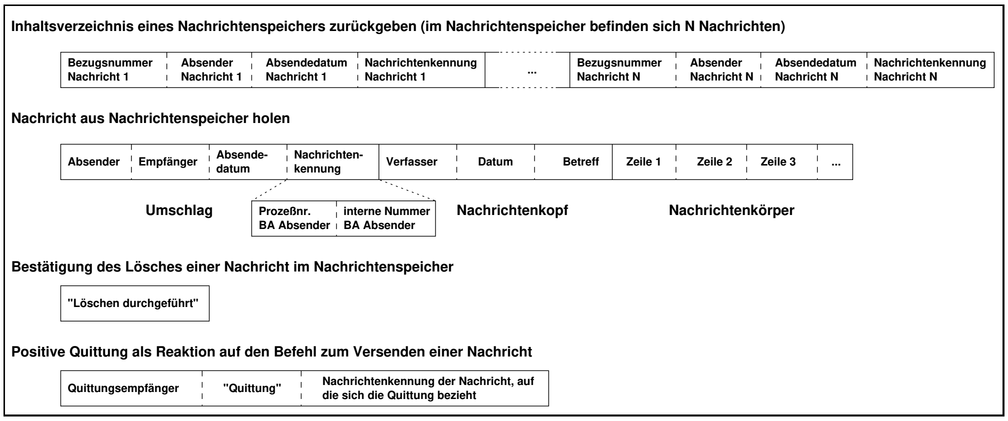

Abbildung 4.24: Antworten des Transferagenten, falls kein Fehler aufgetreten ist.

Kommunikation zwischen zwei Transferagenten: Zwischen zwei Transferagenten werden Nachrichten und Quittungen ausgetauscht. Die Nachrichten bzw. Quittungen haben dieselben Formate wie die Nachrichten bzw. die Quittungen, die zwischen Benutzeragenten

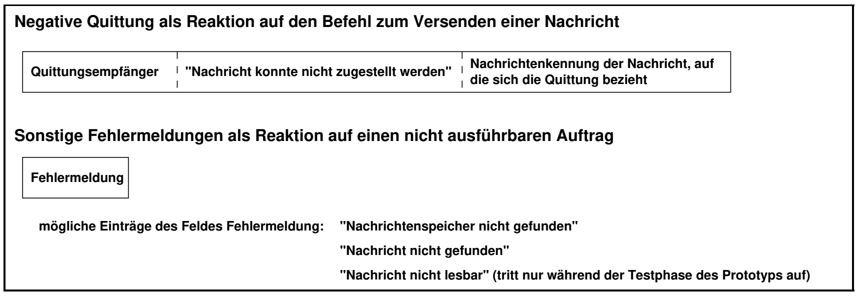

Abbildung 4.25: Antworten des Transferagenten, falls ein Fehler aufgetreten ist.

und Transferagenten ausgetauscht werden (siehe Abbildungen 4.23, 4.24 und 4.25). Wenn ein Transferagent eine Nachricht oder eine Quittung an einen anderen Transferagent gesendet hat, sendet der empfangende Transferagent eine *Empfangsbestätigung* an den sendenden Transferagenten zurück.

Diese Empfangsbestätigung hat das in der Abbildung 4.26 dargestellte Format. Sie unterscheidet sich von den bisher betrachteten Quittungen. Die Bezeichnung Quittung wird in dieser Arbeit für Informationen verwendet, die angeben, ob eine Nachricht bis zum Nachrichtenspeicher des vorgesehenen Empfängers weitergeleitet werden konnte. Die hier betrachtete Empfangsbestätigung bezieht sich im Gegensatz dazu nur auf den Informationsaustausch zwischen zwei Transferagenten. Diese Empfangsbestätigung wird von dem Transferagenten, der Informationen empfängt, an den sendenden Transferagenten geschickt.

Für das Warten eines sendenden Transferagenten auf die Reaktion eines empfangenden Transferagenten wird bei der Einrichtung des Systems ein Zeitlimit vorgegeben.

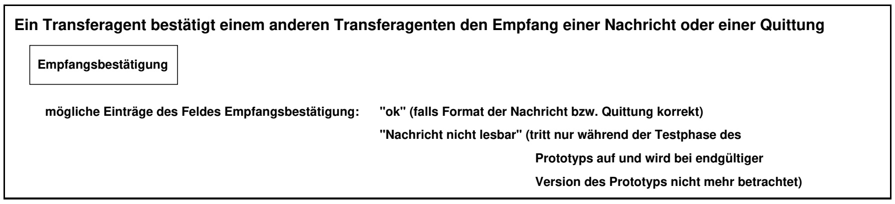

Abbildung 4.26: Format einer Empfangsbestätigung.

Implementierung des Prototyps

Bei der Implementierung des Prototyps kann auf die bereits in den den Abschnitten 4.6.1 und 4.6.2 vorgestellten Prototypen für Benutzer- und Transferagenten zurückgegriffen werden. Die Moduln für die Klassen der Objekte BA_Client_Kommunikation, TA_Client_Kommunikation, TA_Server_Kommunikation und Nachrichtenspeicher (siehe Abbildungen 4.20 und 4.21) müssen dabei vollständig überarbeitet werden. Der Modul für die Klasse des Objektes Verbindungsinformationen, das die Routing-Tabelle verwaltet, muß noch implementiert werden. Zusätzlich müssen Programme erstellt werden, mit deren Hilfe der Entwickler, der im Rahmen des experimentellen Prototyping mit dem erstellten Prototyp arbeitet, ein Netzwerk spezifizieren und aktivieren kann. Auf die angesprochenen Punkte wird in diesem Abschnitt in der folgenden Reihenfolge eingegangen:

- Zuerst wird beschrieben, wie das Netzwerk, das durch den Prototyp simuliert werden soll, aufgebaut wird.
- Anschließend wird auf die Änderungen und Erweiterungen eingegangen, die in den Implementierungen des Benutzeragentenprogramms und des Transferagentenprogramms vorgenommen

werden müssen.

Aufbau des Netzwerks: Damit der Entwickler, der mit dem Prototyp arbeitet, das Netzwerk, das simuliert werden soll, spezifizieren kann, wird das Programm Netzaufbau implementiert. Durch dieses Programm werden die folgenden Informationen in persistenten Variablen abgelegt:

Beschreibung einer Routing-Matrix: Es wird eine Matrix erstellt, in der angegeben wird, an welche Transferagenten Nachrichten oder Quittungen für die verschiedenen Benutzer weitergeleitet werden müssen. Die Namen von Transferagenten und Benutzern des E-Mail-Systems werden durch den Entwickler, der das Programm Netzaufbau benutzt, spezifiziert. Für jeden Transferagenten ist eine Zeile in der erstellten Matrix reserviert, für jeden möglichen Empfänger einer Nachricht ist eine Spalte reserviert. Diese Matrix wird im folgenden als *Routing-Matrix* bezeichnet. Ihr Aufbau soll anhand eines Beispiels verdeutlicht werden:

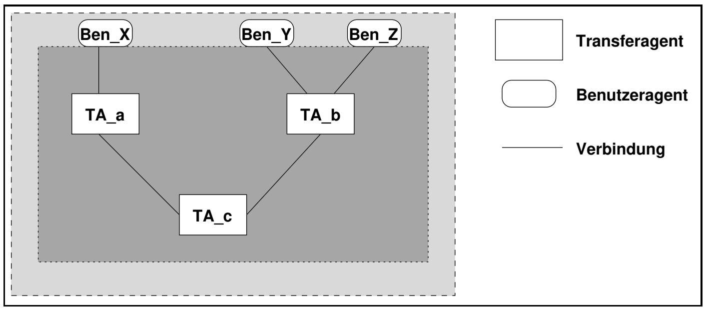

Abbildung 4.27: Beispiel für ein einfaches Netzwerk.

Beispiel: In der Abbildung 4.27 ist das Beispielnetzwerk, das bereits bei der Beschreibung des Analysemodells verwendet wurde, skizziert. Es existieren die drei Transferagenten TA_a, TA_b und TA_c. Der Transferagent TA_a verwaltet den Nachrichtenspeicher des Benutzers Ben_X. Der Transferagent TA_b verwaltet die Nachrichtenspeicher der Benutzer Ben_Y und Ben_Z. Der Transferagent TA_c dient ausschließlich dazu, Nachrichten und Quittungen weiterzuleiten.

Die Routing-Matrix dieses Netzwerks ist in der Tabelle 4.6 dargestellt. Als Beispiel soll die Zeile für den Transferagenten TA_c in dieser Routing-Matrix betrachtet werden. In dieser Zeile wird beschrieben, wohin der Transferagent TA_c Nachrichten und Quittungen weiterleiten muß. Wenn eine Nachricht oder eine Quittung für den Benutzer Ben_X empfangen wird, muß diese Nachricht bzw. Quittung an den Transferagenten TA_a weitergeleitet werden. Wenn eine Nachricht bzw. eine Quittung für den Benutzer Ben_Y oder für den Benutzer Ben_Z empfangen wird, muß die Nachricht bzw. Quittung an den Transferagenten TA_b weitergesendet werden.

	Ben_X	Ben_Y	Ben_Z
TA_a	—	TA_c	TA_c
TA_b	TA_c	—	—
TA_c	TA_a	TA_b	TA_b

Tabelle 4.6: Routing-Matrix für das Beispielnetzwerk aus Abbildung 4.27.

Mit Hilfe der in der Routing-Matrix gespeicherten Daten werden die Routing-Tabellen beim Start der verschiedenen Transferagenten erzeugt. Da ProSet keinen eigenen Datentyp zur Darstellung einer Matrix anbietet, wird die Routing-Matrix im ProSet-Programm durch geschachtelte endliche Abbildungen dargestellt. Die Repräsentation der in der Tabelle 4.6 dargestellten Routing-Matrix mit Hilfe endlicher Abbildungen ist in der Tabelle 4.7 skizziert.

Transferagent	Informationen zur Weiterleitung	
TA_a	Empfänger	weiterleiten an
	Ben_Y	TA_c
	Ben_Z	TA_c
TA_b	Empfänger	weiterleiten an
	Ben_X	TA_c
TA_c	Empfänger	weiterleiten an
	Ben_X	TA_a
	Ben_Y	TA_b
	Ben_Z	TA_b

Tabelle 4.7: Darstellung der Routing-Matrix aus Tabelle 4.6 durch endliche Abbildungen.

Die Einträge in der Routing-Matrix müssen durch den Software-Entwickler, der mit dem hier erstellten Prototyp experimentiert, selbst festgelegt werden. Es liegt in der Verantwortung des Entwicklers, Netzwerke zu spezifizieren, in denen alle Nachrichten ihre Empfänger erreichen können. Prinzipiell wäre es möglich, durch den Prototyp Netzwerke zu simulieren, in denen Nachrichten zyklisch zwischen verschiedenen Transferagenten weitergeleitet werden, ohne daß die Nachrichten ihr Ziel erreichen.

Das Übertragen der Aufgabe, ein Netzwerk zu spezifizieren, in denen die versendeten Nachrichten ihr Ziel erreichen können, an den Benutzer, der den Prototyp konfiguriert, wird als zulässig betrachtet, weil der zu erstellende Prototyp noch kein System darstellt, das bereits für die Benutzung durch Endanwender konzipiert ist, wie es z.B. bei einem Pilotsystem der Fall wäre (siehe Abschnitt 2.3.1). Mit dem erstellten Prototyp arbeiten die Software-Entwickler selbst, um die Interaktionen zwischen den Komponenten eines Software-Systems, das in der Praxis eingesetzt werden soll, zu simulieren (siehe Seite 78).

Zuordnung der Nachrichtenspeicher: Es wird eine persistente Tabelle angelegt, in der jedem Transferagenten eine Menge mit den Namen der Benutzer zugeordnet ist, für die Nachrichtenspeicher verwaltet werden. Die persistente Tabelle für das in der Abbildung 4.27 skizzierte Netzwerk ist in der Tabelle 4.8 skizziert. Diese Zuordnung der Nachrichtenspeicher zu den Transferagenten könnte auch aus der Routing-Matrix entnommen werden. Die zusätzliche Tabelle dient dazu, die Implementierung zu vereinfachen. So muß z.B. ein neu gestarteter Transferagent erfragen, für welche Benutzer er Nachrichtenspeicher einrichten muß. Diese Informationen können direkt aus der hier beschriebenen Tabelle entnommen werden.

Transferagent	Nachrichtenspeicher für Benutzer
TA_a	{Ben_X}
TA_b	{Ben_Y, Ben_Z}
TA_c	{ }

Tabelle 4.8: Transferagenten und Nachrichtenspeicher des Beispielnetzwerks aus Abbildung 4.27.

Speicherung eines Zeitlimits: Hier wird festgelegt, wie lange ein Benutzeragent oder ein Transferagent bei der Interprozeßkommunikation maximal auf Daten warten soll.

Speicherung des Pfads des Arbeitsverzeichnisses: Hier wird das Verzeichnis angegeben, in dem sich die benötigten ProSet-Programme zur Simulation des E-Mail-Systems befinden.

Die persistenten Variablen werden im P-File EMail_Netzinformationen gespeichert.

Durch das Programm Netzaufbau wird das zu simulierende Netz nur spezifiziert. Um mit dem spezifizierten Netzwerk zu arbeiten, müssen die Transferagenten noch gestartet werden. Hierfür wird das Programm Netzwerk_aktivieren implementiert. Dieses Programm startet für jeden benötigten Transferagenten eine Instanz des Transferagentenprogramms. Als Argument wird jedem gestarteten Transferagenten der Name übergeben, der für ihn mit Hilfe des Programms Netzaufbau festgelegt wurde. Bei der Aktivierung des Beispielnetzwerks aus der Abbildung 4.27 würden also drei Transferagenten gestartet. Diesen Transferagenten würden die Namen TA_a, TA_b und TA_c zugeordnet.

Zusätzlich legt das Programm Netzwerk_aktivieren eine persistente Tabelle an, in der jedem Namen eines Transferagenten eine eindeutige Prozeßnummer zugeordnet ist.

Wenn die Simulation beendet werden soll, wird das Programm Netzwerk_deaktivieren aufgerufen. Durch dieses Programm werden die Prozesse der Transferagenten wieder beendet.

Änderungen und Erweiterungen der bestehenden Programme: Die in den Abschnitten 4.6.1 und 4.6.2 vorgestellten Programme für die Benutzeragenten und die Transferagenten müssen noch überarbeitet werden, damit Informationen zwischen den einzelnen Komponenten des zu simulierenden Netzwerks ausgetauscht werden können. Unter den Komponenten eines Netzwerks werden hier die Benutzeragenten und die Transferagenten verstanden. Da es sich bei den Benutzeragenten und Transferagenten um eigenständige Prozesse handelt, ist zur Simulation des Informationsaustauschs zwischen den Netzwerkkomponenten Interprozeßkommunikation erforderlich. Zur Interprozeßkommunikation werden C-Funktionen implementiert, die vom ProSet-Programm aus aufgerufen werden können. Die C-Funktionen zur Interprozeßkommunikation ermöglichen den Austausch von Zeichenketten zwischen verschiedenen Prozessen. Eine Zeichenkette enthält die Informationen, die in den Abbildungen 4.22 bis 4.26 für den Datenaustausch zwischen Prozessen beschrieben wurden. Zur Trennung der Felder für die verschiedenen Informationen wird beim implementierten Prototyp das Zeichen | benutzt. Dieses Zeichen darf deshalb kein Bestandteil eines durch den Benutzer des Prototyps eingegebenen Textes sein.

Beispiel: Die Zeichenkette, die der Benutzeragent des Benutzers Ben_X an den zugeordneten Transferagenten sendet, wenn eine Nachricht aus dem Nachrichtenspeicher seines Benutzers entfernt werden soll (siehe Abbildung 4.22), wird im folgenden betrachtet. Dem Prozeß des Benutzeragenten sei die eindeutige Identifikationsnummer 9999 zugeordnet. Der zu löschenden Nachricht sei die Nummer 2 zugeordnet. Die Zeichenkette, die der Benutzeragent an den Transferagenten versendet, hat das folgende Format:

"Nachricht aus Nachrichtenspeicher entfernen|Ben_X|9999|2".

Zur Unterstützung der Interprozeßkommunikation wird das Software-System PVM [GAJ+94] benutzt. PVM bietet die Möglichkeit, einen Informationsaustausch zwischen verschiedenen Prozessen durch „*message passing*“ sehr einfach zu realisieren.

Zusätzlich zur Erstellung der C-Funktionen müssen auch die bereits implementierten ProSet-Prototypen für die Benutzeragenten und die Transferagenten noch überarbeitet werden, um ein E-Mail-System simulieren zu könnnen. Die nötigen Änderungen und Erweiterungen werden im folgenden vorgestellt.

Der Benutzeragent: Für jeden Benutzeragenten, der simuliert werden soll, startet der Benutzer des Prototyps eine Instanz des ProSet-Benutzeragentenprogramms.

Zuordnung eines Benutzers: Weil verschiedene Benutzeragenten bei dem hier erstellten Protoyp durch verschiedene Instanzen desselben Benutzeragentenprogramms dargestellt werden, wird beim Start einer Instanz eines Benutzeragentenprogramms der Name des Benutzers, der mit diesem Agenten arbeitet, als Argument übergeben. So wird einer gestarteten Instanz des Benutzeragentenprogramms ein Benutzer zugeordnet. Der Benutzeragent verwendet den übergebenen Namen z.B. zur Erstellung der Absenderadresse beim Versenden von Nachrichten.

Beispiel: Wenn ein Benutzeragent für den Benutzer Ben_X gestartet werden soll, wird das Kommando Benutzeragent Ben_X eingegeben.

Erweiterung des Moduls zur Kommunikation: Bei dem im Abschnitt 4.6.1 vorgestellten Prototyp wurde die Kommunikation des Benutzeragenten mit dem zugeordneten Transferagenten durch eine Instanz des Moduls BA_Client_Kommunikation nur simuliert. Die Implementierung dieses Moduls wird erweitert, um die Kommunikation mit dem Transferagenten durchführen zu können. Zur Realisierung der Interprozeßkommunikation werden die implementierten C-Funktionen benötigt.
Der Name und die Prozeßnummer des zugordneten Transferagenten wird in den persistenten Tabellen gefunden, die durch die Programme Netzaufbau und Netzwerk_aktivieren angelegt wurden.

Der Transferagent: Die Transferagenten im Netzwerk werden durch mehrere Instanzen eines Transferagentenprogramms repräsentiert. Diese Instanzen werden durch das Programm Netzwerk_aktivieren gestartet. Die Ausführung der Instanzen wird durch das Programm Netzwerk_deaktivieren beendet. Damit die Transferagenten ihre vollständige Funktionalität erhalten, werden an der im Abschnitt 4.6.2 vorgestellten Implementierung noch die folgenden Änderungen bzw. Erweiterungen vorgenommen:

Zuordnung eines Namens: Wenn eine Instanz des Transferagentenprogramms durch das Programm Netzwerk_aktivieren gestartet wird, wird der Name des Transferagenten, der durch die neu gestartete Instanz repräsentiert wird, als Argument zum Programmaufruf erwartet.

Beispiel: Die Transferagenten des in der Abbildung 4.27 skizzierten Netzwerks haben die Namen TA_a, TA_b und TA_c. Der Transferagent TA_a wird z.B. durch den Programmaufruf Transferagent TA_a gestartet.

Kommunikation mit Benutzeragenten und fremden Transferagenten: Die Kommunikation mit Benutzeragenten oder mit fremden Transferagenten wurde bei dem im Abschnitt 4.6.2 vorgestellten Prototyp durch Instanzen der Moduln TA_Server_Kommunikation und TA_Client_Kommunikation nur simuliert. Die Implementierung dieser Moduln wird nun geändert und erweitert, um die Kommunikation durchführen zu können. Für die Interprozeßkommunikation wird auf die implementierten C-Funktionen zurückgegriffen. Zusätzlich wird der Modul Verbindungsinformationen implementiert. Jeder Transferagent besitzt eine Instanz dieses Moduls zur Verwaltung seiner Routing-Tabelle. Die nötigen Informationen zum Aufbau der Routing-Tabelle liefern die persistenten Variablen, die durch die Programme Netzaufbau und Netzwerk_aktivieren angelegt wurden.

Erweiterung der Implementierung für den Nachrichtenspeicher: Jeder Transferagent verwaltet Nachrichtenspeicher für eine festgelegte Menge von Benutzern. Die Benutzermengen für die verschiedenen Transferagenten können flexibel mit Hilfe des Programms Netzaufbau festgelegt werden. Bei dem im Abschnitt 4.6.2 vorgestellten Prototyp war noch fest im Programm vorgegeben, für welche Benutzer Nachrichtenspeicher verwaltet werden. Der Modul Nachrichtenspeicher wird deshalb überarbeitet. Jeder gestartete Transferagent verwaltet bei dem erweiterten Prototyp die Nachrichtenspeicher, die ihm durch den Benutzer des Prototyps zugeordnet sind.

Die Programme, aus denen die endgültige Version des Prototyps eines E-Mail-Systems besteht, sind in der Tabelle 4.9 noch einmal dargestellt. Die Programme Netzaufbau, Netzwerk_aktivieren und Netzwerk_deaktivieren sind nur Hilfsprogramme um den Prototyp zu konfigurieren, zu starten und um die Ausführung des Prototyps zu beenden. Sie stellen im Gegensatz zu den Benutzeragenten und den Transferagenten jedoch keine Bestandteile eines E-Mail-Systems dar und werden deshalb in den Objectory-Modellen nicht berücksichtigt.

Programm	Beschreibung
Netzaufbau	Die Struktur des zu simulierenden Netzwerks wird spezifiziert
Netzwerk_aktivieren	Die Transferagenten des spezifizierten Netzwerks werden gestartet.
Transferagent	Durch Instanzen dieses Programms werden die Transferagenten simuliert.
Benutzeragent	Durch Instanzen dieses Programms werden die Benutzeragenten simuliert.
Netzwerk_deaktivieren	Die Ausführung der durch das Programm Netzwerk_aktivieren gestarteten Instanzen des Programms Transferagent wird beendet.

Tabelle 4.9: Programme zur Simulation des E-Mail-Systems.

Ausblick auf weitere Arbeiten am Entwurfsmodell

Die Erstellung des im Abschnitt 4.6.3 beschriebenen vollständigen Prototyps ist nur ein Zwischenschritt zur Erstellung eines vollständigen Entwurfsmodells. Für ein vollständiges Entwurfsmodell müssen die erstellten Interaktionsdiagramme noch erweitert werden. Bei den Interaktionsdiagrammen wurde bisher nur die Kommunikation innerhalb eines Benutzeragenten bzw. innerhalb eines Transferagenten beschrieben. Die Kommunikation zwischen Benutzeragenten und Transferagenten bzw. zwischen verschiedenen Transferagenten wird noch nicht berücksichtigt.

Für eine Darstellung der Kommunikationsbeziehungen zwischen den verschiedenen Prozessen des E-Mail-Systems in den Interaktionsdiagrammen bietet der Prototyp eine gute Grundlage. Die Art und die Reihenfolge der zwischen den Prozessen auszutauschenden Stimuli kann zum Teil direkt aus der Implementierung des Prototyps entnommen werden. Dabei muß jedoch berücksichtigt werden, daß ein Protokoll für ein in der Praxis eingesetztes E-Mail-System wesentlich umfangreicher wäre, als das Protokoll, das der Implementierung des Prototyps zugrunde liegt. Es könnte z.B. nötig sein, zusätzliche Ausnahmefälle, wie z.B. Verfälschungen des Nachrichteninhalts während der Datenübertragung, in den Interaktionsdiagrammen zu berücksichtigen.

4.7 Fazit

In diesem Abschnitt werden die Erkenntnisse, die sich bei der Bearbeitung des betrachteten Fallbeispiels ergeben haben, beschrieben. Dazu sollen zunächst noch einmal wesentliche Punkte aus den Abschnitten 4.4 bis 4.6 zusammengefaßt werden.

Anforderungsmodell und Analysemodell (Abschnitte 4.4 und 4.5)

- Eine Einteilung von Komponenten des E-Mail-Systems in Werkzeuge und Materialien ist in einigen Fällen problematisch. So erfüllen Transferagenten und die Verbindungen zwischen verschiedenen Agenten weder die Kriterien für Materialien noch die Kriterien für Werkzeuge. Auch die Berücksichtigung der Metapher Automat hilft hier nicht weiter.

  Um alle Objekte des Problembereichsmodells mit den Metaphern beschreiben zu können, die durch den WAM-Ansatz zur Verfügung gestellt werden, werden die Transferagenten und die Verbindungen bei der Betrachtung des E-Mail-Systems den Werkzeugen zugeordnet.

- Bei der Erstellung des Analysemodells können sämtliche Materialien, die im Problembereichsmodell aufgeführt sind, übernommen werden. Das Analysemodell enthält nur ein Materialobjekt, das nicht im Problembereichsmodell aufgeführt ist (das Objekt Verbindungsinformationen).

  Die Modellierung der Werkzeuge im Problembereichsmodell bietet zwar eine grobe Übersicht über den Aufbau eines E-Mail-Systems. Sie gibt im Gegensatz zur Modellierung der Materialien im Problembereichsmodell für die weitere Systementwicklung jedoch keine wesentliche Hilfe, weil die Modellierung der Werkzeuge im Problembereichsmodell noch nicht detailliert genug ist. So sind im Problembereichsmodell des E-Mail-Systems zwar die Objekte Benutzeragent und Transferagent aufgeführt. Über den inneren Aufbau von Benutzeragenten und Transferagenten gibt das Problembereichsmodell jedoch keinen Aufschluß. Darüber hinaus werden bei der weiteren Systementwicklung nicht mehr alle der im Problembereichsmodell aufgeführten Werkzeugobjekte benötigt (z.B. werden physikalische Verbindungen im Analysemodell bereits vernachlässigt).

Entwurfsmodell und Prototyp (Abschnitt 4.6)

- Der Aufbau eines Benutzeragenten und eines Transferagenten, der im Analysemodell beschrieben wird, bietet eine gute Grundlage für die Modellierung des Benutzeragenten und des Transferagenten im Entwurfsmodell. Trotz einiger Änderungen, die bei der Erstellung des Entwurfsmodells an der Struktur der Agenten vorgenommen werden, bleiben große Teile der im Analysemodell vorgestellten Struktur erhalten.
- Die Modellierung der Materialien im Analysemodell wird im Entwurfsmodell nur zu sehr geringen Teilen übernommen. Dies liegt daran, daß Entwurfsmodellobjekte in der Implementierung durch Modulinstanzen repräsentiert werden und bei vielen Materialien eine Abbildung auf Modulinstanzen nicht erwünscht ist. Hiervon sind die Materialien betroffen, die zwischen verschiedenen Prozessen ausgetauscht werden müssen. Ein Austausch von Modulinstanzen zwischen verschiedenen Prozessen wird als zu aufwendig eingeschätzt.

  Trotzdem ist auch die Modellierung der nicht im Entwurfsmodell aufgeführten Materialobjekte im Analysemodell sinnvoll, weil durch die Modellierung der Materialien im Analysemodell dargestellt wird, aus welchen Bestandteilen sich Materialien (z.B. Nachrichten und Quittungen) zusammensetzen. Die Informationen über die Zusammensetzung von Materialien können bei der Implementierung von Prozeduren des ProSet-Prototyps genutzt werden, wenn Datenstrukturen definiert werden müssen, in denen Informationen über Materialien oder über einzelne Bestandteile von Materialien vorübergehend gespeichert werden sollen. Dies wurde im Abschnitt 4.6.1 exemplarisch für die Speicherung des Inhalts einer Nachricht innerhalb einer ProSet-Prozedur mit Hilfe eines Tupels beschrieben (siehe Beschreibung der Abbildung 4.17).
- Die enge Verzahnung zwischen der Erstellung des Entwurfsmodells und den Prototypingphasen erwies sich als sehr zweckmäßig. Die Prototypen konnten anhand bestehender Entwurfsmodellteile einfach erstellt werden und lieferten Aufschlüsse über Verbesserungsmöglichkeiten und mögliche Weiterentwicklungen des Entwurfsmodells. Hierzu wurden in den Abschnitten 4.6.1 und 4.6.3 die folgenden Beispiele erwähnt:

  Abschnitt 4.6.1:

  - Im Entwurfsmodell des Benutzeragenten wurde das Objekt BA_Start neu eingefügt.
  - Ein eigenständiges Objekt zur Kontrolle des Anwendungsfalls Nachricht erzeugen wurde im Entwurfsmodell des Benutzeragenten aufgrund der Ergebnisse des Prototyping entfernt.
  - Durch das Prototyping fallen Stimuli, die in den erstellten Interaktionsdiagrammen fehlen oder falsch eingetragen wurden, sowie Aktionen, die in den Interaktionsdiagrammen nicht oder falsch beschrieben wurden, auf. Die Interaktionsdiagramme können während des Prototyping oder nach dem Prototyping korrigiert werden.

  Abschnitt 4.6.3:

  - Bei den in dieser Arbeit betrachteten Interaktionsdiagrammen wird die Kommunikation zwischen Benutzeragenten und Transferagenten bzw. zwischen verschiedenen Transferagenten noch nicht berücksichtigt. Bei einer entsprechenden Erweiterung der Interaktionsdiagramme sollte zumindest ein Teil der Stimuli, die noch benötigt werden, und ein Teil der Aktionen, die noch beschrieben werden müssen, anhand des erstellten Prototyps identifiziert werden können.

Bei der Abbildung erstellter Entwurfsmodellteile auf ein ProSet-Programm ergaben sich keine Schwierigkeiten.

Abschließend sollen noch einige ergänzende Betrachtungen zur Erstellung des Prototyps des E-Mail-Systems angestellt werden.

- Bei der Erstellung des Prototyps für das E-Mail-System traten weniger Schwierigkeiten auf als bei der Erstellung des Prototyps für das Bibliotheksverwaltungssystem. Hierfür können die folgenden Ursachen angeführt werden:

  - Der ProSet-Compiler und der H-PCTE-Server liefen bei der Implementierung des Prototyps für das E-Mail-System bereits sehr stabil. Fehler traten zwar weiterhin bei der Ausnahmebehandlung für innerhalb von Prozeduren deklarierte persistente Variablen oder Konstanten auf (vgl. Abschnitt 3.6). Diese Probleme haben die Entwicklung des Prototyps jedoch nicht behindert.
  - Einschränkungen und Probleme bei der Nutzung persistenter Variablen waren bereits durch die Entwicklung des Prototyps für das System zur Bibliotheksverwaltung bekannt (siehe Abschnitt 3.6). Sie wurden bei der Implementierung des Prototyps für das betrachtete E-Mail-System berücksichtigt. So wurde z.B. die Idee, die Interprozeßkommunikation mit Hilfe des Persistenzmechanismus zu realisieren, bereits vor der Implementierung des Prototyps verworfen (siehe Abschnitt 4.6.3, Beschreibung des Datenaustauschs beim Prototyp).
  - Die Benutzeragenten und die Transferagenten müssen nur lesend auf persistente Daten zugreifen. Es reicht daher aus, in den Prozeduren der Moduln des Benutzeragentenprogramms und des Transferagentenprogramms persistente Konstanten zu deklarieren. Die Deklaration persistenter Variablen ist nicht erforderlich. Die Übersetzung eines ProSet-Programms, in dem persistente Konstanten deklariert sind, ist unproblematisch. Bei einer Nutzung persistenter Variablen kann dagegen die Übersetzung eines Programms mit der Fehlermeldung internal assembler error: out of memory abgebrochen werden, wenn das zu übersetzende Programm zu umfangreich wird (siehe hierzu Abschnitt 3.5.1).

    Schreibend müssen nur die Hilfsprogramme Netzaufbau und Netzwerk_aktivieren auf persistente Daten zugreifen. Diese Programme besitzen jedoch nur einen geringen Umfang. Eine Übersetzung dieser Programme führt nicht zu Problemen.
  - Im Entwurfsmodell wird bereits berücksichtigt, daß Beziehungen zwischen Klassen, wie z.B. Objectory-„extends“-Beziehungen nicht direkt implementiert werden können. Bei der Erstellung des Prototyps für das E-Mail-System müssen deshalb auch keine Beziehungen zwischen Klassen mehr beachtet werden.

    Im Gegensatz dazu mußten bei der Erstellung des Prototyps für das System zur Bibliotheksverwaltung die Vererbungsbeziehungen, die im erstellten Teil des Analysemodells für dieses Fallbeispiel dargestellt sind, auf ein ProSet-Programm übertragen werden.

- Persistente Daten spielen bei dem betrachteten E-Mail-System eine weniger wichtige Rolle als ursprünglich angenommen wurde. Dies hat die folgende Ursache:

  - In der Einleitung zum Kapitel 4 wurde bereits erwähnt, daß die grundlegende Idee, ein E-Mail-System als Beispiel für eine Anwendung, in der persistente Daten benötigt werden, zu betrachten, von Gamerman et al. stammt [GLV92]. Das in der Diplomarbeit modellierte System unterscheidet sich jedoch von dem System, das bei Gamerman et al. vorgestellt wird. Hierauf wurde bereits bei der Beschreibung der Anforderungen an das in der Diplomarbeit zu modellierende System eingegangen (siehe Abschnitt 4.3). Während in der Diplomarbeit ein Nachrichtenübertragungssystem modelliert wird, das aus beliebig vielen Transferagenten besteht, die untereinander Daten austauschen, wird das Nachrichtenübertragungssystem bei Gamerman et al. durch eine Datenbank simuliert. Die Arbeit, die bei Gamerman et al. durch die Datenbank erledigt wird, wird bei dem in der Diplomarbeit erstellten Prototyp durch verschiedene Instanzen des Transferagentenprogramms durchgeführt.

Da Transferagenten ein wesentlicher Bestandteil des E-Mail-Systems sind, das in der Ist-Analyse (Abschnitt 4.2) und bei der Beschreibung der Anforderungen (Abschnitt 4.3) beschrieben wird, erschien es nicht zulässig, die Transferagenten bei der Erstellung der Objectory-Modelle oder beim experimentellen Prototyping zu vernachlässigen.

Statt auf der Speicherung von Daten liegt der Schwerpunkt bei dem E-Mail-System, das in diesem Kapitel betrachtet wurde, stärker auf dem Austausch von Daten zwischen verschiedenen Prozessen, die weitgehend unabhängig voneinander agieren. Hier wäre eine Nutzung der Möglichkeiten zur Parallelprogrammierung, die von ProSet angeboten werden, sinnvoll gewesen. Die Nutzung dieser Möglichkeiten war jedoch nicht möglich, weil die Verwendung des ProSet-Persistenzmechanismus die Parallelprogrammierung bei den bislang zur Verfügung stehenden Compilerversionen noch ausschließt. Die Implementierung des ProSet-Prototyps unter Nutzung des Persistenzmechanismus war jedoch durch die Aufgabenstellung dieser Diplomarbeit vorgegeben.

Kapitel 5

Drittes Fallbeispiel: Ein System zur Pflegedienst- und Terminplanung im stationären Krankenhausbereich

Anhand der Entwicklung eines Systems zur *Pflegedienst- und Terminplanung* im stationären Krankenhausbereich soll evolutionäres Prototyping mit ProSet im Zusammenhang mit dem Objectory-Entwicklungsprozeß untersucht werden. Zunächst wird ein Pilotsystem implementiert, mit dem die Diensteinteilung für das Pflegepersonal geplant werden kann. In einem zweiten Schritt soll dieses System um eine Komponente zur jährlichen Terminplanung der Ärzte und des Pflegepersonals erweitert werden.

Analog zu den Beschreibungen der bisher betrachteten Fallbeispiele (Kapitel 3 und 4) wird zunächst allgemein die Vorgehensweise für die Bearbeitung dieses Fallbeispiels vorgestellt (Abschnitt 5.1). Eine Beschreibung der derzeitigen Praxis bei der Pflegedienst- und Terminplanung im Krankenhaus, sowie zusätzliche Informationen, die für die Bearbeitung dieses Fallbeispiels relevant sind, werden im Abschnitt 5.2 aufgeführt. Im Abschnitt 5.3 werden anschließend die Anforderungen an das zu entwickelnde Pilotsystem vorgestellt. Die Beschreibungen der für die Entwicklung des Pilotsystems erstellten Objectory-Modelle, sowie eine Zusammenfassung der Implementierung und des Tests des Pilotsystems befinden sich in Abschnitten 5.4 und 5.5.

In den Abschnitten 5.6 bis 5.8 wird auf die Erweiterung des Pilotsystems eingegangen. Im Abschitt 5.6 werden die Anforderungen an das erweiterte System informell beschrieben. Die Abschnitte 5.7 und 5.8 enthalten Beschreibungen der erstellten Objectory-Modelle und eine Zusammenfassung der Implementierung und des Tests des erweiterten Systems.

Zum Abschluß dieses Kapitels werden im Abschnitt 5.9 noch einmal die Ergebnisse, die sich bei der Bearbeitung dieses Fallbeispiels ergeben haben, zusammengefaßt.

5.1 Vorgehensweise

Zunächst soll eine Ist-Analyse durchgeführt werden, bei der die Informationen, die für die Bearbeitung dieses Fallbeispiels relevant sind, gesammelt werden. Wie bei den bisher betrachteten Fallbeispielen ist die Ist-Analyse aus Zeitgründen kürzer und oberflächlicher als dies bei der Entwicklung eines Software-Systems, das in der Praxis tatsächlich eingesetzt werden soll, der Fall wäre. Im Anschluß an die Ist-Analyse werden zunächst die Anforderungen an das Pilotsystem informell beschrieben. Anschließend wird ein Glossar mit Begriffen aus dem Anwendungsbereich erstellt (siehe auch Abschnitt 2.2). Dieses Glossar wird bei der weiteren Systementwicklung ständig ergänzt. Das Glossar zu dem betrachteten Fallbeispiel befindet sich im Zusatzdokument zu dieser Diplomarbeit.

Im Pilotsystem wird nur die Pflegedienstplanung berücksichtigt. Die individuelle längerfristige Terminplanung der Ärzte und der Pflegekräfte (z.B. die Planung von Urlaubsterminen oder von Terminen für Fortbildungsmaßnahmen) wird noch außer acht gelassen. Auf der Basis der Beschreibung der Anforderungen wird das Pilotsystem als eigenständiges System unabhängig von später betrachteten Erweiterungen entwickelt. Das Pilotsystem wird in der Sprache ProSet implementiert (siehe Kapitel 1). Eine Transformation des ProSet-Quellcodes in den Code einer anderen Programmiersprache wird im Rahmen dieser Diplomarbeit nicht mehr durchgeführt.

Nachdem das Pilotsystem in der Sprache ProSet implementiert ist und die Tests abgeschlossen sind, wird das Pilotsystem erweitert. Dazu werden zunächst die Anforderungen an das erweiterte System vorgestellt. Anschließend werden die Objectory-Modelle, die bereits für das Pilotsystem erstellt wurden, erweitert. Ziel ist es, das Pilotsystem so zu ergänzen, daß neben der Planung des Pflegedienstes auch die individuelle Terminplanung der Ärzte und der Pflegekräfte (z.B. für Urlaube oder Fortbildungsmaßnahmen) unterstützt wird.

Durch die Trennung der Entwicklung des Pilotsystems und der Erweiterung des Pilotsystems bei dem in diesem Kapitel betrachteten Beispiel wird berücksichtigt, daß Anforderungen von Endbenutzern an eine Erweiterung eines Software-Systems im Rahmen einer evolutionären Systementwicklung zum Teil erst dann entstehen, wenn erste praktische Erfahrungen mit dem System, das erweitert werden soll, vorliegen [BKKZ92, S. 69]. So ist es nach Budde et al. nicht möglich, sämtliche Anforderungen an ein Software-System vor der Erstellung des Systems vollständig zu bestimmen [BKKZ92, S. 67]. Das erweiterte System stellt selbst wieder ein neues Pilotsystem für eine weitere evolutionäre Systementwicklung dar, die im Rahmen dieser Diplomarbeit jedoch nicht mehr betrachtet wird.

Datenschutz- und Datensicherheitsaspekte werden bei diesem Fallbeispiel nicht berücksichtigt. Sie könnten der Gegenstand einer späteren Erweiterung des Systems im Rahmen der evolutionären Systementwicklung sein. Eine Diskussion über Schutzkonzepte im Kontext von ProSet sowie deren Realisierung auf Basis des Objektverwaltungssystems H-PCTE befindet sich in der Diplomarbeit von Kappert [Kap95].

5.2 Ist-Analyse

In diesem Abschnitt wird die derzeitige Praxis bei der Pflegedienst- und Terminplanung beschrieben, wie sie derzeit noch in vielen Krankenhäusern durchgeführt wird. Leichte Abweichungen bei der Planung zwischen verschiedenen Krankenhäusern sind dabei jedoch möglich. Grundlage der folgenden Beschreibung bildet die Pflegedienst- und Terminplanung des St. Irmgardis Krankenhauses in Viersen.

Jede Pflegekraft ist einer *Station* (Abteilung im Krankenhaus) zugeteilt. Die Einteilung des Pflegedienstes wird in den einzelnen Stationen von der *Stationsleitung* durchgeführt. Die Stationsleitung besteht aus einer Krankenschwester oder aus einem Krankenpfleger. Zur Unterstützung der Pflegedienstplanung existiert ein Formular. Die wichtigsten Felder dieses Formulars sind in der Abbildung 5.1 skizziert. Dienstpläne für die Tagschichten (Früh- und Spätschicht) und für die Nachtschichten werden separat erstellt. Ein Dienstplan bezieht sich immer auf den Zeitraum eines Monats. Personal für Früh- und Spätschichten wird üblicherweise nicht in Nachtschichten eingesetzt. Personal für Nachtschichten wird üblicherweise nicht in Früh- und Spätschichten eingesetzt. Die erstellten Dienstpläne werden einer übergeordneten Stelle, der *Pflegedienstleitung*, die für das gesamte Krankenhaus zuständig ist, zur Genehmigung vorgelegt. Eine stationsübergreifende Koordinierung des Pflegedienstes findet jedoch nicht statt.

Bei der Erstellung des Dienstplans muß berücksichtigt werden, daß in jeder Schicht mindestens eine ausgebildete Krankenschwester oder ein ausgebildeter Krankenpfleger arbeitet.

Bei der Abrechnung der geleisteten Arbeitsstunden wird zwischen Dienst-Tagen, freien Tagen und Urlaubstagen unterschieden. Freie Tage gehen nicht in die Berechnung der geleisteten Stunden ein. Für Urlaubstage werden der Pflegekraft Arbeitsstunden gutgeschrieben. Die Anzahl der Ar-

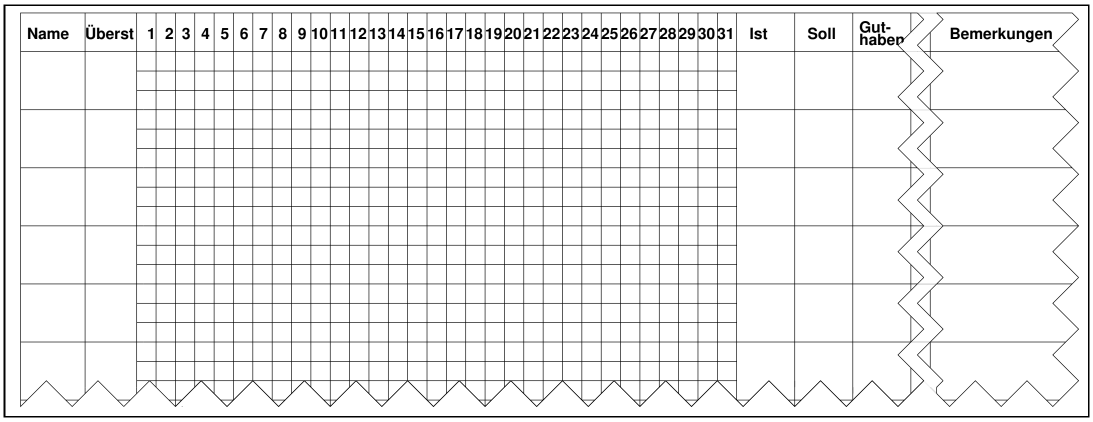

Abbildung 5.1: Skizze eines Formulars zur Pflegedienstplanung.

In der Spalte Name sind die Namen der Pflegekräfte aufgeführt. Die Spalte Überst enthält die Anzahl der Überstunden, die die Pflegekräfte noch aus dem vorherigen Monat übernehmen können. Die folgenden 31 Spalten dienen dazu, den Dienstplan für alle Tage eines Monats aufzustellen. Falls der betrachtete Monat weniger als 31 Tage hat, werden die nicht benötigten Spalten frei gelassen. Für jede Person existieren drei Zeilen. In der ersten Zeile wird der geplante Dienst der Pflegekraft eingetragen (z.B. f für Frühschicht, s für Spätschicht, 0 für frei, u für Urlaub). Die anderen Zeilen werden genutzt, um zusätzliche Informationen aufzunehmen. Hier werden z.B. nachträgliche Änderungen oder geleistete Überstunden vermerkt. In die Spalte Ist wird nach Ablauf eines Monats die Anzahl der abgeleisteten Stunden einschließlich der alten Überstunden aus der Spalte Überst eingetragen. Die Spalte Soll enthält die Anzahl der Soll-Stundenzahlen der Pflegekräfte. Die Differenz zwischen den Werten in der Ist- und in der der Soll-Stunden-Spalte wird in der Spalte Guthaben eingetragen. In die Spalte Bemerkungen können weitere Informationen, wie z.B. Gründe für Überstunden, aufgenommen werden.

beitsstunden, die für einen Urlaubstag angerechnet werden, kann bei verschiedenen Pflegekräften unterschiedlich sein. So werden einer Vollzeitpflegekraft mehr Stunden für einen Urlaubstag angerechnet als einer Pflegekraft, die nur wenige Tage im Monat arbeitet.

Neben der Planung des Pflegedienstes muß die Urlaubsplanung der Pflegekräfte abgestimmt werden. Analog zur Pflegedienstplanung wird auch die Planung der Urlaubstermine der Pflegekräfte stationsintern geregelt. Eine stationsübergreifende Koordinierung findet auch hier nicht statt. Zur Unterstützung der Planung befindet sich in jeder Station ein Terminkalender, der den Zeitraum eines Jahres umfaßt. In diesem Terminkalender werden die geplanten Urlaubstermine der Pflegekräfte markiert. Das Aussehen dieses Terminkalenders ist in der Abbildung 5.2 skizziert. In diesen Kalender werden nur Urlaubstermine eingetragen. Ein Hilfsmittel zur Koordination sonstiger Termine (z.B. für Fortbildungsmaßnahmen) existiert nicht.

Einschränkungen bei der Urlaubsplanung ergeben sich dadurch, daß die Bedingungen bei der Planung des Pflegedienstes (siehe oben) weiter eingehalten werden müssen, d.h. alle Schichten müssen mit mindestens einer Krankenschwester oder einem Krankenpfleger besetzt werden können. Dabei muß ein möglicher Ausfall von Pflegekräften (z.B. durch Krankheit) berücksichtigt werden.

Die Terminplanung der Ärzte im Krankenhaus ist unabhängig von der Terminplanung des Pflegepersonals. Sie wird von den beteiligten Ärzten ohne Absprache mit den Stationsleitungen oder mit der Pflegedienstleitung durchgeführt.

Für die Verwaltung der Daten der Ärzte und der Pflegekräfte des Krankenhauses sind Verwaltungsangestellte zuständig. Zur Unterstützung der Verwaltungangestellten wird für jeden Angestellten des Krankenhauses (z.B. Ärzte und Pflegekräfte) eine Personalakte angelegt, die die benötigten persönlichen Daten des Angestellten enthält.

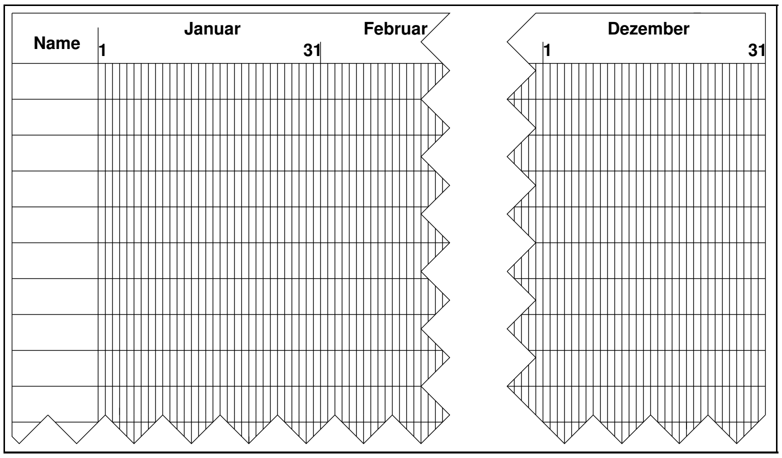

Abbildung 5.2: Skizze eines Terminkalenders.

5.3 Beschreibung der Anforderungen an das Pilotsystem

Zunächst wird ein Pilotsystem implementiert, das die monatliche Pflegedienstplanung und die Abrechnung der geleisteten Arbeitsstunden unterstützt. Die jährliche Terminplanung für Pflegekräfte und Ärzte wird noch nicht berücksichtigt. Das Pilotsystem erfüllt die folgenden Aufgaben:

- Die folgenden Daten des Pflegepersonals können gespeichert werden:
  - Name,
  - Ausbildung,
  - Bezeichnung der Station, auf der die Pflegekraft arbeitet,
  - vorgesehene durchschnittliche Wochenarbeitszeit (im Krankenhaus wird häufig mit Aushilfen gearbeitet, die weniger Stunden arbeiten als Vollzeitkräfte),
  - angerechnete Arbeitsstunden pro Schicht,
  - Stunden, die für einen Urlaubstag angerechnet werden,
  - Tag- oder Nachtschicht,
  - zustehende Urlaubstage,
  - ein zusätzlicher Kommentar.

  Die Daten der Personen können bei Bedarf geändert werden.

- Die monatliche Dienstplanung der einzelnen Stationen wird unterstützt. Für jede Station kann der Dienstplan (siehe Abbildung 5.1) erstellt werden.

- Dienstpläne werden durch die Pflegedienstleitung genehmigt. Dabei kann überprüft werden, ob bei der Dienstplanung berücksichtigt worden ist, daß in jeder Schicht mindestens eine ausgebildete Pflegekraft arbeiten muß.

- In dem Dienstplan können nachträgliche Änderungen der Diensteinteilung, geleistete Überstunden und Ausfälle von Pflegekräften eingetragen werden. Zu jeder im Dienstplan eingetragenen Person können zusätzliche Informationen, wie z.B. der Besuch von Fortbildungsmaßnahmen gespeichert werden.

- Am Ende eines Monats können die geleisteten Arbeitsstunden durch das Pilotsystem berechnet und mit den Soll-Stundenzahlen verglichen werden. Die Anzahl der insgesamt geleisteten Überstunden für die einzelnen Pflegekräfte wird automatisch ermittelt und gespeichert.

- Dienstpläne, die nicht mehr benötigt werden, können gelöscht werden. Vorgaben, nach denen nur solche Dienstpläne gelöscht werden dürfen, deren Daten gemäß den gesetzlichen Vorschriften ausreichend lange aufbewahrt wurden, werden bei dem in dieser Arbeit erstellten System nicht beachtet.

5.4 Analyse und Entwurf des Pilotsystems

In diesem Abschnitt werden die Erstellung des Objectory-Anforderungsmodells, des Objectory-Analysemodells und des Objectory-Entwurfsmodells für das Pilotsystem beschrieben.

5.4.1 Das Anforderungsmodell

Im Anforderungsmodell werden wie bei den bereits in den Kapiteln 3 und 4 betrachteten Fallbeispielen nur das Anwendungsfallmodell und das Problembereichsmodell betrachtet. Da Benutzerschnittstellen für diese Diplomarbeit nur eine untergeordnete Rolle spielen (siehe Abschnitt 1.2), wird auf eine Beschreibung der Benutzeroberflächen im Rahmen des Anforderungsmodells verzichtet. Bei der Software-Entwicklung im Rahmen eines realen Projektes wären Beschreibungen von Benutzeroberflächen jedoch erforderlich. Jacobson et al. empfehlen hierzu die Entwicklung von Oberflächenprototypen [JCJÖ93, S. 129].

Um eine spätere Umstellung der Benutzeroberflächen der bei diesem Fallbeispiel erstellten Programme zu vereinfachen, werden die Schnittstellenoperationen in Objekten gekapselt. Die Benutzeroberflächen könnten dadurch weitgehend unabhängig vom übrigen System entwickelt und anschließend angepaßt werden.

Das Anwendungsfallmodell

Die Akteure des Anwendungsfallmodells des Pilotsystems sind in der Abbildung 5.3 dargestellt. Dabei handelt es sich um die Akteure Pflegekraft, Stationsleitung, Pflegedienstleitung und Verwaltungsangestellter.

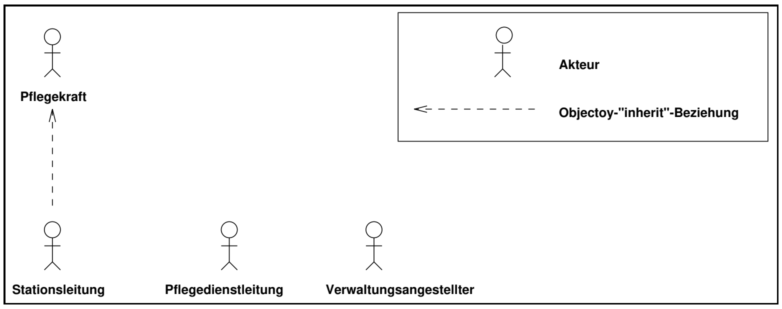

Abbildung 5.3: Akteure für das Pilotsystem.

Die benötigten Anwendungsfälle sind in der Abbildung 5.4 aufgeführt. Anwendungsfälle, die sich auf die Konfigurierung des Systems beziehen, werden analog zum Anwendungsfallmodell des E-Mail-Systems (siehe Abschnitt 4.4.1) und im Gegensatz zum Anwendungsfallmodell des Systems zur Bibliotheksverwaltung (siehe Abschnitt 3.4.1) nicht aufgeführt. Nach den Erfahrungen aus den ersten beiden Fallbeispielen wird die weitere Systementwicklung durch die Vernachlässigung dieser Anwendungsfälle nicht beeinträchtigt. Die Anwendungsfälle des Pilotsystems sollen im folgenden kurz vorgestellt werden:

Der Anwendungsfall Personaldaten bearbeiten: Durch die Ausführung einer Instanz dieses Anwendungsfalls können Daten des Krankenhauspersonals neu eingegeben, geändert oder (einschließlich gutgeschriebener Überstunden) gelöscht werden.

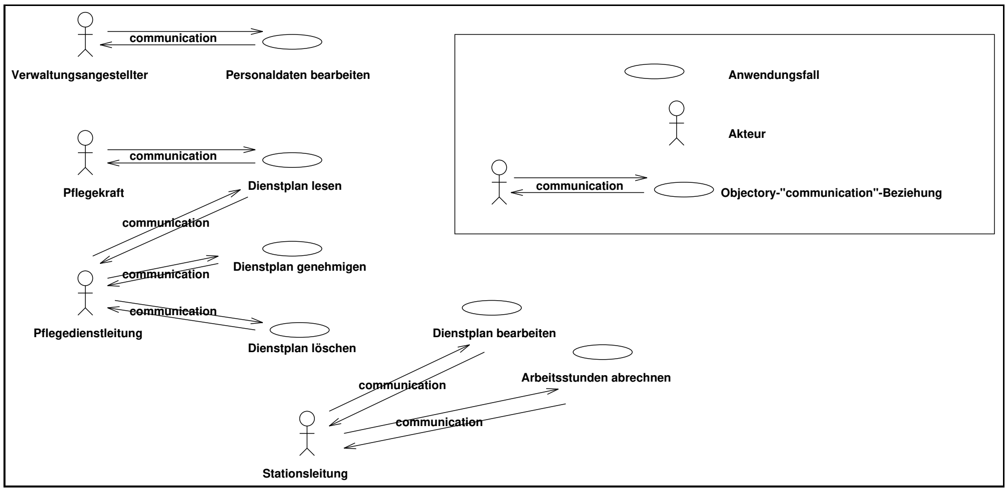

Abbildung 5.4: Anwendungsfälle für das Pilotsystem.

Der Anwendungsfall Dienstplan lesen: Der Anwendungsfall Dienstplan lesen ermöglicht den Pflegekräften und der Pflegedienstleitung das Lesen eines Dienstplans. Zusätzlich wird angezeigt, ob ein Dienstplan bereits durch Pflegedienstleitung genehmigt wurde und ob die auf dem Dienstplan eingetragenen Arbeitsstunden bereits abgerechnet wurden.

Der Anwendungsfall Dienstplan genehmigen: Durch die Ausführung einer Instanz des Anwendungsfalls Dienstplan genehmigen kann die Pflegedienstleitung einen Dienstplan genehmigen. Dabei wird durch das zu entwerfende System überprüft, ob auf dem Dienstplan alle Angaben, die für die spätere Abrechnung der geleisteten Arbeitsstunden nötig sind, vorhanden sind und ob alle Schichten mit mindestens einer Krankenschwester oder einem Krankenpfleger besetzt sind. Sind die Bedingungen nicht erfüllt, wird ein Warnhinweis ausgegeben. Die Pflegedienstleitung kann nach der Ausgabe des Warnhinweises noch von der Genehmigung des ausgewählten Dienstplans absehen.

Der Anwendungsfall Dienstplan bearbeiten: Durch den Anwendungsfall Dienstplan bearbeiten wird das Erstellen und das Ändern von Dienstplänen ermöglicht. Es können nur Dienstpläne geändert werden, bei denen die eingetragenen Arbeitsstunden noch nicht abgerechnet wurden. Wenn eine Änderung an einem bereits genehmigten Dienstplan durchgeführt werden soll (z.B. der Tausch der Dienste zweier Pflegekräfte), werden sowohl die ursprünglichen Eintragungen als auch die Änderung gespeichert. Wenn ein neuer Dienstplan erstellt werden soll, werden automatisch alle benötigten Informationen über die betroffenen Pflegekräfte (z.B. Name, Anzahl der gutgeschriebenen Überstunden) auf dem Dienstplan eingetragen.

Der Anwendungsfall Arbeitsstunden abrechnen: Die Abrechnung der in einem Dienstplan eingetragenen Arbeitsstunden und das Modifizieren der Überstunden, die für die einzelnen Pflegekräfte gespeichert werden, wird ermöglicht. Die in einem Dienstplan eingetragenen Arbeitsstunden werden nur dann abgerechnet, wenn dieser Dienstplan bereits durch die Pflegedienstleitung genehmigt wurde.

Der Anwendungsfall Dienstplan löschen: Dienstpläne, die nicht mehr benötigt werden, können durch die Pflegedienstleitung gelöscht werden. Bei einem System, das für einen tatsächlichen Einsatz in der Praxis entwickelt wird, wäre es nötig sicherzustellen, daß nur solche Dienstpläne gelöscht werden können, deren Daten gemäß den gesetzlichen Vorgaben ausreichend lange aufbewahrt wurden. In dieser Diplomarbeit werden diese gesetzlichen Vorgaben nicht beachtet (siehe Abschnitt 5.3).

Die Struktur der Anwendungsfallbeschreibungen entspricht der Struktur der Beschreibungen der Anwendungsfälle des Systems zur Bibliotheksverwaltung (siehe Abschnitt 3.4.1) und des E-Mail-Systems (siehe Abschnitt 4.4.1). Auf die Angabe eines Beispiels für eine vollständig Beschreibung eines Anwendungsfalls wird daher in diesem Abschnitt verzichtet. Die vollständigen Beschreibungen aller Anwendungsfälle dieses Fallbeispiels befinden sich im Zusatzdokument zur Diplomarbeit.

Das Problembereichsmodell

Die Problembereichsmodelle der in den Kapiteln 3 und 4 betrachteten Fallbeispiele sind unterschiedlich ausführlich. Die Unterschiede dieser Modelle wurden bereits im Abschnitt 4.4.2 beschrieben und begründet. Wesentlich für die Entscheidung, das Problembereichsmodell des E-Mail-Systems nicht so ausführlich zu gestalten wie das Problembereichsmodell des Systems zur Bibliotheksverwaltung, waren zum einen die geringere Anzahl der für das E-Mail-System benötigten Daten und die geringere Komplexität der Beziehungen zwischen den Daten. Zum anderen wurde berücksichtigt, daß bis zur Implementierung eines Prototyps für das E-Mail-System noch ein vollständiges Analysemodell und große Teile des Entwurfsmodells erstellt wurden. Der Prototyp des Systems zur Bibliotheksverwaltung basierte hingegen ausschließlich auf dem Anwendungsfallmodell und auf einem Teil des Analysemodells.

Viele Rahmenbedingungen des Systems zur Pflegedienst- und Terminplanung im stationären Krankenhausbereich gleichen den Rahmenbedingungen des E-Mail-Systems. Auch beim System zur Pflegedienst- und Terminplanung ist die Anzahl der verschiedenen benötigten Daten weitaus geringer als beim System zur Bibliotheksverwaltung (dies wird bei der Aufzählung der benötigten Materialien im weiteren Verlauf dieses Abschnittes noch deutlicher werden). Zwischen den verschiedenen Daten existieren daüber hinaus nicht so viele Beziehungen, die berücksichtigt werden müssen, um die Konsistenz des Datenbestandes zu erhalten. Des weiteren werden bis zur Implementierung des Programms das Anforderungsmodell, das Analysemodell und das Entwurfsmodell erstellt.

Deshalb wird beim Problembereichsmodell des in diesem Kapitel betrachteten Fallbeispiels analog zum E-Mail-System nicht jeder Anwendungsfall separat betrachtet. Es wird ein Problembereichsmodell erstellt, in dem Objekte aus den Problembereichen sämtlicher Anwendungsfälle gemeinsam berücksichtigt werden.

In die Problembereichsmodelle des Systems zur Bibliotheksverwaltung und des E-Mail-Systems wurden nur Objekte aufgenommen, die nicht als Attribute anderer Objekte angesehen werden. Dies wurde bereits in den Abschnitten 3.1 und 4.4.2 diskutiert. Dieses Vorgehen hat sich als günstig erwiesen und soll auch für das hier beschriebene Problembereichsmodell übernommen werden.

Gegenüber dem Problembereichsmodell des E-Mail-Systems weist das in diesem Abschnitt vorgestellte Problembereichsmodell die folgenden Abweichungen auf:

- Das Problembereichsmodell des E-Mail-Systems enthält neben den Materialien noch weitere Objekte, die als Werkzeuge interpretiert wurden. Im Gegensatz dazu werden Werkzeuge im Problembereichsmodell des Systems zur Pflegedienst und Terminplanung im Krankenhaus nicht berücksichtigt. Dies ergibt sich zwangsläufig aus der Ist-Analyse des Systems zur Pflegedienst- und Terminplanung, die aus Zeitgründen nur oberflächlich durchgeführt werden konnte und keine Werkzeuge enthält. Hier zeigt sich eine Analogie zwischen dem in diesem Kapitel betrachteten System zur Pflegedienst- und Terminplanung und dem System zur Bibliotheksverwaltung. In der Ist-Analyse zum Bibliotheksverwaltungssystem (siehe Abschnitt 3.2) wurden ebenfalls keine Werkzeuge aufgeführt.

  Im Abschnitt 4.4.2 wurde begründet, warum in der Ist-Analyse des E-Mail-Systems im Gegensatz zur Ist-Analyse des Systems zur Bibliotheksverwaltung Werkzeuge aufgeführt sind, obwohl beide Ist-Analysen nur oberflächlich durchgeführt werden konnten. Diese Begründung kann für einen Vergleich der Ist-Analysen des Systems zur Pflegedienst- und Terminplanung und des E-Mail-Systems übernommen werden. So kann bei der Entwicklung des Systems zur Pflegedienst- und Terminplanung nicht auf eine detaillierte Empfehlung über den Aufbau des Systems, in der auch Werkzeuge aufgeführt sind, zurückgegriffen werden. Bei der Modellierung des E-Mail-Systems war eine solche Empfehlung durch X.400 vorhanden.

Nach den Erfahrungen, die bei der Entwicklung des Prototyps für das E-Mail-System gemacht wurden, bietet die Darstellung von Werkzeugen im Problembereichsmodell jedoch keine wesentliche Hilfe bei der Systementwicklung (siehe Abschnitt 4.7). Das Fehlen der Werkzeuge im Problembereichsmodell sollte die weitere Systementwicklung deshalb nicht negativ beeinflussen.

- Während im Problembereichsmodell des Systems zur Bibliotheksverwaltung noch statische Beziehungen zwischen Instanzen dargestellt wurden, soll im Problembereichsmodell des Systems zur Pflegedienst- und Terminplanung auf die Darstellung dieser Beziehungen verzichtet werden. Nachteile für die Implementierung des Systems werden nicht erwartet, da die statischen Beziehungen zwischen Instanzen im Analysemodell aufgeführt werden.

  Die Darstellung von statischen Beziehungen zwischen Instanzen, die bei der Modellierung des E-Mail-Systems noch bei der Erstellung des Problembereichsmodells durchgeführt wurde, wird somit verlagert. Sie wird erst bei der Erstellung des Analysemodells durchgeführt.

Das Problembereichsmodell des betrachteten Pilotsystems ist in der Abbildung 5.5 skizziert. Die modellierten Materialobjekte werden im folgenden beschrieben:

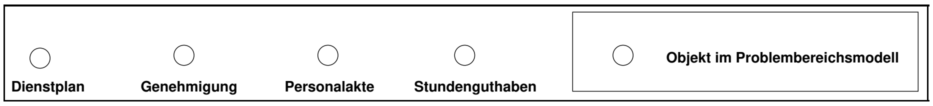

Abbildung 5.5: Objekte im Problembereichsmodell des Pilotsystems.

Das Objekt Dienstplan: Dieses Objekt repräsentiert einen Dienstplan zur Planung des Pflegedienstes einer Station.

Das Objekt Genehmigung: Jeder Dienstplan muß durch die Pflegedienstleitung genehmigt werden. Das Objekt Genehmigung repräsentiert eine Genehmigung eines Dienstplans.

Das Objekt Personalakte: Eine Personalakte enthält die Daten einer im Krankenhaus angestellten Person, wie z.B. einer Pflegekraft.

Das Objekt Stundenguthaben: Für jede Pflegekraft werden die Arbeitsstunden ermittelt, die in einem Monat geleistet wurden. Die für eine Pflegekraft ermittelte Stundenzahl wird mit der Soll-Stundenzahl dieser Pflegekraft verglichen. Die Abweichung der Ist-Stundenzahl von der Soll-Stundenzahl wird als Stundenguthaben gespeichert. Das Stundenguthaben wird als Guthaben in den jeweils nächsten Monat übernommen und dort zur geleisteten Arbeitsstundenzahl hinzuaddiert. Ein negatives Stundenguthaben zeigt an, daß die geleistete Arbeitsstundenzahl unterhalb der Soll-Stundenzahl liegt.

Wenn anstelle der Objekte die Klassen der Objekte betrachtet werden, entspricht das hier erstellte Problembereichsmodell dem im Abschnitt 2.2 vorgestellten fachlichen Klassenentwurf. Bei dem betrachteten Fallbeispiel werden jedoch keine Vererbungsbeziehungen benötigt, mit deren Hilfe bei umfangreicheren Systemen Hierarchien von Materialklassen erstellt werden.

5.4.2 Das Analysemodell

Bei der Erstellung des Analysemodells des Pilotsystems werden die Materialobjekte des Problembereichsmodells als Entitätsobjekte übernommen. Zusätzlich werden im Analysemodell auch die

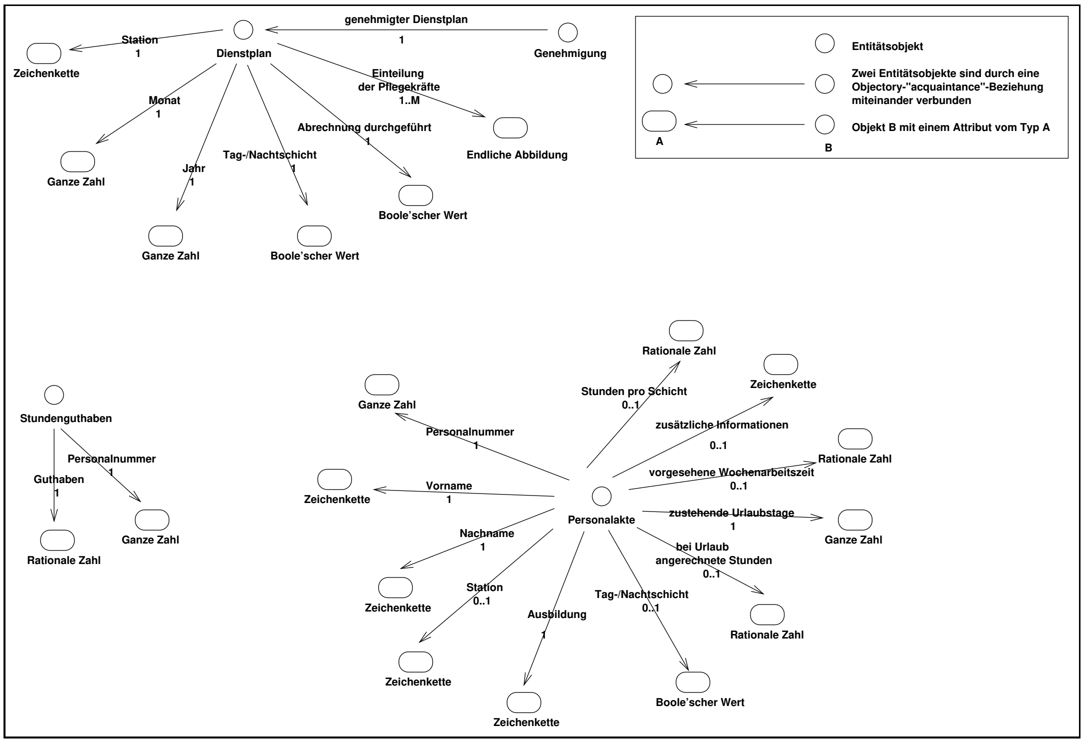

Abbildung 5.6: Materialien im Analysemodell des Pilotsystems.

statischen Beziehungen zwischen Materialobjekten und die Attribute dieser Objekte betrachtet. Die Modellierung der Materialien im Analysemodell ist in der Abbildung 5.6 skizziert.

Zwischen den Materialobjekten ist hier genau eine statische Beziehung modelliert. Dabei handelt es sich um die „acquaintance"-Beziehung zwischen dem Objekt Genehmigung und dem Objekt Dienstplan. Jedes Objekt Genehmigung repräsentiert die Genehmigung eines Dienstplans durch die Pflegedienstleitung und ist durch die Beziehung genehmigter Dienstplan mit genau einem Objekt Dienstplan verbunden, das den Dienstplan repräsentiert, auf den sich die Genehmigung bezieht. Zusätzlich können Objekte Dienstplan existieren, die nicht mit einem Objekt Genehmigung verbunden sind. Diese Objekte Dienstplan repräsentieren dann Dienstpläne, die noch nicht von der Pflegedienstleitung genehmigt wurden.

Die Attribute, die in der Abbildung 5.6 modelliert sind, werden durch die Beschriftungen der Kanten zwischen den Objekten und den jeweiligen Attributen charakterisiert. Bei der Betrachtung des Materialobjektes Personalakte fällt auf, das hier einige Attribute zwingend vorgeschrieben sind (1-Beziehungen), während andere Attribute optional sind (0..1-Beziehungen). Der Grund für diese Unterscheidung ist nicht direkt ersichtlich und soll deshalb kurz erläutert werden: Personalakten können sich auf sämtliche Personen beziehen, die im Krankenhaus arbeiten. Dabei können die Informationen, die zu den Angehörigen verschiedener Personengruppen gespeichert werden müssen, unterschiedlich sein. Beispielsweise wird den Pflegekräften im Krankenhaus eine Station zugeordnet, auf der diese arbeiten. Einem Angestellten der Krankenhausverwaltung kann im Gegensatz dazu üblicherweise keine Station zugeordnet werden. Attribute, die Informationen enthalten, die nicht immer für alle im Krankenhaus beschäftigten Personen nötig sind, sind in dem vorgestellten Modell optional. Die restlichen Attribute sind vorgeschrieben.

Das Materialobjekt Personalakte wird hier nur mit den Attributen versehen, die für das betrachtete Beispiel benötigt werden. Bei einem realen System, das in der Regel einen größeren Umfang als das hier betrachtete System hätte, wären noch weitere Attribute, wie z.B. die Angabe der Adresse der Beschäftigten, sinnvoll. Die anderen Materialobjekte besitzen keine optionalen Attribute. Alle Attribute sind hier entweder genau einmal (1-Beziehungen) oder mindestens einmal (1..M-Beziehung) vorhanden.

Neben den Materialien wird auch die Struktur des vollständigen Systems im Analysemodell dargestellt. Die Modellierung des Pilotsystems ist in der Abbildung 5.7 skizziert. Um die Abbildung übersichtlicher zu gestalten, sind die Attribute und die statischen Beziehungen zwischen Instanzen aus der Abbildung 5.6 hier nicht mehr aufgeführt.

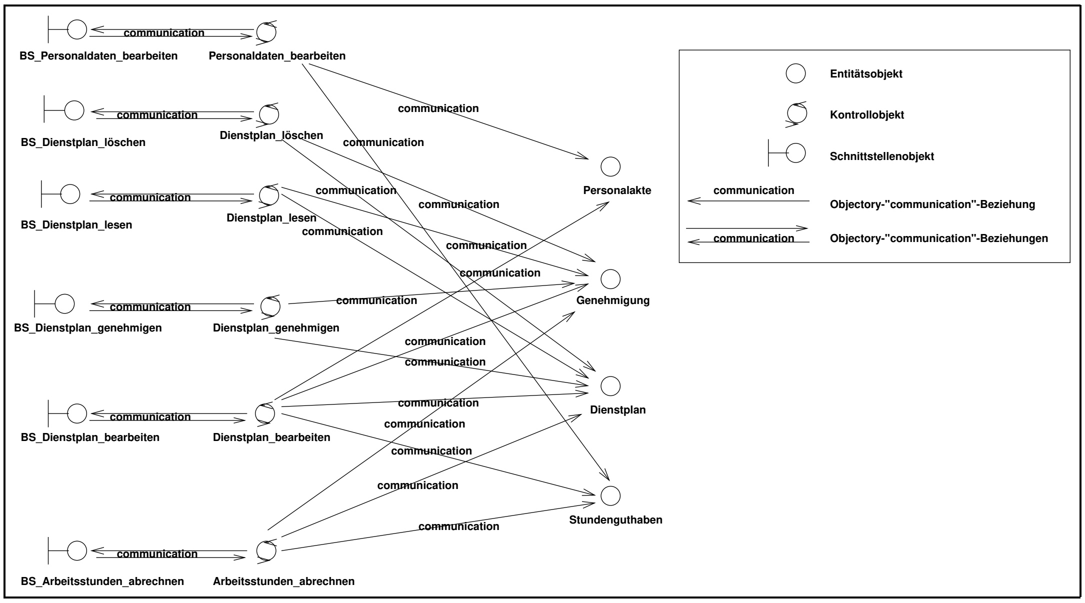

Abbildung 5.7: Analysemodell des Pilotsystems ohne Berücksichtigung von Attributen und statischen Beziehungen zwischen Instanzen.

Bei der Modellierung des Pilotsystems wird analog zum Analysemodell des E-Mail-Systems für jeden Anwendungsfall genau ein Kontrollobjekt modelliert (siehe Abschnitt 4.5). Die Zuordnung von Kontrollobjekten zu Anwendungsfällen ist in der Tabelle 5.1 dargestellt. Jedem Kontrollobjekt ist durch „communication"-Beziehungen ein Schnittstellenobjekt zugeordnet, über das Interaktionen zwischen dem modellierten System und den Benutzern abgewickelt werden. Eine solche Kombination aus einem Kontrollobjekt und einem Schnittstellenobjekt repräsentiert jeweils immer genau ein Werkzeug, mit dessen Hilfe Materialien bearbeitet oder auf ihren Zustand hin überprüft werden können (siehe Abschnitt 2.2). Beispielsweise besteht ein Werkzeug zum Löschen eines Dienstplans aus dem Kontrollobjekt Dienstplan_löschen und aus dem Schnittstellenobjekt BS_Dienstplan_löschen.

Im Gegensatz zur Modellierung des Benutzeragenten des E-Mail-Systems (siehe Abbildung 4.12 im Abschnitt 4.5) sind in der Abbildung 5.7 mehrere Werkzeuge dargestellt. Bei der Modellierung der „communication"-Beziehungen werden die Regeln eingehalten, die bereits bei der Beschreibung des Analysemodells des E-Mail-Systems vorgestellt wurden (siehe Abschnitt 4.5, Seite 74).

Eine Übersicht über alle Werkzeuge des Pilotsystems gibt die Tabelle 5.2. Mit Hilfe der Werkzeuge werden die in der Abbildung 5.6 dargestellten Materialien Personalakte, Genehmigung, Dienstplan und Stundenguthaben manipuliert oder auf ihren Zustand überprüft. Die Materialien werden mit Hilfe von Entitätsobjekten dargestellt. Der Zugriff von Werkzeugen auf Materialien wird im Analysemodell durch „communication"-Beziehungen zwischen Kontrollobjekten und Entitätsobjekten modelliert. Gemäß den Vorschriften von Objectory sind die „communication"-Beziehungen zwischen Kontrollobjekten und Entitätsobjekten immer einseitig von den den Kontrollobjekten zu den Entitätsobjekten hin gerichtet.

Anwendungsfall	Kontrollobjekt
Arbeitsstunden abrechnen	Arbeitsstunden_abrechnen
Dienstplan bearbeiten	Dienstplan_bearbeiten
Dienstplan genehmigen	Dienstplan_genehmigen
Dienstplan lesen	Dienstplan_lesen
Dienstplan löschen	Dienstplan_löschen
Personaldaten bearbeiten	Personaldaten_bearbeiten

Tabelle 5.1: Zuordnung von Kontrollobjekten zu Anwendungsfällen.

Analysemodellobjekte	Beschreibung des Werkzeugs
BS_Personaldaten_bearbeiten Personaldaten_bearbeiten	Eine Personalakte kann neu angelegt, modifiziert oder gelöscht werden.
BS_Dienstplan_löschen Dienstplan_löschen	Ein Dienstplan kann gelöscht werden.
BS_Dienstplan_lesen Dienstplan_lesen	Ein Dienstplan kann gelesen werden.
BS_Dienstplan_genemigen Dienstplan_genehmigen	Ein Dienstplan kann genehmigt werden. Wenn der Dienstplan nicht alle Angaben enthält, die für eine Abrechnung der geleisteten Arbeitsstunden nötig sind, oder wenn nicht alle Schichten vorschriftsmäßig besetzt sind, wird ein Warnhinweis ausgegeben. Der Benutzer kann die Genehmigung des gewählten Dienstplans nach der Ausgabe dieses Warnhinweises noch rückgängig machen.
BS_Dienstplan_bearbeiten Dienstplan_bearbeiten	Ein Dienstplan kann erstellt werden. Bereits erstellte Dienstpläne können modifiziert werden.
BS_Arbeitsstunden_abrechnen Arbeitsstunden_abrechnen	Die geleisteten Arbeitsstunden der Pflegekräfte werden berechnet und im Dienstplan eingetragen. Zusätzlich werden die Stundenguthaben der Pflegekräfte angepaßt. Bei Bedarf können die berechneten Stundenguthaben noch manuell geändert werden.

Tabelle 5.2: Beschreibung der Werkzeuge des Pilotsystems und Angabe beteiligter Analysemodellobjekte.

5.4.3 Das Entwurfsmodell

Das Entwurfsmodell des Pilotsystems ist in der Abbildung 5.8 skizziert.

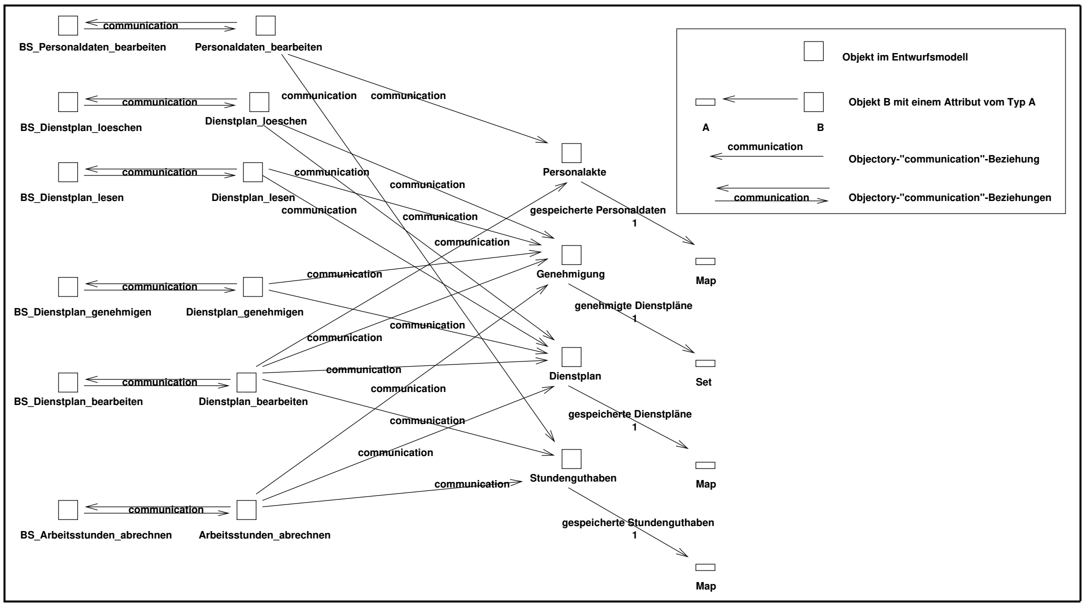

Abbildung 5.8: Objekte und Beziehungen im Entwurfsmodell des Pilotsystems.

Die bereits bei der Beschreibung des Analysemodells vorgestellte Struktur (siehe Abbildung 5.7 im Abschnitt 5.4.2) bleibt im Entwurfsmodell weitgehend erhalten. Durch die Berücksichtigung der Einschränkungen, die durch die verwendete Soft- und Hardware vorgegeben sind, ergeben sich jedoch auch einige Änderungen, die im folgenden vorgestellt werden:

- Wie beim Entwurfsmodell des E-Mail-Systems kommen in den Namen der Entwurfsmodellobjekte keine Umlaute mehr vor, weil Umlaute auf der Tastatur der Computer, die zur Programmierung zur Verfügung stehen, nicht vorhanden sind (siehe auch Abschnitt 4.6.1).

- Die Namen der im Analysemodell vorgestellten Materialobjekte bleiben erhalten. Die Bedeutungen dieser Objekte ändern sich jedoch leicht. Dies soll zunächst exemplarisch anhand des Objektes Personalakte beschrieben werden:

  Während das Analysemodellobjekt Personalakte genau eine Personakte einer Person repräsentiert, repräsentiert das Entwurfsmodellobjekt Personalakte sämtliche im Krankenhaus vorhandenen Personalakten. Diese Abweichung vom Analysemodell ist nötig, weil es sich bei Personalakten um Daten handelt, die persistent gemacht werden. Dabei muß die durch ProSet gemachte Vorgabe, daß Namen von P-File-Einträgen, die in einem ProSet-Programm benötigt werden, vor der Übersetzung des Programms feststehen müssen, berücksichtigt werden. Vor der Übersetzung der ProSet-Programme, die auf die in den Personalakten gespeicherten Daten zugreifen, ist jedoch nicht bekannt, für welche und für wieviele Personen Personalakten angelegt werden müssen. Ähnliche Probleme wurden bereits bei der Modellierung des Systems zur Bibliotheksverwaltung und bei der Modellierung des E-Mail-Systems diskutiert (siehe Abschnitte 3.6 und 4.6.2).

  Eine Lösungsmöglichkeit bestünde darin, für jede Personalakte ein eigenes P-File zu erzeugen. Dann könnte in jedem dieser P-Files genau ein Eintrag (z.B. eine ProSet-Modulinstanz) mit dem Namen Personalakte gespeichert werden, der die Daten genau einer Personalakte einer Person enthält. Diese Lösung setzte jedoch voraus, daß der Anwender zunächst mit einem externen Werkzeug, wie z.B. dem xpft, ein neues P-File erzeugt, wenn er die Daten eines neuen Angestellten eingeben möchte. Dieses P-File sollte er auch wieder entfernen, wenn die Daten des Angestellten nicht mehr benötigt werden. Für ein praktisches Arbeiten mit dem zu entwickelnden System erscheint diese Lösung als zu umständlich.

Deshalb ist im Entwurfsmodell nur genau ein Objekt Personalakte, das die Daten aller Personalakten enthält, vorgesehen. Die Personalakten der verschiedenen Personen sollen im Objekt Personalakte mit Hilfe einer endlichen Abbildung gespeichert werden. Anhand der Personalnummern der einzelnen Personen können die in dieser Abbildung gespeicherten verschiedenen Personalakten eindeutig identifiziert werden.

Zum Entwurfsmodellobjekt Personalakte ist also nur noch genau ein Attribut modelliert. Der Typ Map soll andeuten, daß es sich bei diesem Attribut um eine endliche Abbildung handelt. Die Modellierung der Attribute im Analysemodell (siehe Abbildung 5.6 im Abschnitt 5.4.2) ist jedoch nicht überflüssig. Diese Attribute stellen auch weiterhin die Daten dar, die zu einer Personalakte gespeichert werden. Hierauf wird bei der Beschreibung der Implementierung im Abschnitt 5.5.1 noch eingegangen.

Die Speicherung sämtlicher Personalakten in einer Datenstruktur besitzt jedoch auch Nachteile, die in Kauf genommen werden müssen. So sind die Möglichkeiten zum parallelen Datenzugriff eingeschränkt. Die Daten sämtlicher Personen sind für einen Zugriff gesperrt, solange auf die Daten einer Person schreibend zugegriffen wird. Daten einer beliebigen Person können erst dann durch eine Instanz eines Programms manipuliert werden, wenn keine Daten einer beliebigen (evtl. anderen) Person mehr von einer anderen Programminstanz gelesen werden. Außerdem können Zeit- und Speicherplatzprobleme auftreten, weil immer die Daten sämtlicher Personen geladen werden müssen, wenn auf die Daten einer Person zugegriffen werden soll.

Die hier beschriebenen Schwierigkeiten bei der Modellierung des Objektes Personalakte gelten analog auch für die Objekte Dienstplan und Stundenguthaben. Die Lösung, die für das Objekt Personalakte beschrieben wurde, wird auch für die Objekte Dienstplan und Stundenguthaben übernommen.

Das Objekt Genehmigung repräsentiert im Analysemodell die Genehmigung genau eines Dienstplans. Im Entwurfsmodell repräsentiert das Objekt Genehmigung im Gegensatz dazu die Genehmigungen mehrerer Dienstpläne. Die „acquaintance“-Beziehung genehmigter Dienstplan, die im Analysemodell die Objekte Genehmigung und Dienstplan verbindet (siehe Abbildung 5.6 im Abschnitt 5.4.2), entfällt im Entwurfsmodell, weil eine direkte Abbildung statischer Beziehungen zwischen Entwurfsmodellobjekten auf ein ProSet-Programm, z.B. durch die Möglichkeit Zeigervariablen zu nutzen, nicht möglich ist. Stattdessen wird zum Entwurfsmodellobjekt Genehmigung ein Attribut modelliert, in dem zu jedem genehmigten Dienstplan ein Datum gespeichert wird, mit dem dieser Dienstplan eindeutig identifiziert werden kann. Auf die Zusammensetzung dieses Datums wird bei der Beschreibung der Implementierung im Abschnitt 5.5.1 eingegangen.

Zusätzlich zur Darstellung von Objekten, Attributen und Beziehungen zwischen Objekten enthält das Entwurfsmodell noch Interaktionsdiagramme, in denen der Austausch der Stimuli zwischen den Objekten für jeden Anwendungsfall beschrieben wird. Der Aufbau der Interaktionsdiagramme entspricht dem Aufbau, der bereits bei der Beschreibung des Entwurfsmodells des E-Mail-Systems vorgestellt wurde (siehe Abschnitt 4.6.1). Exemplarisch ist in der Abbildung 5.9 das Interaktionsdiagramm zum Anwendungsfall Personaldaten bearbeiten aufgeführt. Für die Interaktionsdiagramme zu den restlichen Anwendungsfällen wird wiederum auf das Zusatzdokument verwiesen.

Zusätzlich zu den Interaktionsdiagrammen ermöglicht Objectory die Erstellung von Zustands-Übergangsdiagrammen. Zustands-Übergangsdiagramme werden bei Jacobson et al. vorgestellt [JCJÖ93, S. 233ff.]. Sie dienen zur Beschreibung des Verhaltens *zustandskontrollierter* Objekte. Ein zustandskontrolliertes Objekt ist dadurch gekennzeichnet, daß ein durch dieses Objekt empfangener Stimulus die Ausführung unterschiedlicher Operationen auslösen kann. Welche Operation

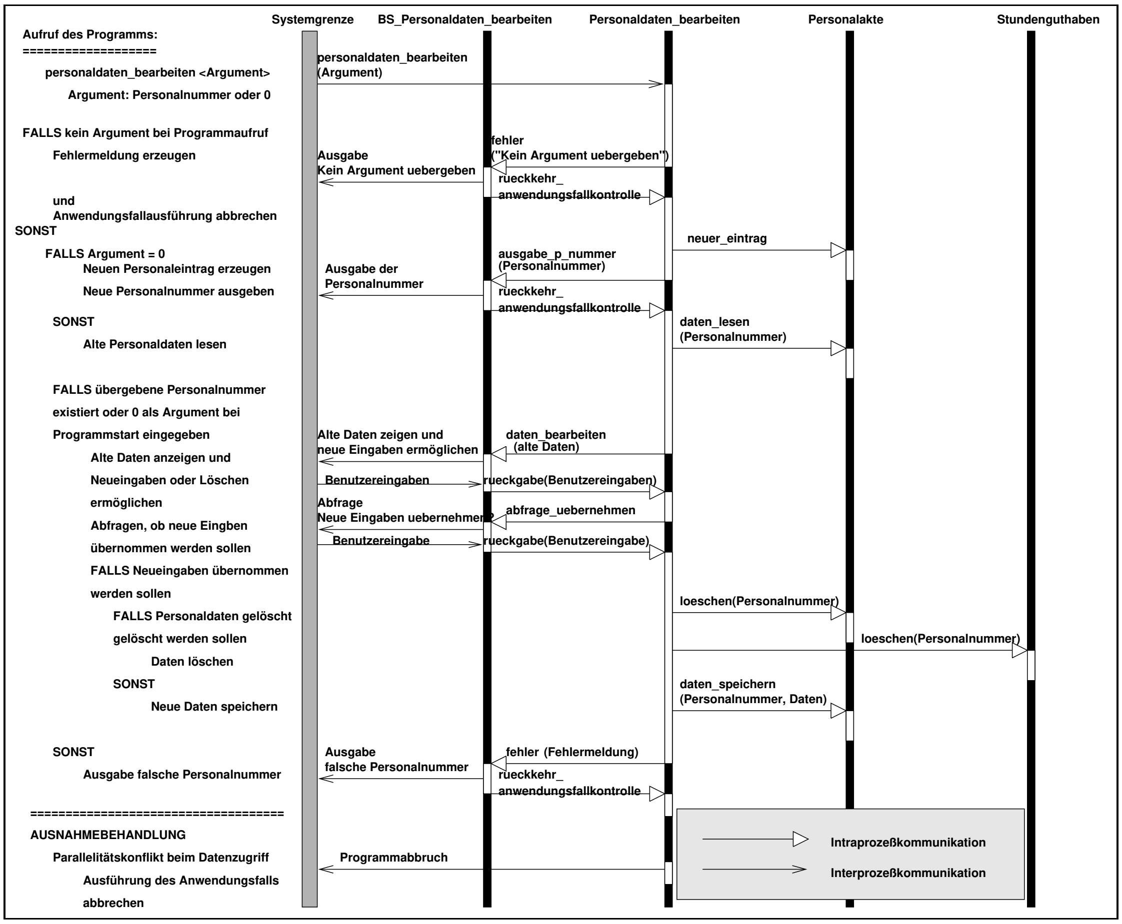

Abbildung 5.9: Interaktionsdiagramm für den Anwendungsfall Personaldaten bearbeiten.

ausgelöst wird hängt von einem in diesem Objekt gespeicherten Zustand ab [JCJÖ93, S. 239f.]. Da bei dem hier betrachteten Entwurfsmodell die Stimuli, die von Objekten empfangen werden, immer die gleichen Operationen auslösen, sind die hier modellierten Objekte nicht zustandskontrolliert. Zustands-Übergangsdiagramme werden deshalb nicht benötigt.

5.5 Implementierung und Test des Pilotsystems

Zu einer vollständigen Systementwicklung gehören in Objectory noch das Implementierungs- und das Testmodell, in denen der Quellcode der Implementierung bzw. die Spezifikation und die Ergebnisse der Tests dargestellt werden (siehe Abschnitt 2.1). Die Grundlagen zur Erstellung dieser Modelle für das betrachtete Fallbeispiel sollen im folgenden in den Abschnitten 5.5.1 und 5.5.2 beschrieben werden.

5.5.1 Implementierung

Im ersten Teil dieses Abschnitts wird auf die Implementierung des Pilotsystems in der Prototyping-Sprache ProSet eingegangen. Bei der Entwicklung eines Systems, das in der Praxis tatsächlich eingesetzt werden soll, ist es sinnvoll, das ProSet-Programm in eine andere effizientere, aber schwerer handhabbare Programmiersprache, wie z.B. C, zu überführen. Hierauf soll im zweiten Teil dieses Abschnitts kurz eingegangen werden.

Abbildung der Objectory-Modelle auf ProSet

Bei der Abbildung des Entwurfsmodells auf eine ProSet-Implementierung des Pilotsystems wird auf die Grundlagen zurückgegriffen, die bereits bei der Beschreibung der Implementierung des Prototyps des E-Mail-Systems dargestellt wurden (siehe Abschnitt 4.6). Sie sollen im folgenden kurz wiederholt werden:

- Für jede Klasse eines Entwurfsmodellobjektes wird ein ProSet-Modul implementiert.
- Entwurfsmodellobjekte werden durch ProSet-Modulinstanzen dargestellt.
- Der Austausch von Stimuli zwischen Entwurfsmodellobjekten wird im ProSet-Programm durch Prozeduraufrufe und durch return-Befehle nachgebildet.

Im Gegensatz zur Erstellung des Prototyps des E-Mail-Systems wird bei der Implementierung des Pilotsystems des in diesem Kapitel betrachteten Fallbeispiels nun wieder stärkerer Gebrauch von persistenten Daten gemacht. Für die Entwurfsmodellobjekte Dienstplan, Genehmigung, Personalakte und Stundenguthaben werden ProSet-Modulinstanzen erzeugt und persistent gemacht. Diese ProSet-Modulinstanzen speichern die Daten, die zu den jeweiligen Materialien benötigt werden und stellen Operationen zum Zugriff auf diese Daten bereit. Die Entscheidung, die Daten für die Materialien und die Zugriffsoperationen im selben Objekt zusammenzufassen, wurde aufgrund der Erfahrungen, die bei der Modellierung des Systems zur Bibliotheksverwaltung gemacht wurden, getroffen (siehe Abschnitt 3.5.1, Seite 47 und Abschnitt 3.6). Bei der Modellierung des Systems zur Bibliotheksverwaltung wurden Daten und Zugriffsoperationen getrennt.

Die benötigten persistenten Modulinstanzen werden in verschiedenen P-Files gespeichert, die im folgenden beschrieben werden:

- Für jede Station im Krankenhaus wird ein eigenes P-File mit dem Namen dieser Station erzeugt. In jedem dieser P-Files werden jeweils eine Modulinstanz zu den Entwurfsmodellobjekten Dienstplan und Genehmigung gespeichert. Diese Modulinstanzen haben die Namen dienstplan (für das Entwurfsmodellobjekt Dienstplan) und genehmigung (für das Entwurfsmodellobjekt Genehmigung). Die Namen der ProSet-Modulinstanzen werden klein geschrieben, weil im ProSet-Programm im Gegensatz zu den Objectory-Modellen bei der Namensgebung zwischen Klassen und Instanzen unterschieden werden muß. Namen von Moduln, die im ProSet-Programm dazu dienen, Klassen darzustellen, werden bei diesem Fallbeispiel groß geschrieben.

  Beispiel: Für ein Krankenhaus, das aus den drei Stationen Station1, Station2 und Station3 besteht, werden drei P-Files eingerichtet. Diese P-Files werden mit Station1, Station2 und Station3 bezeichnet. In jedem dieser P-Files wird eine ProSet-Modulinstanz dienstplan und eine ProSet-Modulinstanz genehmigung gespeichert.

- Zusätzlich wird ein weiteres P-File erstellt, das Krankenhausdaten genannt wird. In dieses P-File werden die ProSet-Modulinstanzen personalakte und stundenguthaben für die Entwurfsmodellobjekte Personalakte und Stundenguthaben aufgenommen. Diese personenbezogenen Daten werden nicht in den P-Files gespeichert, die den Stationen im Krankenhaus zugeordnet sind, weil Pflegekräfte die Station, auf der sie arbeiten, wechseln können.

Die benötigten P-Files müssen mit einem externen Werkzeug, wie z.B. dem xpft, angelegt werden. Um die benötigten Moduln zu initialisieren und um die Instanzen persistent zu machen, sind die in der Tabelle 5.3 aufgeführten Hilfsprogramme implementiert.

Obwohl die Attribute, die im Analysemodell zu den Materialobjekten modelliert wurden, im Entwurfsmodell nicht mehr erkennbar sind (siehe Abbildungen 5.6 und 5.8 in den Abschnitten 5.4.2 und 5.4.3), ist die Modellierung der Attribute im Analysemodell für die ProSet-Implementierung des Systems nicht überflüssig. Dies soll zunächst exemplarisch anhand des Objektes Personalakte erläutert werden:

Die Klasse des Entwurfsmodellobjektes Personalakte wird durch den ProSet-Modul Personalakte repräsentiert. Das Entwurfsmodellobjekt Personalakte wird durch die (persistente) ProSet-Modulinstanz personalakte dargestellt. Die endliche Abbildung, die als Attribut zum Entwurfsmodellobjekt Personalakte modelliert ist (siehe Abbildung 5.8 im Abschnitt 5.4.3), wird unter dem

Programm	Argument	bearbeiteter Modul
dienstplan_initialisieren	Stationsname	Dienstplan
genehmigung_initialisieren	Stationsname	Genehmigung
personalakte_initialisieren	"Krankenhausdaten"	Personalakte
stundenguthaben_initialisieren	"Krankenhausdaten"	Stundenguthaben

Tabelle 5.3: Hilfsprogramme, um Modulinstanzen zu erzeugen und persistent zu machen.

Variablennamen gepeicherte_personaldaten direkt als Datenstruktur in den ProSet-Modul Personalakte übernommen und steht somit in der ProSet-Modulinstanz personalakte zur Aufnahme von Daten zur Verfügung.

In der endlichen Abbildung gespeicherte_personaldaten werden sämtliche benötigte Personalakten gespeichert. Auf die einzelnen Personalakten kann über das im Analysemodell als Attribut modellierte Schlüsselelement Personalnummer zugegriffen werden (siehe Abbildung 5.6 im Abschnitt 5.4.2). Die zu den benötigten einzelnen Personalakten gehörenden Daten werden selbst wieder mit Hilfe einer endlichen Abbildung gespeichert. Die endliche Abbildung gespeicherte_personaldaten hat also das folgende Aussehen:

$$gespeicherte\_personaldaten : \begin{cases} \textsf{integer} \longrightarrow \textsf{map} \\ personalnummer \mapsto einzelne\_personalakte. \end{cases}$$

Die Variable personalnummer enthält die Personalnummer der Person, auf deren Personalakte zugegriffen werden soll.

Die endliche Abbildung einzelne_personalakte enthält die Daten genau einer Personalakte einer Person. Diese Daten können aus dem Analysemodell entnommen werden. Jedes Attribut, das zum Analysemodellobjekt Personalakte modelliert ist (siehe Abbildung 5.6 im Abschnitt 5.4.2), entspricht einem Eintrag in der Abbildung einzelne_personalakte. Die einzige Ausnahme hiervon stellt das Attribut Personalnummer dar. Die Personalnummer wird bereits in der Abbildung gespeicherte_personaldaten verwendet, um die einzelnen Personalakten identifizieren zu können (siehe oben). In der endlichen Abbildung einzelne_personalakte wird die Personalnummer nicht mehr benötigt.

Bei der endlichen Abbildung einzelne_personalakte bietet es sich an, über Atome auf die Daten zuzugreifen, die zu einer gewählten Personalakte gespeichert sind, wie es in der Tabelle 5.4 skizziert ist (die Einträge in der Spalte x sind Atome).

x	$einzelne\_personalakte\ (x)$
vorname	Vorname der Person
nachname	Nachname der Person
station	Bezeichnung der Station oder om, falls kein Eintrag
ausbildung	Bezeichnung der Ausbildung
tagschicht	true = Tagschicht, false = Nachtschicht, om = kein Eintrag
arbeiturlaub	bei Urlaub angerechnete Stunden oder om, falls kein Eintrag
urlaub	zustehende Urlaubstage
arbeitszeit	vorgesehene Wochenarbeitszeit oder om, falls kein Eintrag
sonstiges	zusätzliche Informationen oder om, falls kein Eintrag
stundenschicht	Anzahl Arbeitsstunden pro Schicht oder om, falls kein Eintrag

Tabelle 5.4: Einträge für eine einzelne Personalakte einer Person.

Bei der verwendeten ProSet-Compiler-Version 0.6 führt die Verwendung von Atomen innerhalb einer persistenten ProSet-Modulinstanz jedoch noch zu einem Fehler bei der Programmausführung (Fehlermeldung: Exception type_mismatch could not be handled!). Die in der Tabelle 5.4 beschriebenen Atome, über die auf die Einträge in der Abbildung *einzelne_personalakte* zugegriffen werden soll, werden deshalb in der ProSet-Implementierung durch ganzzahlige Konstanten ersetzt.

Die Implementierung der ProSet-Moduln Stundenguthaben und Dienstplan verläuft analog zur Implementierung des Moduls Personalakte. Als Schlüsselelement zum Zugriff auf die einzelnen

Stundenguthaben der verschiedenen Pflegekräfte dient wiederum die Personalnummer. Die gespeicherten Dienstpläne werden eindeutig anhand eines Tupels identifiziert, das aus dem Monat des Dienstplans, dem Jahr des Dienstplans und der Angabe, ob es sich bei dem Plan um die Einteilung der Tagschichten oder um die Einteilung der Nachtschichten handelt, besteht (weil die Modulinstanzen für die Dienstpläne verschiedener Stationen in verschiedenen P-Files gespeichert werden ist eine zusätzliche Angabe der Station, auf die sich ein Dienstplan bezieht, nicht nötig).

Um das Entwurfsmodellobjekt Genehmigung auf eine ProSet-Implementierung abzubilden, wird wiederum zunächst ein ProSet-Modul implementiert, der die Klasse des Entwurfsmodellobjektes Genehmigung repräsentiert. Die (endliche) Menge, die als Attribut zum Entwurfsmodellobjekt Genehmigung modelliert wurde, wird als Datenstruktur direkt in diesen ProSet-Modul übernommen und kann in Instanzen des Moduls zur Aufnahme von Daten genutzt werden. In der Menge werden Daten gespeichert, mit denen die genehmigten Dienstpläne eindeutig identifiziert werden können. Bei diesen Daten handelt es sich um die gleichen Tupel, die bereits die Dienstpläne eindeutig identifizieren (siehe oben). Ein Tupel besteht aus dem Monat des Dienstplans, dem Jahr des Dienstplans und der Angabe, ob es sich bei dem Dienstplan um die Einteilung der Pflegekräfte für die Tagschichten oder für die Nachtschichten handelt.

Da die durch ProSet vorgegebenen Einschränkungen, wie z.B. Schwierigkeiten bei der Abbildung von statischen Beziehungen zwischen Instanzen (siehe Beschreibung des Entwurfsmodellobjektes Genehmigung im Abschnitt 5.4.3, Seite 115), bereits bei der Erstellung des Entwurfsmodells berücksichtigt wurden, läßt sich die Umsetzung des Entwurfsmodells auf eine ProSet-Implementierung relativ einfach mit Hilfe der entworfenen Interaktionsdiagramme durchführen.

Während die Materialobjekte Dienstplan, Genehmigung, Personalakte und Stundenguthaben in P-Files gespeichert werden, dienen die restlichen Objekte dazu, die für den Benutzer sichtbaren Werkzeuge zu implementieren. Die Namen der implementierten Werkzeuge, die Argumente, die beim Aufruf der Werkzeuge übergeben werden müssen und die beteiligten Entwurfsmodellobjekte sind in der Tabelle 5.5 aufgeführt.

Programmname	Argument beim Programmaufruf	Entwurfsmodellobjekte
personaldaten_bearbeiten	Personalnummer oder 0, falls neue Personalakte angelegt werden soll	BS_Personaldaten_bearbeiten Personaldaten_bearbeiten
dienstplan_loeschen	Stationsbezeichnung	BS_Dienstplan_loeschen Dienstplan_loeschen
dienstplan_genehmigen	Stationsbezeichnung	BS_Dienstplan_genehmigen Dienstplan_genehmigen
dienstplan_lesen	Stationsbezeichnung	BS_Dienstplan_lesen Dienstplan_lesen
dienstplan_bearbeiten	Stationsbezeichnung	BS_Dienstplan_bearbeiten Dienstplan_bearbeiten
arbeitsstunden_abrechnen	Stationsbezeichnung	BS_Arbeitsstunden_abrechnen Arbeitsstunden_abrechnen

Tabelle 5.5: Werkzeuge des Pilotsystems und beteiligte Entwurfsmodellobjekte.

Einsatz anderer Programmiersprachen

In dieser Diplomarbeit wird die Erstellung des Pilotsystems mit der Implementierung und dem Test (Abschnitt 5.5.2) der zugehörigen ProSet-Programme beendet. Bei einer Entwicklung eines Systems, das in der Praxis eingesetzt werden soll, ist es jedoch sinnvoll, die ProSet-Programme in eine andere Programmiersprache zu transformieren. Für eine solche Transformation sprechen die folgenden Argumente:

- ProSet stellt zwar komfortable Operationen und Datenstrukuren zur Verfügung, die eine schnelle Programmerstellung unterstützen. ProSet-Programme benötigen im Vergleich zu

Programmen mit gleicher Funktionalität, die in weniger komfortablen Programmiersprachen, wie z.B. C, implementiert sind, jedoch üblicherweise mehr Speicherplatz und werden langsamer ausgeführt.

- Die Möglichkeiten zur Ausgabe von Daten auf dem Bildschirm und für Eingaben des Benutzers sind in ProSet stark eingeschränkt. Die Integration einer graphischen Benutzeroberfläche ist bei einem Programm, das in einer anderen Programmiersprache erstellt wird, evtl. einfacher.

- Die Verwaltung großer Datenmengen mit Hilfe des ProSet-Persistenzmechanismus ist problematisch, weil oft wesentlich mehr persistente Daten von der Datenbank, die für den Persistenzmechanismus genutzt wird, geladen werden müssen als benötigt werden. Für das in diesem Kapitel betrachtete System wurde dies im Abschnitt 5.4.3 erläutert (Seite 114f.). Dadurch ist zum einen die Zeitdauer für einen Datenbankzugriff relativ hoch, zum anderen sind immer mehr Daten für einen zweiten parallelen Zugriff gesperrt, als eigentlich erforderlich wäre.

  Es ist deshalb sinnvoll, ein Datenbanksystem zu nutzen, mit dem feingranularer auf einzelne Einträge zugegriffen werden kann, als dies bei einem ProSet-Programm der Fall ist.

- Die in ProSet vorgesehene Möglichkeit zur Ausnahmebehandlung funktioniert bei den bislang vorhandenen Compilerversionen nicht im Zusammenhang mit persistenten Variablen, die lokal in Prozeduren deklariert werden (siehe Abschnitt 3.6). Auftretende Ausnahmen können daher oft nicht abgefangen werden.

Die Transformation in eine andere Programmiersprache wird durch ProSet dadurch unterstützt, daß zunächst innerhalb des ProSet-Programms schrittweise Transformationen des Programmcodes von höheren programmiersprachlichen Ebenen auf niedrigere Ebenen durchgeführt werden können. Beispielsweise kann eine Iteration über Tupel oder Mengen schrittweise so verändert werden, bis sie einer „for-Schleife“ eines C-Programms so weit angenähert ist, daß der Aufbau der entsprechenden C-Schleife direkt aus der ProSet-Implementierung ersichtlich wird.

5.5.2 Test

Im folgenden soll ein Überblick über die bei diesem Fallbeispiel durchgeführten Tests gegeben werden. Diese Beschreibungen können als Grundlage für die Erstellung eines Testmodells angesehen werden, in dem nach Jacobson et al. die Spezifikation und die Ergebnisse der Tests eines implementierten Systems dokumentiert werden [JCJÖ93, S. 114]. Da eine Transformation des ProSet-Quellcodes in den Code einer anderen Programmiersprache im Rahmen dieser Diplomarbeit nicht mehr durchgeführt wird, beziehen sich die durchgeführten Tests auf die in ProSet implementierten Programme.

Zur Planung und zur Durchführung von Tests geben Jacobson et al. ausführliche Hinweise [JCJÖ93, S. 315ff.]. Im Rahmen dieser Diplomarbeit können aus Zeitgründen nur einige der von Jacobson et al. vorgeschlagenen Tests betrachtet werden. Dennoch sollten die durchgeführten Tests ausreichen, um eine stabile Version der ProSet-Implementierung des entwickelten Pilotsystems zu erhalten, da der Umfang des hier betrachteten Fallbeispiels nicht mit dem Umfang eines Projektes, auf das sich die Vorschläge von Jacobson et al. beziehen, vergleichbar ist. So beschreiben Jacobson et al., daß Objectory bis zum Sommer 1993 für Projekte mit einem Umfang von drei bis 50 Mann-Jahren eingesetzt wurden [JCJÖ93, S. 436]. Das in der Diplomarbeit betrachtete Beispiel hat einen wesentlich geringeren Umfang und sollte daher auch weniger potentielle Fehlerquellen besitzen.

Im folgenden wird ein Überblick über die durchgeführten Tests gegeben:

Modultest: Der durchgeführte Modultest kann drei in verschiedene Phasen aufgeteilt werden. Diese Phasen werden im folgenden vorgestellt:

Phase 1: Zunächst wird für jeden Modul, der Prozeduren für die Kommunikation mit dem Benutzer enthält, ein Programm erstellt, mit dem jede Prozedur, die von dem zugehörigen Modul exportiert wird, aufgerufen und getestet wird. Bei den in dieser Phase getesteten Moduln handelt es sich um die Moduln BS_Arbeitsstunden_abrechnen, BS_Dienstplan_bearbeiten, BS_Dienstplan_lesen, BS_Dienstplan_genehmigen, BS_Dienstplan_loeschen und BS_Personaldaten_bearbeiten.

Phase 2: Im zweiten Schritt werden die Moduln getestet, die die Klassen der Materialobjekte repräsentieren. Hierbei handelt es sich um die Moduln Dienstplan, Genehmigung, Stundenguthaben und Personalakte. Das Testen verläuft analog zum Testen der Benutzerschnittstellenmoduln. Für jeden zu testenden Modul wird ein Programm implementiert, mit dem die Prozeduren des betrachteten Moduls, die von außen aufgerufen werden können, unabhängig voneinander getestet werden. Dabei wird berücksichtigt, daß in den Instanzen der Moduln zur Darstellung der Materialklassen ein Zustand gespeichert wird. Ein Prozeduraufruf kann, abhängig von den vorher ausgeführten Tests, unterschiedliche Ergebnisse liefern.

Phase 3: Im nächsten Schritt werden die Moduln, in denen die Ereignisflüsse der einzelnen Anwendungsfälle kontrolliert werden, getestet. Dabei handelt es sich um die Moduln Arbeitsstunden_abrechnen, Dienstplan_bearbeiten, Dienstplan_genehmigen, Dienstplan_lesen, Dienstplan_loeschen und Personaldaten_bearbeiten. Diese Moduln besitzen jeweils nur eine von außen aufrufbare Prozedur. Beim Test werden die Moduln instanziiert, und die entsprechende Prozedur wird aufgerufen.

Beim Test der Moduln zur Kontrolle der Ereignisflüsse der Anwendungsfälle werden sowohl Prozeduren, die in den Benutzerschnittstellenmoduln (siehe Phase 1) definiert sind als auch Prozeduren, die in den Moduln, die die Materialklassen repräsentieren (siehe Phase 2), definiert sind, benötigt.

Um alle Moduln weitgehend unabhängig voneinander testen zu können, werden für die Moduln zur Darstellung der Materialklassen Dienstplan, Genehmigung, Stundenguthaben und Personalakte neue Moduln implementiert, die die Moduln zur Darstellung der Materialklassen simulieren. Die Prozeduren in den Moduln zur Simulation besitzen nicht die vollständige Funktionalität der Prozeduren in den Originalmoduln, und die Instanzen der Moduln zur Simulation speichern immer einen fest vorgegebenen Zustand.

Zusätzlich könnten auch für die Moduln, in denen die Operationen für die Kommunikation mit dem Benutzer definiert sind, Simulationsmoduln implementiert und beim Modultest verwendet werden. Da die Modellierung der Benutzerschnittstellen bei dieser Diplomarbeit jedoch nur eine untergeordnete Rolle spielt (siehe Abschnitt 5.4.1), sind die Prozeduren in den Benutzerschnittstellenmoduln bereits so einfach aufgebaut, daß eine Simulation dieser Prozeduren die vorhandenen Prozeduren kaum vereinfachen würde und die Moduln zur Simulation der in der Phase 1 getesteten Moduln lediglich zusätzliche potentielle Fehlerquellen wären. Die im Rahmen dieses Fallbeispiels implementierten ProSet-Schnittstellenmoduln werden stattdessen selbst als Simulation einer komfortableren Benutzerschnittstelle angesehen, die im Rahmen dieser Diplomarbeit nicht betrachtet wird. Eine Implementierung von zusätzlichen Moduln zur Simulation der Benutzerschnittstelle wird daher nicht durchgeführt.

Der Quellcode der Programme und der Moduln, die zum Test implementiert wurden, befindet sich im Zusatzdokument zur Diplomarbeit.

Integrationstest: Der Integrationstest umfaßt nach Jacobson et al. eine Vielzahl von Testaktivitäten, in denen sowohl einzelne Komponenten eines Software-Systems als auch das gesamte System getestet werden [JCJÖ93, S. 330ff.]. Im Rahmen dieser Diplomarbeit werden beim Integrationstest zum einen die einzelnen Anwendungsfälle isoliert voneinander getestet. Zum anderen werden Tests durchgeführt, die sich auf das gesamte System beziehen. Zur Kontrolle der Testergebnisse sind die Spezifikation der Anwendungsfälle im Anwendungsfallmodell (siehe Abschnitt 5.4.1) und die Darstellung der Interaktionen zwischen den beteiligten Objekten in den Interaktionsdiagrammen (siehe Abschnitt 5.4.3) wichtig.

Zusätzlich beschreiben Jacobson et al. noch einen *Systemtest* (*system test*). Hier soll das erstellte System aus der Sicht der Anwender getestet werden [JCJÖ93, S. 333]. Ein solcher Test des Systems aus der Sicht der Anwender läßt sich im Rahmen dieser Diplomarbeit jedoch aus den folgenden Gründen nicht korrekt nachbilden:

- Das entwickelte Pilotsystem dient zur Betrachtung des Prototyping mit persistenten Daten mit ProSet im Rahmen einer objektorientierten Vorgehensweise bei der Systementwicklung. Es basiert jedoch nicht auf den Anforderungen einer speziellen Gruppe von Endanwendern, wie dies bei der Entwicklung eines Systems, das in der Praxis eingesetzt werden soll, der Fall wäre. Deshalb sind die Bedingungen, unter denen das erstellte System betrieben werden würde (z.B. Anzahl der mit dem System arbeitenden Personen oder Größe der zu verwaltenden Datenmengen), nicht bekannt.

- In dieser Diplomarbeit können nur Ausschnitte aus einer vollständigen Systementwicklung betrachtet werden. So wird z.B. auf die Modellierung einer graphischen Benutzeroberfläche vollständig verzichtet. Bei einem System, das in der Praxis eingesetzt werden soll, wäre die Benutzeroberfläche jedoch ein wesentlicher Bestandteil des Systems.

- In dieser Diplomarbeit wird eine Transformation des ProSet-Codes in den Code einer anderen Programmiersprache nicht mehr betrachtet. Zum Test steht in dieser Arbeit daher nur die ProSet-Implementierung des Pilotsystems zur Verfügung. Durch ProSet kann zwar die Programmerstellung verkürzt werden, weil bereits relativ früh ein ablauffähiges Modell des zu entwickelnden Systems zu Verfügung steht, eine Implementierung eines Anwendungssystems mit ProSet als Zielsprache ist jedoch nicht realistisch (siehe Abschnitt 5.5.1, Unterabschnitt *Einsatz anderer Programmiersprachen*). Da beim Systemtest im Gegensatz zum Integrationstest nicht mehr unter idealisierten Bedingungen gearbeitet werden kann, führten die Einschränkungen, die in ProSet noch bestehen, wie z.B. das Auftreten von Deadlocks beim parallelen Datenzugriff (siehe Abschnitt 3.6) im Zusammenhang mit der noch nicht vollständig funktionierenden Ausnahmebehandlung oder lange Zugriffszeiten bei der Verwaltung einer großen Anzahl persistenter Daten durch den grobgranularen Datenzugriff (siehe Beschreibung der Implementierung des Moduls Personalakte im Abschnitt 5.5.1) zu Problemen bei der Arbeit mit dem ProSet-System.

Der Systemtest wird in dieser Diplomarbeit daher nicht weiter betrachtet.

5.6 Beschreibung der Anforderungen an das erweiterte System

Das in den Abschnitten 5.3 bis 5.5 vorgestellte Pilotsystem wird nun erweitert. Das erweiterte System stellt selbst wieder ein Pilotsystem für nachfolgende Erweiterungen im Rahmen der evolutionären Systementwicklung dar. Die weitere evolutionäre Systementwicklung nach der Erstellung des erweiterten Systems wird im Rahmen dieser Diplomarbeit jedoch nicht mehr betrachtet. Der Begriff Pilotsystem wird im folgenden ausschließlich für das in den Abschnitten 5.3 bis 5.5 beschriebene System verwendet. Synonym zum Begriff Pilotsystem wird im folgenden der Begriff *Kernsystem* benutzt. Das System, das aus der Erweiterung des Kernsystems hervorgehen soll, wird im folgenden als *erweitertes System* bezeichnet. Die zusätzlichen Anforderungen an das erweiterte System werden im folgenden beschrieben.

Das erweiterte System stellt zusätzlich zu den Funktionen des Pilotsystems Funktionen zur Koordination der Terminplanung der Pflegekräfte und der Ärzte bereit. Neben den Daten, die bereits im Kernsystem zu den Pflegekräften gespeichert werden, verwaltet das erweiterte System zusätzlich

- die Namen der im betrachteten Krankenhaus arbeitenden Ärzte und
- die Anzahl der zustehenden Urlaubstage für jeden Arzt.

Des weiteren werden die folgenden Einschränkungen für die Planung von Terminen (z.B. für Urlaube oder Fortbildungsmaßnahmen) gespeichert:

- Anzahl der Ärzte, die im gesamten Krankenhaus immer mindestens erreichbar sein müssen (eine Person gilt an den Tagen als erreichbar, an denen sie keinen Termin geplant hat),
- Anzahl der ausgebildeten Pflegekräfte für die Tagschichten, die pro Station immer mindestens erreichbar sein müssen,
- Anzahl der ausgebildeten Pflegekräfte für die Nachtschichten, die pro Station immer mindestens erreichbar sein müssen.

Das erweiterte System soll die folgenden über das Pilotsystem hinausgehenden Möglichkeiten bieten:

- Die Tage, an denen eine Pflegekraft oder ein Arzt nicht im Krankenhaus arbeiten kann, werden in einem Terminkalender, der den Zeitraum eines Jahres umfaßt, markiert. Als Vorbild für diesen Terminkalender dient der Terminkalender, der bei der Beschreibung des Ist-Zustandes im Abschnitt 5.2 in der Abbildung 5.2 für die Terminplanung der Pflegekräfte einer Station skizziert wurde.

  Bei der Planung von Terminen soll zwischen Urlaubsterminen und sonstigen Terminen (z.B. für Fortbildungsmaßnahmen) unterschieden werden können. Wenn durch den Eintrag eines Termins die Anzahl der Ärzte oder der Pflegekräfte, die mindestens erreichbar sein müssen, unterschritten wird, wird ein Warnhinweis ausgegeben und die Möglichkeit geboten, den Eintrag rückgängig zu machen.

  Zusätzlich kann bei der Planung neuer Urlaubstermine für Ärzte oder Pflegekräfte durch das erweiterte System überprüft werden, ob die Anzahl der geplanten Urlaubstage die Anzahl der zustehenden Urlaubstage überschreitet. Ist dies der Fall, wird eine Warnmeldung ausgegeben und die Möglichkeit geboten, einen „erlaubten" Zustand des Terminkalenders wiederherzustellen.

  Der Vergleich der Anzahl geplanter und der Anzahl zustehender Urlaubstage kann deaktiviert werden. In diesem Fall wird keine Warnmeldung ausgegeben, wenn bei der Planung neuer Urlaubstermine die Anzahl der zustehenden Urlaubstage überschritten wird.

  Im Rahmen der Diplomarbeit werden bei der Terminplanung nur vollständige Tage betrachtet, an denen eine Person für das Krankenhaus entweder erreichbar ist oder einen Termin im Terminkalender eingetragen hat. Bei einem System, das in der Praxis eingesetzt werden soll, könnte es sinnvoll sein, eine feingranularere Terminplanung zu ermöglichen.

- Die eingetragenen Termine werden bei der Planung des Pflegedienstes berücksichtigt (siehe Abschnitt 5.3). Wenn ein Dienstplan erstellt werden soll, werden die im Terminkalender eingetragenen Termine einer Pflegekraft auch auf Dienstplan dargestellt. Das zu erstellende System sorgt dafür, daß die geplanten Termine im Terminkalender und auf den Dienstplänen immer konsistent sind.

5.7 Erweiterung der Modelle zur Analyse und zum Entwurf

Um das Pilotsystem zu erweitern, wird auf die bereits zum Pilotsystem erstellten Objectory-Modelle zurückgegriffen. Diese Modelle werden anschließend bei der Entwicklung des erweiterten Systems ergänzt. In diesem Abschnitt werden das erweiterte Anforderungsmodell (Abschnitt 5.7.1), das erweiterte Analysemodell (Abschnitt 5.7.2) und das erweiterte Entwurfsmodell (Abschnitt 5.7.3) vorgestellt.

5.7.1 Das Anforderungsmodell

Das erweiterte Anforderungsmodell basiert auf dem im Abschnitt 5.4.1 vorgestellten Anforderungsmodell des Kernsystems. Das Anforderungsmodell des Kernsystems wird für die Modellierung des erweiterten Systems übernommen und anschließend ergänzt, um die im Abschnitt 5.6 vorgestellten zusätzlichen Anforderungen zu erfüllen zu können. Wie beim Pilotsystem werden auch beim erweiterten System im Rahmen dieser Diplomarbeit keine Schnittstellenbeschreibungen betrachtet, da die Modellierung von Benutzerschnittstellen in dieser Diplomarbeit nur eine untergeordnete Rolle spielt. Bei einem System das in der Praxis eingesetzt werden soll, wäre eine Beschreibung der Benutzerschnittstellen im Gegensatz dazu erforderlich.

Im folgenden werden das Anwendungsfallmodell und das Problembereichsmodell des erweiterten Systems vorgestellt.

Das Anwendungsfallmodell

Die Akteure des erweiterten Anwendungsfallmodells sind in der Abbildung 5.10 dargestellt. Zusätzlich zu den Akteuren des im Abschnitt 5.4.1 vorgestellten Anwendungsfallmodells (siehe Abbildung 5.3) werden nun auch die Ärzte des Krankenhauses berücksichtigt.

Das Anwendungsfallmodell des erweiterten Systems enthält sämtliche Anwendungsfälle, die bereits im Anwendungsfallmodell des Kernsystems modelliert sind (siehe Abbildung 5.4 im Abschnitt 5.4.1), sowie zusätzliche Anwendungsfälle. Die Anwendungsfälle des erweiterten Systems sind in den Abbildungen 5.11 und 5.12 dargestellt. Bei den neu modellierten Anwendungsfällen kann zwischen den eigenständigen Anwendungsfällen

- Termineinschränkungen festlegen,
- Terminkalender bearbeiten und
- Terminkalender löschen

und den Anwendungsfällen

- Termineinschränkungen überprüfen,
- Terminkalender abfragen und
- Terminkalender anpassen,

die dazu dienen, andere Anwendungsfälle zu erweitern, unterschieden werden.

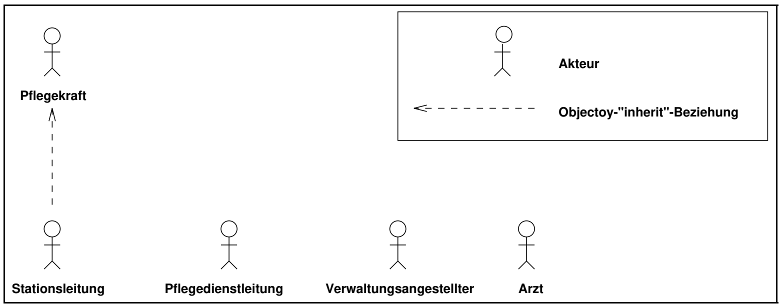

Abbildung 5.10: Akteure für das erweiterte System.

Die Akteure, mit denen die Anwendungsfälle Termineinschränkungen überprüfen, Terminkalender abfragen und Terminkalender anpassen interagieren, werden weiter unten in diesem Abschnitt bei der Kurzbeschreibung dieser Anwendungsfälle vorgestellt. Im folgenden werden alle neuen Anwendungsfälle kurz beschrieben:

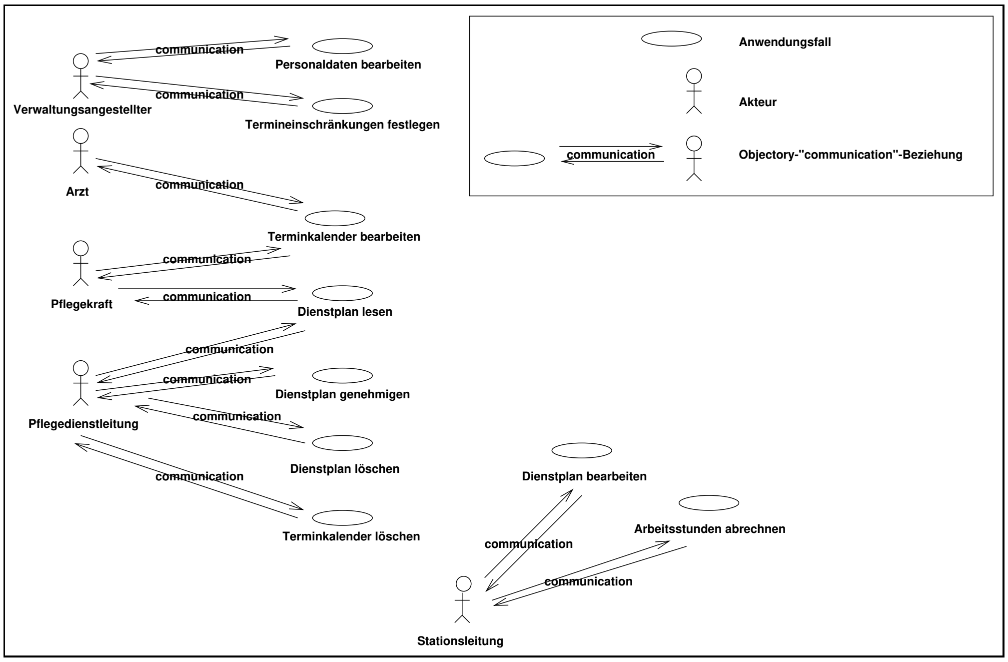

Abbildung 5.11: Eigenständige Anwendungsfälle des erweiterten Systems und Akteure.

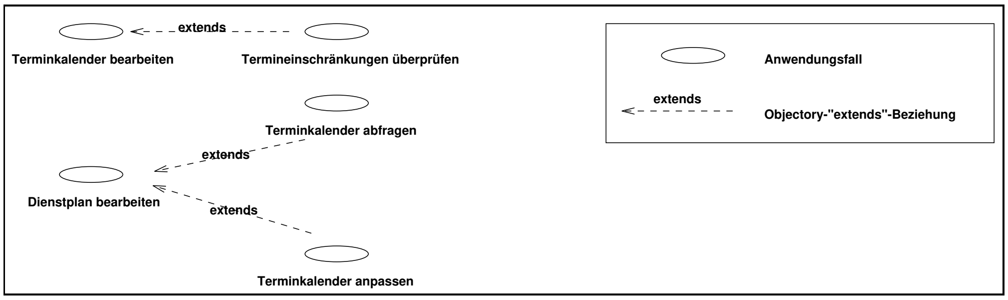

Abbildung 5.12: Erweiterungen von Anwendungsfällen.

Eigenständige Anwendungsfälle (siehe Abbildung 5.11)

Der Anwendungsfall Termineinschränkungen festlegen**:** Bei der Ausführung einer Instanz dieses Anwendungsfalls werden die folgenden Festlegungen getroffen und gespeichert:

- Es wird festgelegt, wieviele Ärzte des Krankenhauses immer mindestens erreichbar sein müssen. Dabei wird eine Person immer dann als erreichbar angesehen, wenn diese Person für den Tag, für den die Erreichbarkeit überprüft werden soll, keinen Termin im Terminkalender markiert hat.
- Es wird festgelegt, wieviele ausgebildete Krankenschwestern bzw. Krankenpfleger für die Tagschichten pro Station des Krankenhauses immer mindestens erreichbar sein müssen.
- Es wird festgelegt, wieviele ausgebildete Krankenschwestern bzw. Krankenpfleger für die Nachtschichten pro Station des Krankenhauses immer mindestens erreichbar sein müssen.
- Es wird festgelegt, ob das System bei der Planung neuer Urlaubstermine für einen Arzt oder für eine Pflegekraft einen Warnhinweis ausgeben soll, wenn die Anzahl der geplanten Urlaubstage die Anzahl der zustehenden Urlaubstage überschreitet.

Der Anwendungsfall Terminkalender bearbeiten: Durch die Ausführung einer Instanz des Anwendungsfalls Terminkalender bearbeiten werden die im Terminkalender markierten Termine angezeigt (für jedes Jahr verwaltet das erweiterte System genau einen Terminkalender).

Bei Bedarf können Ärzte und Pflegekräfte im Terminkalender weitere Tage markieren, an denen sie nicht erreichbar sind. Dabei wird zwischen Urlaubsterminen und sonstigen Terminen unterschieden.

Damit durch neue Einträge im Terminkalender keine bereits erstellten Dienstpläne verändert werden können, kann eine Pflegekraft nur in den Monaten Termine im Terminkalender markieren, an denen für sie noch kein Dienstplan erstellt wurde.

Bei einem System, das in der Praxis eingesetzt werden soll, wäre ein Schutzmechanismus sinnvoll, mit dessen Hilfe sichergestellt wird, daß eine Person nur die eigenen Einträge im Terminkalender manipulieren kann. Gemäß der im Abschnitt 5.1 beschriebenen Voraussetzung, Datenschutz- und Datensicherheitsaspekte im Rahmen dieser Diplomarbeit außer acht zu lassen, wird ein solcher Schutzmechanismus hier nicht betrachtet.

Der Anwendungsfall Terminkalender löschen: Wenn ein Terminkalender eines Jahres nicht mehr benötigt wird, kann die Pflegedienstleitung den Terminkalender löschen.

Ein eigenständiger Anwendungsfall Terminkalender lesen, der es den Ärzten und Pflegekräften ermöglicht, die in einem Terminkalender markierten Termine anzeigen zu lassen, ist nicht nötig, weil die Ausgabe von Terminen auf dem Bildschirm bereits ein Bestandteil des Anwendungsfalls Terminkalender bearbeiten ist. Die Ausführung von Instanzen des Anwendungsfalls Terminkalender bearbeiten kann sowohl durch Ärzte als auch durch Pflegekräfte gestartet werden.

Anwendungsfälle zur Erweiterung (siehe Abbildung 5.12)

Der Anwendungsfall Termineinschränkungen überprüfen: Der Anwendungsfall Termineinschränkungen überprüfen interagiert mit denselben Akteuren wie der Anwendungsfall Terminkalender bearbeiten.

Mit Hilfe des Anwendungsfalls Termineinschränkungen überprüfen wird überprüft, ob durch die Markierung eines neuen Termins im Terminkalender die Einschränkungen für die Planung von Terminen (siehe Beschreibung des Anwendungsfalls Termineinschränkungen festlegen) verletzt werden.

Werden die Einschränkungen verletzt, wird ein Warnhinweis ausgegeben. Der Benutzer kann den Eintrag im Terminkalender, durch den der Warnhinweis ausgelöst wurde, stornieren. Die Ausgabe des Warnhinweises und die Möglichkeit, die Markierung des Termins im Terminkalender zu stornieren, wird hier als Unterbrechung des normalen Ablaufs des Anwendungsfalls Terminkalender bearbeiten aufgefaßt. Gemäß der Empfehlung Jacobson et al. wird der Anwendungsfall, der die Unterbrechung verursacht (der Anwendungsfall Termineinschränkungen überprüfen) durch eine „extends"-Beziehung mit dem Anwendungsfall, der unterbrochen wird (der Anwendungsfall Terminkalender_bearbeiten), verbunden [JCJÖ93, S. 165].

Der Anwendungsfall Terminkalender abfragen: Der Anwendungsfall Terminkalender abfagen erweitert den Anwendungsfall Dienstplan bearbeiten. Falls ein neuer Dienstplan erstellt werden soll, wird überprüft, ob für die betroffenen Pflegekräfte Termine im Terminkalender des betroffenen Jahres eingetragen sind. Ist dies der Fall, werden diese Termine in den neu zu erstellenden Dienstplan aufgenommen, bevor er durch die Stationsleitung nach den Vorgaben des Anwendungsfallls Dienstplan bearbeiten bearbeitet wird. Es finden keine Interaktionen zwischen einem Akteur und dem Anwendungsfall Terminkalender abfragen statt.

Um den Anwendungsfall Dienstplan bearbeiten aus dem Pilotsystem unverändert in das erweiterte System übernehmen zu können, wird für das Lesen des Terminkalenders ein eigenständiger Anwendungsfall modelliert, der durch eine „extends"-Beziehung mit dem Anwendungsfall Dienstplan bearbeiten verbunden wird, wie es Jacobson et al. für das Einfügen zusätzlicher Routinen in bestehende Anwendungsfälle empfehlen [JCJÖ93, S. 173].

Der Anwendungsfall Terminkalender anpassen: Der Anwendungsfall Terminkalender anpassen erweitert den Anwendungsfall Dienstplan bearbeiten. Er interagiert wie der Anwendungsfall Dienstplan bearbeiten mit dem Akteur Stationsleitung und dient dazu, die Konsistenz zwischen erstellten Dienstplänen und den zu einem Terminkalender gespeicherten Daten sicherzustellen. Um den Anwendungsfall Dienstplan bearbeiten aus dem Pilotsystem unverändert in das erweiterte System übernehmen zu können, wird analog zum Anwendungsfall Terminkalender abfragen ein eigenständiger Anwendungsfall zum Sicherstellen der Konsistenz zwischen einem Dienstplan und dem Terminkalender des betroffenen Jahres modelliert und mit dem Anwendungsfall Dienstplan bearbeiten durch eine „extends"-Beziehung verbunden.

Wenn ein Dienstplan gespeichert wird, sorgt der Anwendungsfall Terminkalender anpassen dafür, daß die Daten des betroffenen Terminkalenders an die Daten des gespeicherten Dienstplans angepaßt werden, falls dies nötig ist. Zusätzlich wird überprüft, ob die Einschränkungen für die Planung von Terminen (siehe Beschreibung des Anwendungsfalls Termineinschränkungen festlegen) bei der Bearbeitung eines Dienstplans verletzt wurden. Wurden die Einschränkungen verletzt, wird ein Warnhinweis ausgegeben. Die Manipulationen, die am betrachteten Dienstplan durchgeführt wurden, können dann auf Befehl des Benutzers rückgängig gemacht werden. Die Konsistenz zwischen dem betroffenen Terminkalender und dem betrachteten Dienstplan wird in jedem Fall sichergestellt.

Die vollständigen Beschreibungen aller Anwendungsfälle sind im Zusatzdokument zur Diplomarbeit aufgeführt.

Das Problembereichsmodell

Das im Abschnitt 5.4.1 vorgestellte Problembereichsmodell des Kernsystems wird nun um die Materialien Termineinschränkungen und Terminkalender ergänzt, wie es in der Abbildung 5.13 skizziert ist.

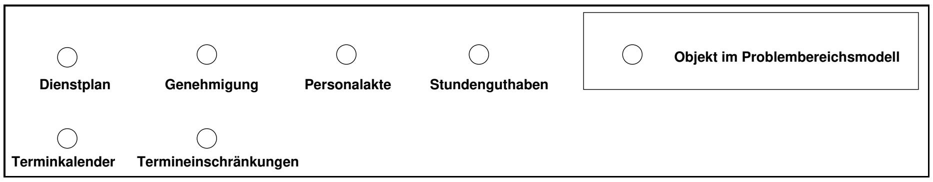

Abbildung 5.13: Objekte im Problembereichsmodell des erweiterten Systems.

Die neuen Objekte Termineinschränkungen und Terminkalender werden im folgenden beschrieben:

Das Objekt Termineinschränkungen: Das Objekt Termineinschränkungen repräsentiert die Einschränkungen, die für die Markierung von Terminen im Terminkalender vorgegeben sind (siehe Beschreibung des Anwendungsfalls Termineinschränkungen festlegen). Diese Einschränkungen werden im einzelnen durch die folgenden Vorgaben bestimmt:

- Es ist Mindestanzahl von Ärzten festgelegt, die im Krankenhaus immer erreichbar sein müssen (eine Person ist an den Tagen erreichbar, an denen sie keinen Termin im Terminkalender markiert hat).
- Es ist eine Mindestanzahl von ausgebildeten Krankenschwestern und Krankenpflegern festgelegt, die pro Station für die Tagschichten immer erreichbar sein müssen.
- Es ist eine Mindestanzahl von ausgebildeten Krankenschwestern und Krankenpflegern festgelegt, die pro Station für die Nachtschichten immer erreichbar sein müssen.
- Es ist festgelegt, ob bei der Markierung neuer Urlaubstermine im Terminkalender überprüft werden soll, ob die Anzahl der geplanten Urlaubstage der Ärzte und der Pflegekräfte die Anzahl der zustehenden Urlaubstage überschreitet.

Das Objekt Terminkalender: Das Objekt Terminkalender repräsentiert den Terminkalender eines Jahres, in dem Ärzte und Pflegekräfte die Tage markieren, an denen sie für das Krankenhaus nicht erreichbar sind. Als Vorbild dient der Terminkalender, der in der Abbildung 5.2 (siehe Abschnitt 5.2) skizziert ist.

5.7.2 Das Analysemodell

Im Analysemodell des erweiterten Systems werden zunächst sämtliche Objekte, Attribute und Beziehungen, die im Analysemodell des Kernsystems (siehe Abschnitt 5.4.2) modelliert sind, übernommen. Anschließend wird das Modell erweitert. Im folgenden wird zunächst auf die Modellierung der Materialien im Analysemodell des erweiterten Systems eingegangen, die in der Abbildung 5.14 dargestellt ist.

Zusätzlich zu den in der Abbildung 5.5 (siehe Abschnitt 5.4.2) skizzierten Materialien des Analysemodells des Pilotsystems sind in der Abbildung 5.14 die Materialobjekte Terminkalender und Termineinschränkungen aufgeführt.

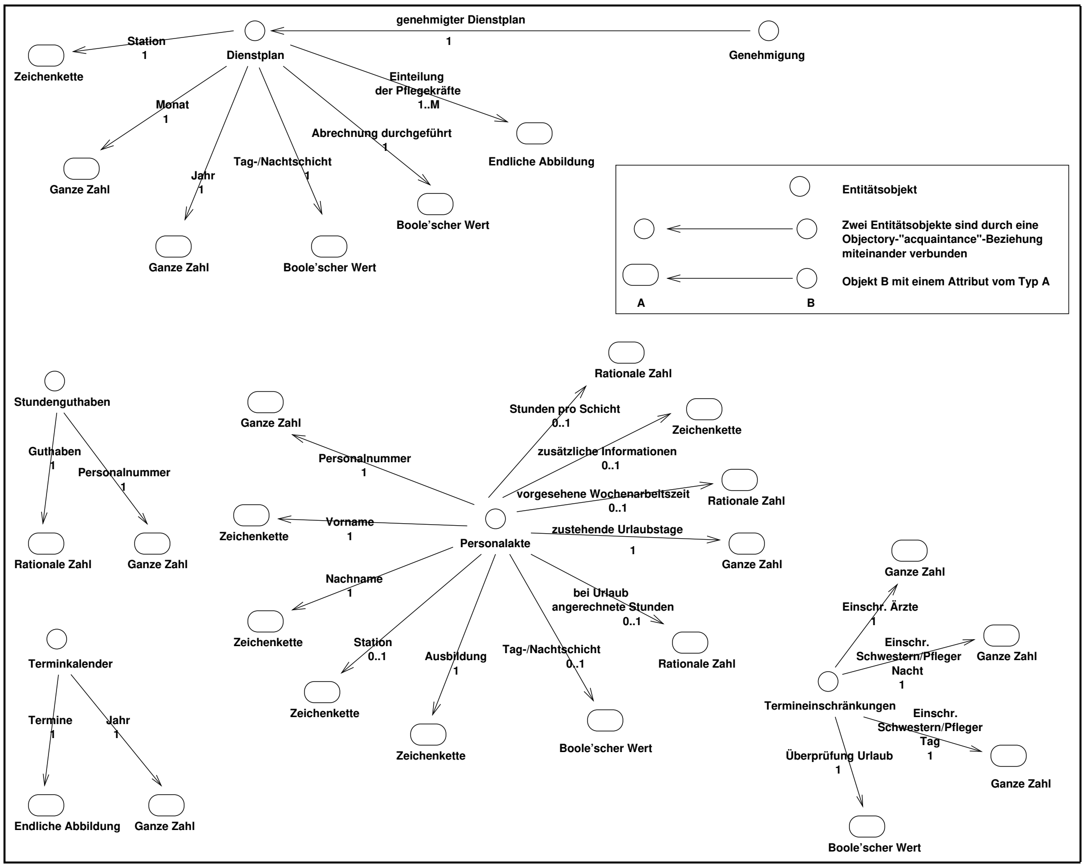

Abbildung 5.14: Materialien im Analysemodell des erweiterten Systems.

Das Materialobjekt Terminkalender besitzt zwei Attribute. In diesen Attributen wird zum einen das Jahr gespeichert, auf das sich der Terminkalender, der durch das Objekt repräsentiert wird, bezieht. Zum anderen werden die markierten Termine der Ärzte und der Pflegekräfte gespeichert. Zum Speichern der Termine der Ärzte und der Pflegekräfte ist eine endliche Abbildung vorgesehen. In dieser endlichen Abbildung kann zu jeder Person eine Liste aller geplanten Termine gespeichert werden.

Das Materialobjekt Termineinschränkungen besitzt insgesamt vier Attribute. In diesen Attributen werden die folgenden Daten gespeichert:

Anwendungsfall	Kontrollobjekt
Temineinschränkungen festlegen	Temineinschränkungen_festlegen
Terminkalender bearbeiten	Terminkalender_bearbeiten
Termineinschränkungen überprüfen	Termineinschränkungen_überprüfen
Terminkalender löschen	Terminkalender_löschen
Terminkalender abfragen	Terminkalender_abfragen
Terminkalender anpassen	Terminkalender_anpassen

Tabelle 5.6: Zuordnung von neuen Kontrollobjekten im Analysemodell des erweiterten Systems zu Anwendungsfällen.

- Anzahl der Ärzte, die im Krankenhaus immer mindestens erreichbar sein müssen,
- Anzahl der Krankenschwestern bzw. Krankenpfleger für die Tagschichten, die pro Station immer mindestens erreichbar sein müssen,
- Anzahl der Krankenschwestern bzw. Krankenpfleger für die Nachtschichten, die pro Station immer mindestens erreichbar sein müssen,
- Angabe, ob bei der Planung neuer Urlaubstermine überprüft werden soll, ob die Anzahl geplanter Urlaubstage der Ärzte und der Pflegekräfte die Anzahl der zustehenden Urlaubstage überschreitet.

Die Modellierung der Materialobjekte Dienstplan, Genehmigung, Personalakte und Stundenguthaben wird unverändert aus dem Analysemodell des Kernsystems übernommen (siehe Abbildung 5.6 im Abschnitt 5.4.2).

Die Modellierung des vollständigen erweiterten Systems ist in der Abbildung 5.15 skizziert. Um die Abbildung 5.15 übersichtlicher zu gestalten, sind statische Beziehungen zwischen Objekten und Attribute hier nicht mehr aufgeführt. Zur Modellierung des erweiterten Systems wird die in der Abbildung 5.7 (siehe Abschnitt 5.4.2) skizzierte Struktur des Kernsystems übernommen. Anschließend wird das übernommene Modell um zusätzliche Objekte und Beziehungen ergänzt. Die Erweiterungen gegenüber der im Abschnitt 5.4.2 beschriebenen Systemstruktur des Kernsystems werden im folgenden vorgestellt:

- Zusätzlich zu den übernommenen Materialobjekten Dienstplan, Genehmigung, Personalakte und Stundenguthaben werden die in der Abbildung 5.14 dargestellten neuen Materialobjekte Termineinschränkungen und Terminkalender aufgeführt.
- Für jeden Anwendungsfall des erweiterten Systems, der nicht im Anwendungsfallmodell des bereits implementierten ursprünglichen Systems enthalten ist (siehe Abschnitt 5.7.1), wird ein neues Kontrollobjekt modelliert. Die neu modellierten Kontrollobjekte und die zugehörigen Anwendungsfälle aus dem Anwendungsfallmodell des erweiterten Systems sind in der Tabelle 5.6 aufgeführt.
- Die Objectory-„extends"-Beziehungen, die im Anwendungsfallmodell zwischen den Anwendungsfällen
    - Termineinschränkungen überprüfen und Terminkalender bearbeiten,
    - Terminkalender abfragen und Dienstplan bearbeiten sowie
    - Terminkalender anpassen und Dienstplan bearbeiten

  modelliert sind (siehe Abbildung 5.12 im Abschnitt 5.7.1), werden im Analysemodell direkt als „extends"-Beziehungen zwischen den Klassen der Kontrollobjekte, die den entsprechenden Anwendungsfällen zugeordnet sind, übernommen. Die Namen der betroffenen Kontrollobjekte können aus den Tabellen 5.1 (siehe Abschnitt 5.4.2) und 5.6 entnommen werden.

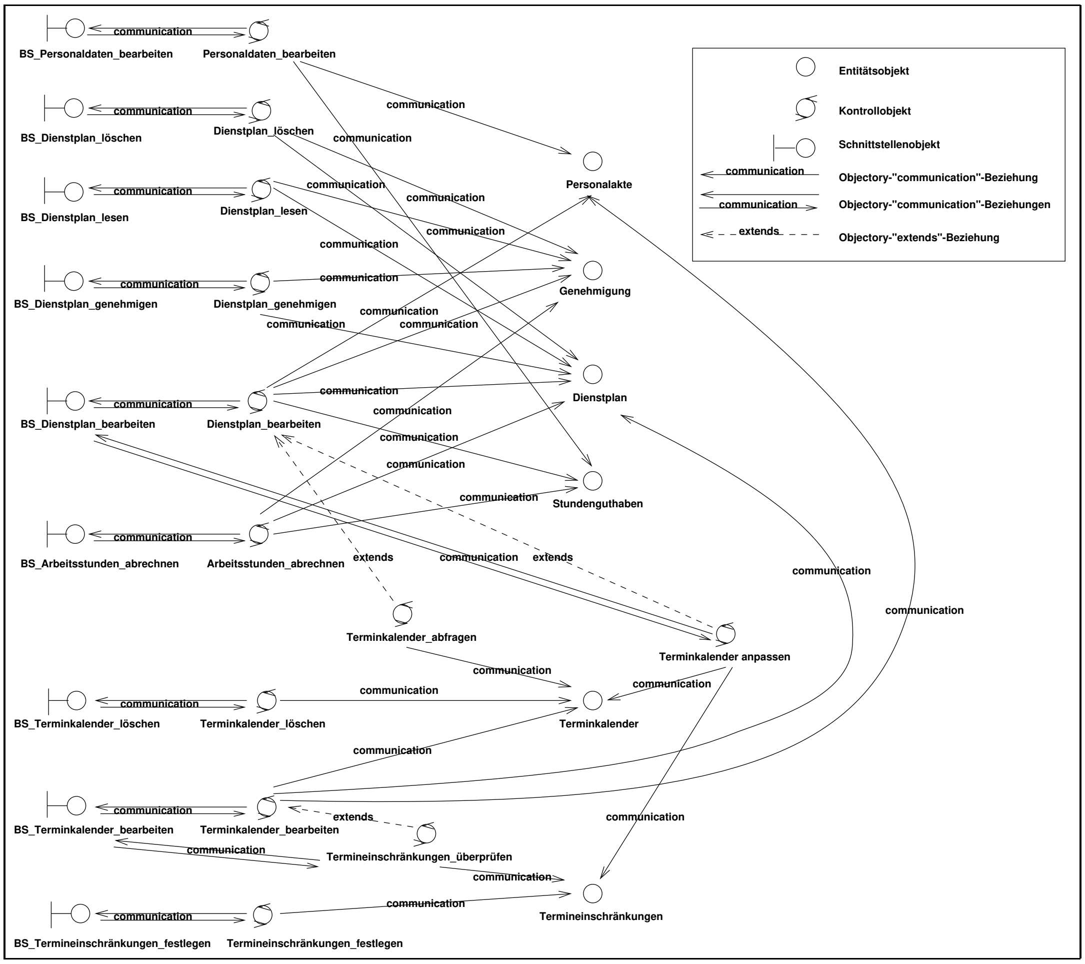

Abbildung 5.15: Analysemodell des erweiterten Systems ohne Berücksichtigung von Attributen und statischen Beziehungen zwischen Instanzen.

- Wenn die für das Analysemodell des erweiterten Systems neu modellierten Kontrollobjekte (siehe Tabelle 5.6) Daten anzeigen sollen oder Eingaben des Benutzers benötigen, ist es nötig, für diese Kontrollobjekte neue „communication"-Beziehungen zu Schnittstellenobjekten, durch die die Kommunikation zwischen dem System und dem Benutzer abgewickelt wird, einzufügen. Bei dem hier modellierten System können dabei zwei Fälle unterschieden werden:

  Fall 1: Ein neu modelliertes Kontrollobjekt kann auf Operationen zurückgreifen, die von einem Schnittstellenobjekt bereitgestellt werden, das schon im Analysemodell des bereits implementierten Pilotsystems modelliert ist. In diesem Fall werden Objectory-„communication"-Beziehungen zwischen dem Kontrollobjekt und dem entsprechenden Schnittstellenobjekt modelliert.

  Dies ist bei dem Kontrollobjekt Terminkalender_anpassen der Fall. Zur Durchführung der Kommunikation mit dem Benutzer des Systems sendet das Kontrollobjekt Terminkalender_anpassen Stimuli an das Objekt BS_Dienstplan_bearbeiten, um in diesem Objekt die Ausführung von Operationen zu veranlassen. Eingaben des Benutzers werden vom Objekt BS_Dienstplan_bearbeiten an das Objekt Terminkalender_anpassen gesendet.

  Das Objekt BS_Dienstplan_bearbeiten wird aus dem Analysemodell des ursprünglichen Systems übernommen (siehe Abbildung 5.7 im Abschnitt 5.4.2).

Analysemodellobjekte	Beschreibung des Werkzeugs
BS_Personaldaten_bearbeiten Personaldaten_bearbeiten	Eine Personalakte kann neu angelegt, modifiziert oder gelöscht werden.
BS_Dienstplan_löschen Dienstplan_löschen	Ein Dienstplan kann gelöscht werden.
BS_Dienstplan_lesen Dienstplan_lesen	Ein Dienstplan kann gelesen werden.
BS_Dienstplan_genemigen Dienstplan_genehmigen	Ein Dienstplan kann genehmigt werden. Wenn der Dienstplan nicht alle Angaben enthält, die für eine Abrechnung der geleisteten Arbeitsstunden nötig sind, oder wenn nicht alle Schichten vorschriftsmäßig besetzt sind, wird ein Warnhinweis ausgegeben. Der Benutzer kann die Genehmigung des gewählten Dienstplans nach der Ausgabe dieses Warnhinweises noch rückgängig machen.
BS_Dienstplan_bearbeiten Dienstplan_bearbeiten Terminkalender_abfragen Terminkalender_anpassen	Ein Dienstplan kann erstellt werden. Bereits erstellte Dienstpläne können modifiziert werden.
BS_Arbeitsstunden_abrechnen Arbeitsstunden_abrechnen	Die geleisteten Arbeitsstunden der Pflegekräfte werden berechnet und im Dienstplan eingetragen. Zusätzlich werden die Stundenguthaben der Pflegekräfte angepaßt. Bei Bedarf können die berechneten Stundenguthaben noch manuell geändert werden.
BS_Terminkalender_löschen Terminkalender_löschen	Ein nicht mehr benötigter Terminkalender wird durch die Pflegedienstleitung gelöscht.
BS_Terminkalender_bearbeiten Terminkalender_bearbeiten Termineinschränkungen_überprüfen	Ein Terminkalender kann gelesen und bearbeitet werden.
BS_Termineinschränkungen_festlegen Termineinschränkungen_festlegen	Die Einschränkungen für Einträge im Terminkalender werden festgelegt.

Tabelle 5.7: Beschreibung der Werkzeuge des erweiterten Systems und Angabe beteiligter Analysemodellobjekte.

Fall 2: Für Kontrollobjekte müssen neue Schnittstellenobjekte modelliert werden. Dies ist immer dann der Fall, wenn neue Kontrollobjekte kein bereits im Pilotsystem vorhandenes Werkzeug erweitern sondern Bestandteil eines neuen Werkzeugs sind. Dies ist bei den Kontrollobjekten Termineinschränkungen_festlegen, Termineinschränkungen_überprüfen, Terminkalender_löschen und Terminkalender_bearbeiten der Fall. Die Kontrollobjekte Termineinschränkungen_überprüfen und Terminkalender_bearbeiten benutzen dabei dasselbe Schnittstellenobjekt. Für die übrigen Kontrollobjekte wird jeweils ein eigenes Schnittstellenobjekt modelliert.

Des weiteren enthält das Analysemodell des erweiterten Systems noch das neu modellierte Kontrollobjekt Terminkalender_abfragen. Dieses Kontrollobjekt veranlaßt weder die Anzeige von Daten, noch benötigt es Eingaben eines Benutzers (siehe hierzu Beschreibung des Anwendungsfalls Terminkalender abfragen im Abschnitt 5.7.1). Für das Kontrollobjekt Terminkalender_abfragen wird deshalb kein Schnittstellenobjekt benötigt.

- Zusätzlich werden „communication"-Beziehung zwischen Kontrollobjekten und den Materialobjekten, auf die die Kontrollobjekte zugreifen, modelliert. Diese „communication"-Beziehungen gehen immer von den Kontrollobjekten aus und enden bei den Materialobjekten (siehe Abschnitt 4.5).

Die Werkzeuge des erweiterten Systems, die sich aus der in der Abbildung 5.15 skizzierten Modellierung ergeben, werden zusammen mit den zugehörigen Objekten in der Tabelle 5.7 vorgestellt. Diese Tabelle ist eine Erweiterung der Tabelle 5.2, die bei der Beschreibung des Analysemodells des Pilotsystems vorgestellt wurde (siehe Abschnitt 5.4.2).

5.7.3 Das Entwurfsmodell

Die Objekte, die Attribute und die Beziehungen zwischen Objekten des Entwurfsmodells des erweiterten Systems sind in der Abbildung 5.16 aufgeführt. Zur Erstellung dieses Modells wird zunächst

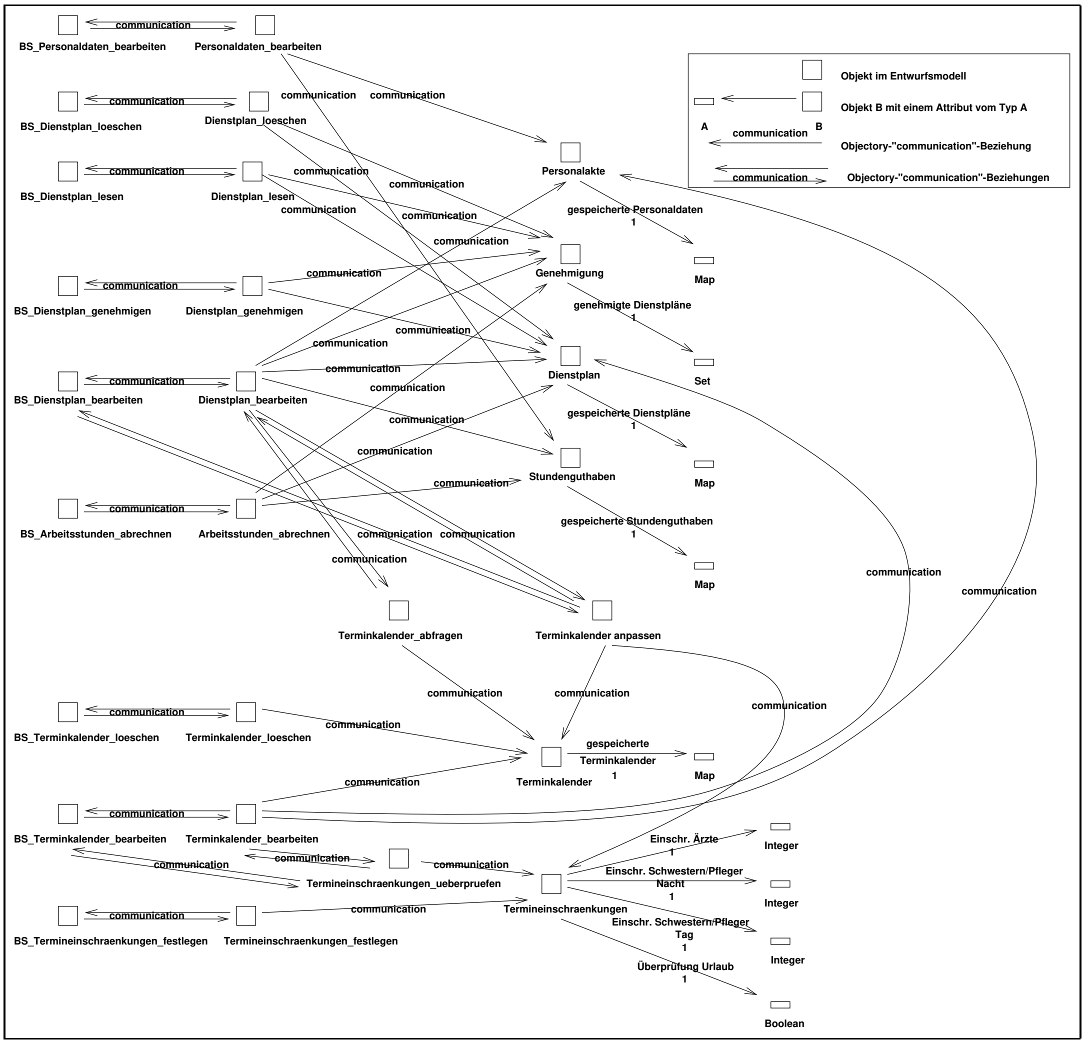

Abbildung 5.16: Objekte und Beziehungen im Entwurfsmodell des erweiterten Systems.

der in der Abbildung 5.8 (siehe Abschnitt 5.4.3) skizzierte Systementwurf des Pilotsystems übernommen. Das übernommene Modell wird anschließend folgendermaßen erweitert:

- Für jedes Objekt, das im Analysemodell des erweiterten Systems vorhanden ist (siehe Abbildung 5.15 im Abschnitt 5.7.2) und für das noch kein entsprechendes Objekt im Entwurfsmodell des erweiterten Systems modelliert ist, wird ein neues Objekt in das Entwurfsmodell des erweiterten Systems aufgenommen. Die neu aufgenommenen Entwurfsmodellobjekte werden so benannt wie die entsprechenden Analysemodellobjekte. Die einzige Ausnahme bilden hier wieder Objekte, in denen Umlaute vorkommen. Die Umlaute ä, ö und ü werden bei Entwurfsmodellobjekten durch ae, oe und ue ersetzt, weil Umlaute auf der Tastatur der Computer, die zur Programmierung zur Verfügung stehen, nicht vorhanden sind. Die neu in das Entwurfsmodell aufgenommenen Objekte und die zugehörigen Analysemodellobjekte sind in der Tabelle 5.8 aufgeführt.

Analysemodellobjekt	Entwurfsmodellobjekt
BS_Termineinschränkungen_festlegen	BS_Termineinschraenkungen_festlegen
BS_Terminkalender_bearbeiten	BS_Terminkalender_bearbeiten
BS_Terminkalender_löschen	BS_Terminkalender_loeschen
Termineinschränkungen	Termineinschraenkungen
Termineinschränkungen_festlegen	Termineinschraenkungen_festlegen
Termineinschränkungen_überprüfen	Termineinschraenkungen_ueberpruefen
Terminkalender	Terminkalender
Terminkalender_abfragen	Terminkalender_abfragen
Terminkalender_anpassen	Terminkalender_anpassen
Terminkalender_bearbeiten	Terminkalender_bearbeiten

Tabelle 5.8: Neue Objekte im erweiterten Analysemodell und im erweiterten Entwurfsmodell.

- Zu den Objekten Termineinschraenkungen und Terminkalender werden Attribute modelliert. Für das Objekt Termineinschraenkungen kann die Modellierung der Attribute direkt aus dem Analysemodell (siehe Abbildung 5.14 im Abschnitt 5.7.2) übernommen werden. Nur die Namen der Attributtypen werden im Entwufsmodell geändert. Im Entwurfsmodell werden die Bezeichnungen benutzt, die auch in der ProSet-Sprachbeschreibung verwendet werden. Statt der Bezeichnung Ganze Zahl wird die Bezeichnung Integer verwendet. Statt der Bezeichnung Boole'scher Wert wird die Bezeichnung Boolean verwendet. Die Modellierung der Attribute zum Objekt Terminkalender weicht hingegen von der Modellierung der Attribute zum Objekt Terminkalender im Analysemodell ab. Dies liegt daran, daß das Entwurfsmodellobjekt Terminkalender zur Speicherung sämtlicher vorhandener Terminkalender dient (z.B. Terminkalender für 1995, Terminkalender für 1996 und Terminkalender für 1997), während das Analysemodellobjekt Terminkalender genau einen Terminkalender eines Jahres enthalten soll.

  Die Motivation für die Entscheidung, sämtliche Terminkalender mit Hilfe genau eines Objektes zu speichern, entspricht der Motivation, die im Abschnitt 5.4.3 für die Speicherung aller vorhanden Personalakten in nur einem Entwurfsmodellobjekt gegeben wurde (Seite 114f.). Diese Diskussion soll in diesem Abschnitt nicht wiederholt werden.

  In dem Attribut, das zum Entwurfsmodellobjekt Terminkalender modelliert ist, werden sämtliche vorhandenen Terminkalender gespeichert. Anhand der Jahreszahl des Jahres, für den ein Terminkalender gültig ist, kann ein einzelner Terminkalender identifiziert werden. Als Attributtyp zur Speicherung der Terminkalender bietet sich daher eine endliche Abbildung an, die im vorgestellten Entwurfsmodell als Map bezeichnet wird.

- Mit Ausnahme der „extends"-Beziehungen werden sämtliche Beziehungen, die im Analysemodell zwischen Objekten bestehen, auch zwischen den entsprechenden Entwurfsmodellobjekten modelliert.

  Da „extends"-Beziehungen nicht direkt auf eine ProSet-Implementierung abgebildet werden können, werden anstelle der im Analysemodell modellierten „extends"-Beziehungen im Entwurfsmodell nach der Empfehlung von Jacobson et al. „communication"-Beziehungen modelliert. Die Grundlagen hierzu wurden bereits bei der Beschreibung der Erstellung des Entwurfsmodells für den Benutzeragenten des E-Mail-Systems vorgestellt (siehe Abschnitt 4.6.1). Das Ersetzen einer „extends"-Beziehung durch eine „communication"-Beziehung ist in der Abbildung 5.17 noch einmal exemplarisch für die Objekte Terminkalender_bearbeiten und Termineinschraenkungen_ueberpruefen (Schreibweise für die Entwurfsmodellobjekte ohne Umlaute) dargestellt.

Nach der Modellierung der Objekte und der Beziehungen zwischen Objekten werden die Interaktionsdiagramme für das erweiterte System erstellt. Abhängig vom betrachteten Anwendungsfall entstehen die Interaktionsdiagramme des Entwurfsmodells des erweiterten Systems auf eine der im folgenden beschriebenen drei Arten:

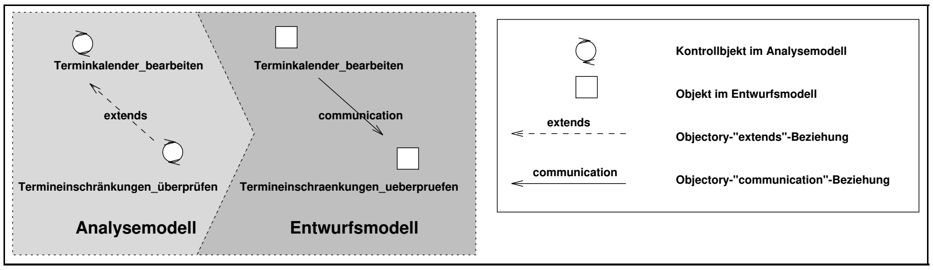

Abbildung 5.17: Ersetzen einer „extends"-Beziehung durch eine „communication"-Beziehung. Diese Skizze soll nur das Prinzip verdeutlichen. Da in dem betrachteten Fallbeispiel das Objekt Termineinschraenkungen_ueberpruefen auch Stimuli an das Objekt Terminkalender_bearbeiten senden kann, müßte im endgültigen Modell zusätzlich noch eine „communication"-Beziehung vom Objekt Termineinschraenkungen_ueberpruefen zum Objekt Terminkalender_bearbeiten hinzugefügt werden.

Fall 1 (direkte Übernahme): Die Interaktionsdiagramme werden direkt aus dem im Abschnitt 5.4.3 beschriebenen Entwurfsmodell des Kernsystems übernommen und nicht verändert. Hiervon sind die Interaktionsdiagramme für sämtliche Anwendungsfälle des Kernsystems mit Ausnahme des Interaktionsdiagramms für den Anwendungsfall Dienstplan bearbeiten betroffen (siehe Abbildung 5.4 im Abschnitt 5.4.1).

Fall 2 (Übernahme und Einfügung von Änderungen): Das Interaktionsdiagramm des Anwendungsfalls Dienstplan bearbeiten wird ebenfalls aus dem im Abschnitt 5.4.3 beschriebenen Entwurfsmodell übernommen. Bei dem erweiterten System wird jedoch berücksichtigt, daß Einträge im Terminkalender bei der Erstellung von Dienstplänen beachtet werden müssen und daß Änderungen von Dienstplänen Änderungen im Terminkalender bewirken können. Hierfür wird das Interaktionsdiagramm für den Anwendungsfall Dienstplan bearbeiten erweitert. Dabei werden neue Stimuli zwischen den Entwurfsmodellobjekten Dienstplan_bearbeiten und Terminkalender_abfragen sowie zwischen den Entwurfsmodellobjekten Dienstplan_bearbeiten und Terminkalender_anpassen (siehe Abbildung 5.16) in das Interaktionsdiagramm eingefügt.

Fall 3 (Neuerstellung): Für sämtliche Anwendungfälle, die beim Kernsystem noch nicht berücksichtigt wurden (siehe Abschnitt 5.7.1), werden neue Interaktionsdiagramme erstellt.

Die Interaktionsdiagramme des Systems zur Pflegedienst- und Terminplanung sind im Zusatzdokument zur Diplomarbeit aufgeführt. Zustands-Übergangsdiagramme werden nicht erstellt, weil die modellierten Objekte nicht zustandskontrolliert sind (siehe Abschnitt 5.4.3).

5.8 Implementierung und Test des erweiterten Systems

In diesem Abschnitt soll auf die Implementierung und auf den Test des erweiterten Systems eingegangen werden. Dabei wird auf die im Abschnitt 5.5.1 vorgestellten Grundlagen zurückgegriffen.

5.8.1 Implementierung

Im folgenden wird die Implementierung des erweiterten Systems in der Sprache ProSet betrachtet. Eine Transformation des ProSet-Codes in den Code einer anderen Programmiersprache wäre bei der Entwicklung eines Systems, das in der Praxis tatsächlich eingesetzt werden soll, sinnvoll. Eine solche Transformation wird im Rahmen dieser Diplomarbeit jedoch nicht mehr betrachtet. Argmente für Transformation des ProSet-Codes in eine andere Programmiersprache wurden bereits im Abschnitt 5.5.1 für das Kernsystem vorgestellt. Die aufgeführten Argumente gelten analog für das erweiterte System.

Bei der Implementierung des erweiterten Systems in ProSet wird auf die ProSet-Implementierung des Kernsystems zurückgegriffen. Anschließend wird diese Implementierung erweitert. Auf die Erweiterungen soll im folgenden eingegangen werden:

- Für jede Klasse eines Entwurfsmodellobjektes, das noch nicht im Entwurfsmodell des Kernsystems modelliert ist, wird ein neuer ProSet-Modul implementiert. Dabei handelt es sich um Moduln für die Klassen der in der Tabelle 5.8 (siehe Abschnitt 5.7.3) aufgeführten Entwurfsmodellobjekte. Jeder neu implementierte ProSet-Modul erhält den Namen des Entwurfsmodellobjektes, dessen Klasse er repräsentiert. Instanzen der Moduln repräsentieren die Entwurfsmodellobjekte. Stimuli, die zwischen Entwurfsmodellobjekten ausgetauscht werden, werden in der ProSet-Implementierung mit Hilfe von Prozeduraufrufen und return-Befehlen nachgebildet (siehe auch Abschnitt 4.6).

  In den Moduln für die Klassen der Entwurfsmodellobjekte Termineinschraenkungen und Terminkalender müssen zusätzlich die Attribute, die im Entwurfsmodell zu diesen Objekten skizziert sind, berücksichtigt werden (siehe Abbildung 5.16 im Abschnitt 5.7.3). Hierzu wird in den Moduln Termineinschraenkungen und Terminkalender für jedes benötigte Attribut jeweils eine globale Variable deklariert. Bei der Implementierung des Moduls Termineinschraenkungen handelt es sich dabei um Variablen der ProSet-Datentypen Integer und Boolean. Bei der Implementierung des Moduls Terminkalender handelt es sich dabei eine Variable des ProSet-Datentyps Set, mit dessen Hilfe eine endliche Abbildung realisiert wird. In dieser endlichen Abbildung kann über die Jahreszahl auf den zu diesem Jahr gespeicherten Terminkalender zugegriffen werden. Der Terminkalender eines Jahres wird nach den Vorgaben des Analysemodells selbst wieder mit Hilfe einer endlichen Abbildung realisiert, in der zu jeder Person eine Liste aller geplanten Termine gespeichert ist (siehe Abbildung 5.14 und Beschreibung des Objektes Terminkalender im Abschnitt 5.7.2).

- Der von der im Abschnitt 5.5.1 vorgestellten Implementierung des Kernsystems übernommene ProSet-Modul Dienstplan_bearbeiten, der die Klasse des Entwurfsmodellobjektes Dienstplan_bearbeiten repräsentiert, wird überarbeitet. In der überarbeiteten Version dieses Moduls wird beachtet, daß geplante Termine, die im Terminkalender eingetragen sind, bei der Erstellung eines Dienstplans berücksichtigt werden müssen und daß Änderungen von Dienstplänen Änderungen im Terminkalender bewirken können. Die Änderungen werden mit Hilfe des für das erweiterte System modifizierten Interaktionsdiagramms für den Anwendungsfall Dienstplan bearbeiten identifiziert (zur Modifikation des Interaktionsdiagramms zum Anwendungsfall Dienstplan bearbeiten siehe Abschnitt 5.7.3, Seite 134).

- Bei der Implementierung des Kernsystems wurden ProSet-Modulinstanzen, die zur Speicherung von Daten für Materialien dienen, persistent gemacht (siehe Abschnitt 5.5.1). Dabei handelte es sich um Modulinstanzen zu den Entwurfsmodellobjekten Dienstplan, Genehmigung, Personalakte und Stundenguthaben. Im erweiterten System werden zusätzlich ProSet-Modulinstanzen zu den Entwurfsmodellobjekten Termineinschränkungen und Terminkalender persistent gemacht. Diese Modulinstanzen werden im P-File Krankenhausdaten, das im Abschnitt 5.5.1 vorgestellt wurde (Seite 117), gespeichert.

  Um die Modulinstanzen persistent zu machen, werden die Hilfsprogamme termineinschraenkungen_initialisieren und terminkalender_initialisieren implementiert. Mit Hilfe dieser Hilfsprogramme können Instanzen des Moduls Termineinschraenkungen bzw. des Moduls Terminkalender erzeugt und in einem ausgewählten P-File gespeichert werden.

Die Werkzeuge, aus denen das erweiterte System besteht, die nötigen Argumente beim Aufruf der Werkzeuge und die zugehörigen Entwurfsmodellobjekte sind in der Tabelle 5.9 aufgeführt. Diese Tabelle ist eine Erweiterung der im Abschnitt 5.5.1 vorgestellten Tabelle 5.5.

5.8.2 Test

Die für das erweiterte System durchgeführten Tests entsprechen den Tests, die für das Kernsystem bereits im Abschnitt 5.5.2 beschrieben wurden. Diese Tests werden für das erweiterte System im

Programmname	Argument beim Programmaufruf	Entwurfsmodellobjekte
personaldaten_bearbeiten	Personalnummer oder 0	BS_Personaldaten_bearbeiten Personaldaten_bearbeiten
dienstplan_loeschen	Stationsbezeichnung	BS_Dienstplan_loeschen Dienstplan_loeschen
dienstplan_genehmigen	Stationsbezeichnung	BS_Dienstplan_genehmigen Dienstplan_genehmigen
dienstplan_lesen	Stationsbezeichnung	BS_Dienstplan_lesen Dienstplan_lesen
dienstplan_bearbeiten	Stationsbezeichnung	BS_Dienstplan_bearbeiten Dienstplan_bearbeiten Terminkalender_abfragen Terminkalender_anpassen
arbeitsstunden_abrechnen	Stationsbezeichnung	BS_Arbeitsstunden_abrechnen Arbeitsstunden_abrechnen
termineinschraenkungen_festlegen	—	BS_Termineinschraenkungen_festlegen Termineinschraenkungen_festlegen
terminkalender_loeschen	—	BS_Terminkalender_loeschen Terminkalender_loeschen
terminkalender_bearbeiten	—	BS_Terminkalender_bearbeiten Terminkalender_bearbeiten Termineinschraenkungen_ueberpruefen

Tabelle 5.9: Werkzeuge des erweiterten Systems und beteiligte Objekte.

folgenden noch einmal zusammengefaßt:

Beim Modultest werden zum einen die Moduln getestet, die bei der Erweiterung des Systems neu implementiert wurden. Dabei handelt es sich um die Moduln, die die Klassen der in der Tabelle 5.8 (siehe Abschnitt 5.7.3) dargestellten Entwurfsmodellobjekte repräsentieren. Zum anderen wird der Modul Dienstplan_bearbeiten noch einmal getestet, weil dieser Modul gegenüber dem Modul Dienstplan_bearbeiten des Kernsystems verändert wurde (siehe Abschnitt 5.8.1). Zum Test der Moduln Dienstplan_bearbeiten und Terminkalender_bearbeiten wird das im Abschnitt 5.5.2 beschriebene Verfahren zum Modultest noch folgendermaßen ergänzt:

Neben Moduln zur Simulation der „Materialmoduln" Dienstplan, Genehmigung, Personalakte, Stundenguthaben, Termineinschraenkungen und Terminkalender werden beim Test der Moduln Dienstplan_bearbeiten und Terminkalender_bearbeiten noch Moduln implementiert, die die Moduln für die Klassen der Entwurfsmodellobjekte Terminkalender_abfragen, Terminkalender_anpassen und Termineinschraenkungen_ueberpruefen simulieren (siehe Abbildung 5.16 im Abschnitt 5.7.3).

Nachdem die einzelnen Moduln des erweiterten Systems getestet sind, werden die einzelnen Anwendungsfälle getestet. Diese Tests beschränken sich auf die Tests der Anwendungsfälle, die zusätzlich zu den Anwendungsfällen des Pilotsystems aufgenommen wurden und auf den Test des Anwendungsfalls Dienstplan bearbeiten. Anwendungsfälle, die durch „extends"-Beziehungen miteinander verbunden sind, werden dabei jeweils gemeinsam getestet. Abschließend wird ein Test des vollständigen erweiterten Systems durchgeführt.

5.9 Fazit

Im folgenden werden die Ergebnisse der Bearbeitung des betrachteten Fallbeispiels zusammengefaßt:

- Analog zum System zur Bibliotheksverwaltung konnten in den betrachteten Problembereichsmodellen (siehe Abschnitte 5.4.1 und 5.7.1) keine Werkzeuge aufgenommen werden, weil in der aus Zeitgründen nur oberflächlich durchgeführten Ist-Analyse keine Werkzeuge identifiziert wurden. Auf die weitere Systementwicklung hat sich das Fehlen der Werkzeuge im Problembereichsmodell jedoch nicht negativ ausgewirkt. Im Gegensatz zum System zur Bibliotheksverwaltung konnten anhand der Ist-Analyse sämtliche Materialien identifiziert werden, die für die Systementwicklung benötigt wurden (vgl. Abschnitt 3.6).

- Während in den Problembereichsmodellen des Systems zur Bibliotheksverwaltung und des E-Mail-Systems (siehe Abschnitte 3.4.1 und 4.4.2) noch statische Beziehungen zwischen Problembereichsmodellobjekten dargestellt wurden, wurde auf die Darstellung der statischen Beziehungen in den in diesem Kapitel betrachteten Problembereichsmodellen verzichtet (siehe Abschnitte 5.4.1 und 5.7.1). Dadurch wurde die Erstellung der Problembereichsmodelle vereinfacht. Der Verzicht auf die Modellierung der statischen Beziehungen in den in betrachteten Problembereichsmodellen hat sich nicht negativ auf die weitere Systementwicklung ausgewirkt.

- Die erstellten Objectory-Modelle des erweiterten Systems basieren auf den Modellen des zunächst erstellten Kernsystems. Diese Modelle wurden für die Entwicklung des erweiterten Systems ergänzt. Bei der Erweiterung der bestehenden Modelle traten keine Probleme auf.

  Auch die nachträgliche Erweiterung des bereits im Anwendungsfallmodell des Kernsystems modellierten Anwendungsfalls Dienstplan bearbeiten durch die zusätzlichen Anwendungsfälle Terminkalender abfragen und Terminkalender anpassen (siehe Abbildung 5.12 im Abschnitt 5.7.1) bereitete keine Schwierigkeiten. Bei der Überarbeitung des Interaktionsdiagramms zum Anwendungsfall Dienstplan bearbeiten waren nur einige Ergänzungen in dem aus dem Entwurfsmodell des Kernsystems übernommenen Interaktionsdiagramm nötig (siehe hierzu Abschnitt 5.7.3, Seite 134).

  Bei der Überarbeitung des ProSet-Moduls Dienstplan_bearbeiten (siehe Abschnitt 5.8.1, Seite 135) reichte es ebenfalls aus, zusätzliche Codezeilen in den vom Pilotsystem übernommenen Modul einzufügen. Eine neue Implementierung des ProSet-Moduls war nicht erforderlich.

- Da das System zur Pflegedienst- und Terminplanung aus relativ kleinen Programmen besteht, ergaben sich im Gegensatz zur Erstellung des Prototyps des Systems zur Bibliotheksverwaltung keine Probleme durch die Beschränkung des Umfangs von Moduln, in denen persistente Variablen deklariert sind und auf die das closure-Konstrukt angewendet wird (vgl. Abschnitt 3.6). Alle zum System zur Pflegedienst- und Terminplanung erstellten Programme konnten durch den ProSet-Compiler übersetzt werden.

- Obwohl die Modellierung von Attributen im Entwurfsmodell von der Modellierung der Attribute im Analysemodell abweicht, wurden beide Modellierungen für die Festlegung der benötigten Datenstrukturen in den implementierten ProSet-Moduln benötigt. Dies wurde im Abschnitt 5.5.1 anhand des Moduls Personalakte veranschaulicht (Seite 117f.).

- Die Verwendung von Atomen innerhalb einer persistenten ProSet-Modulinstanz führt zu einem Fehler bei der Ausführung eines Programms, wenn mit Hilfe der Atome auf Einträge in einer endlichen Abbildung zugegriffen werden soll (siehe Abschnitt 5.5.1). Die Fehlermeldung Exception type_mismatch could not be handled! wird ausgegeben.

Kapitel 6

Abschließende Betrachtungen

Die wesentlichen Ergebnisse der Arbeiten an den betrachteten Fallbeispielen wurden bereits in den Abschnitten 3.6, 4.7 und 5.9 zusammengefaßt. Im folgenden werden noch einige zusätzliche Bemerkungen und Ausblicke zu den durchgeführten Arbeiten aufgeführt und die geforderten abschließenden Bewertungen (siehe Aufgabenstellung, Abschnitt 1.2) vorgenommen.

Hierzu wird im Abschnitt 6.1 auf Objectory eingegangen. Anschließend wird im Abschnitt 6.2 die Ergänzung des Objectory-Entwicklungsprozesses durch Konzepte des WAM-Ansatzes betrachtet. Im Abschnitt 6.3 wird auf die Erstellung der geforderten Prototypen in der Sprache ProSet eingegangen. Abschließend sind im Abschnitt 6.4 noch einige statistische Angaben zu den bearbeiteten Fallbeispielen aufgeführt.

6.1 Die Arbeit mit Objectory

Bei allen betrachteten Beispielen erwiesen sich die in Objectory vorgesehenen Konzepte als geeignet, um die betrachteten Systeme zu modellieren. Bei der Erstellung der Objectory-Modelle ergaben sich keine Schwierigkeiten. Obwohl das Werkzeug Objectory SE für die Arbeiten am zweiten und am dritten Fallbeispiel nicht mehr zur Verfügung stand, ließen sich auch bei diesen Fallbeispielen alle geforderten Objectory-Modelle ohne größere Schwierigkeiten erstellen. Dabei sollte jedoch beachtet werden, daß die in der Diplomarbeit betrachteten Fallbeispiele im Vergleich zu Projekten, in denen Software-Systeme erstellt werden, die in der Praxis tatsächlich eingesetzt werden sollen, einen relativ geringen Umfang haben. Bei größeren Projekten, an denen mehrere Personen arbeiten, wäre ein Werkzeug wie Objectory SE sicher nötig, um die Konsistenz der Modelle, an denen verschiedene Personen arbeiten, zu erhalten.

Zur Modellierung der Daten wurden die Objectory-Analysemodelle genutzt. Ein Objectory-Analysemodell dient zwar in erster Linie dazu, den Aufbau des zu entwickelnden Systems aus verschiedenen Objekten zu beschreiben. Die zur Verfügung gestellten Konzepte erwiesen sich jedoch auch für eine Modellierung von Daten als geeignet. Datenobjekte wurden in den Analysemodellen der Diplomarbeit durch Entitätsobjekte repräsentiert.

Prototyping wird von Jacobson et al. ausdrücklich zur Ergänzung des Objectory-Entwicklungsprozesses vorgeschlagen [JCJÖ93, S. 25f.]. Konkrete Sprachen oder Umgebungen für das Prototyping werden dabei jedoch nicht empfohlen. Mit der in dieser Diplomarbeit betrachteten Prototyping-Sprache ProSet war es möglich, sämtliche geforderten Prototypen auf der Basis von Objectory-Modellen zu erstellen. Dabei war der Aufwand zur Abbildung von Objectory-Modellen auf ProSet-Prototypen bei den betrachteten Fallbeispielen unterschiedlich. Hierauf wird im Abschnitt 6.3 näher eingegangen. Als sehr hilfreich für die Erstellung von Prototypen auf der Basis von Objectory-Modellen erwies sich die Möglichkeit, im Entwurfsmodell berücksichtigen zu können, daß nicht sämtliche objektorientierten Konzepte, wie z.B. Beziehungen zwischen Klassen, auf ProSet abgebildet werden können.

Interessant wäre eine Betrachtung, inwieweit andere Methoden zur objektorientierten Analyse und zum objektorientierten Entwurf für die Systementwicklung in den betrachteten Fallstudien besser oder schlechter geeignet sind als Objectory. Eine solche Betrachtung ist jedoch nicht mehr Bestandteil dieser Arbeit. Für eine Gegenüberstellung verschiedener objektorientierter Methoden sei auf Wilkie [Wil93, S. 106ff.] und auf Stein [Ste93] verwiesen.

6.2 Ergänzung von Objectory durch Konzepte des WAM-Ansatzes

In dieser Arbeit wurden die folgenden Konzepte des WAM-Ansatzes (siehe Abschnitt 2.2) berücksichtigt:

Erstellung eines Glossars: Zu jeder betrachteten Fallstudie wurde ein Glossar erstellt. Die Glossare befinden sich im Zusatzdokument zu dieser Diplomarbeit.

Die erstellten Glossare haben einen geringeren Umfang als Glossare, die bei Projekten zur Entwicklung von Anwendungssystemen für „echte“ Endanwender, erstellt werden. Dies liegt daran, daß ein Großteil der Einträge in einem Glossar bei der Ist-Analyse gefunden wird (siehe hierzu Kilberth et al. [KGZ95, S. 98f.]) und die in dieser Diplomarbeit durchgeführten Ist-Analysen nur oberflächlich duchgeführt werden konnten. In der Diplomarbeit haben die Glossare nur eine geringe Bedeutung für die Bearbeitung der Fallbeispiele. Dies liegt daran, daß an der Bearbeitung der Fallbeispiele dieser Diplomarbeit im Gegensatz zu „realen“ Projekten zur Software-Entwicklung nicht mehrere Personen beteiligt sind, die sich auf eine gemeinsame Projektsprache (siehe Abschnitt 2.2) verständigen müßten.

Erstellung von Systemvisionen: Die Systemvisionen entsprechen den Anwendungsfällen in den erstellten Objectory-Anwendungsfallmodellen.

Die Systemvisionen (bzw. Anwendungfälle) sind bei den betrachten Fallstudien wesentliche Bestandteile der Spezifikation der Anforderungen an die zu modellierenden Systeme.

Verwendung der Metaphern Werkzeug und Material: In den meisten Fällen konnten die Objekte in den Objectory-Modellen mit Hilfe der Metaphern Werkzeug und Material beschrieben werden. In einigen Fällen bei der Modellierung des Systems zur Bibliotheksverwaltung und bei der Modellierung des E-Mail-Systems gab es jedoch auch Schwierigkeiten (z.B. bei der Einteilung des Transferagenten des E-Mail-Systems). Diese Probleme wurden bereits in den Abschnitten 3.6 und 4.7 zusammengefaßt und sollen hier nicht wiederholt werden.

Eine abschließende Bewertung der Einbeziehung von Konzepten des WAM-Ansatzes in den Objectory-Entwicklungsprozeß auf der Grundlage der betrachteten Fallstudien ist aus den folgenden Gründen schwierig:

- Der Umfang der in der Diplomarbeit betrachteten Fallbeispiele ist erheblich geringer als der Umfang von Projekten, für die der WAM-Ansatz konzipiert wurde (siehe hierzu Bäumer et al. [BGZ95]). Bei den Fallbeispielen, die in der Diplomarbeit betrachtet wurden, wäre die Erstellung der geforderten Objectory-Modelle und der Prototypen auch ohne eine Berücksichtung von Konzepten des WAM-Ansatzes möglich gewesen, ohne daß sich der Aufwand erhöht hätte. Bei größeren Projekten könnten die betrachteten Konzepte des WAM-Ansatzes jedoch geeignete Hilfsmittel sein, um den Entwurf von Software-Systemen übersichtlicher zu gestalten und so zu vereinfachen.
- Im WAM-Ansatz ist eine ausführliche Ist-Analyse des betrachteten Anwendungsbereichs vorgesehen, in der einzelne Szenarien (siehe Abschnitt 2.2) beschrieben werden. Da die Ist-Analyse durch Objectory nicht unterstützt wird, kann insbesondere eine Ist-Analyse, wie sie im WAM-Ansatz vorgesehen ist, eine sinnvolle Ergänzung zu Objectory sein. Im Rahmen der Diplomarbeit konnten Ist-Analysen aus Zeitgründen jedoch nur verkürzt durchgeführt werden. Einzelne Szenarien wurden hier nicht beschrieben. Die Vorteile, die die Integration einer

Ist-Analyse nach dem WAM-Ansatz in den Objectory-Entwicklungsprozeß möglicherweise bringen würde, können in dieser Diplomarbeit daher nicht betrachtet werden.

- Die Verwendung der Metaphern Werkzeug und Material soll zur Entwicklung von Software-Systemen dienen, die die Anwender bei ihrer Arbeit unterstützen, ohne ihnen die Kontrolle über den Ablauf von Arbeitsvorgängen zu entziehen. Gryczan und Züllighoven erwarten hiervon eine Verringerung von Akzeptanzproblemen der Endanwender bei der Einführung von Software-Systemen [GZ92]. Die Arbeit von Endanwendern mit entworfenen Software-Systemen wird in dieser Diplomarbeit jedoch nicht betrachtet. Eine Aussage, ob durch die Verwendung des WAM-Ansatzes bei den Entwicklung von Software-Systemen tatsächlich Akzeptanzprobleme bei Anwendern verringert werden, ist im Rahmen dieser Arbeit deshalb nicht möglich.

Für eine fundierte Aussage, ob die Integration von Konzepten des WAM-Ansatzes in den Objectory-Entwicklungsprozeß Vorteile beim Entwurf von Software-Systemen bietet, wären deshalb weitere Studien nötig, in denen die oben beschriebenen Beschränkungen der in der Diplomarbeit betrachteten Fallbeispiele nicht mehr bestehen. Als Ergebnis der Diplomarbeit in Bezug auf die Verwendung von Konzepten des WAM-Ansatzes im Objectory-Entwicklungsprozeß kann festgehalten werden, daß sich die betrachteten Konzepte des WAM-Ansatzes problemlos in die zu den einzelnen Fallbeispielen betrachteten Objectory-Entwicklungsprozesse integrieren ließen.

6.3 Prototyping mit ProSet

Sämtliche geforderten Prototypen konnten mit Hilfe von ProSet realisiert werden (bei der Modellierung des E-Mail-Systems wurden zusätzlich noch C-Funktionen für die Interprozeßkommunikation benutzt). Die Implementierung der Prototypen auf der Basis erstellter Objectory-Modelle erwies sich bei den betrachteten Fallbeispielen als unterschiedlich schwierig. Während bei der Erstellung des ProSet-Prototyps zum ersten Fallbeispiel (siehe Kapitel 3) noch viele Probleme auftraten, gab es bei der Implementierung der zu den anderen Fallbeispielen geforderten ProSet-Programme keine Schwierigkeiten.

Dies drückt sich vor allem in den Zeitdauern aus, die für die Implementierung der geforderten ProSet-Programme benötigt wurden. Beim ersten Fallbeispiel (siehe Kapitel 3) wurden ca. neun Wochen für die Implementierung des Prototyps benötigt, nachdem die geforderten Objecotory-Modelle bereits erstellt waren (bei einer durchschnittlichen Arbeitszeit von ca. 40 Stunden pro Woche). Bei den anderen beiden Fallbeispielen (siehe Kapitel 4 und 5) beschränkte sich die Zeit, die für die Implementierung von ProSet-Programmen aufgewendet werden mußte, auf jeweils ca. drei Wochen pro Fallbeispiel. Für diese unterschiedlichen Zeitdauern können die folgenden Gründe aufgeführt werden:

- Der ProSet-Persistenzmechanismus funktionierte während der Einarbeitungsphase zu dieser Diplomarbeit noch nicht. Deshalb konnten vor der Bearbeitung des ersten Fallbeispiels noch keine Erfahrungen mit der Benutzung des ProSet-Persistenzmechanismus gesammelt werden.

  Bei der Implementierung des Prototyps des Systems zur Bibliotheksverwaltung traten dadurch unerwartete Schwierigkeiten auf. Diese Schwierigkeiten, wie z.B. das Auftreten des Fehlers internal assembler error: out of memory bei der Übersetzung umfangreicher ProSet-Programme, in den das closure-Konstrukt auf Moduln angewendet wird, in denen persistente Variablen deklariert sind, wurden bereits im Abschnitt 3.6 zusammengefaßt.

  Bei der Implementierung von ProSet-Programmen für die anderen betrachteten Fallbeispiele konnte bereits auf die Erfahrungen, die bei der Bearbeitung des ersten Fallbeispiels mit dem ProSet-Compiler gemacht wurden, zurückgegriffen werden. Schwierigkeiten konnten umgangen werden.

- Bei der Bearbeitung des ersten Fallbeispiels traten noch Fehler im benutzten ProSet-Compiler und im H-PCTE-Server auf, von denen die meisten parallel zur Bearbeitung des ersten Fallbeispiels durch Mitglieder der ProSet-Entwicklungsgruppe behoben wurden. Der ProSet-Compiler und der H-PCTE-Server liefen bei der Bearbeitung der anderen beiden betrachteten Fallbeispiele bereits wesentlich stabiler (siehe auch Abschnitt 4.7).

- Bei der Bearbeitung erwies sich das erste Fallbeispiel als umfangreicher als ursprünglich angenommen wurde.

- Bei der Bearbeitung des ersten Fallbeispiels mußten noch Vererbungsbeziehungen auf ProSet abgebildet werden. Die Abbildung von Beziehungen zwischen Klassen wird durch ProSet jedoch nicht unterstützt.

  Bei der Erstellung der ProSet-Programme, die zu den anderen beiden Fallbeispielen gefordert wurden, konnte im Gegensatz zum ersten Fallbeispiel im Objectory-Entwurfsmodell berücksichtigt werden, daß Klassenbeziehungen nicht direkt auf ProSet-Programme abgebildet werden können. Die Erstellung der ProSet-Programme war hier unproblematisch.

In einigen Fällen wäre es bei den vorgestellten Fallbeispielen zur Verkürzung der Zeitdauer des Zugriffs auf benötigte persistente Daten sowie zur Verbesserung der Möglichkeiten zum parallelen Datenzugriff von Vorteil gewesen, wenn entweder die Namen von P-File-Einträgen, auf die in einem ProSet-Programm zugegriffen werden soll, dynamisch zur Laufzeit des Programms hätten festgelegt werden können oder wenn ein Zugriff auf einzelne Teile eines P-File-Eintrags anstelle eines vollständigen P-File-Eintrags möglich wäre. Auf die Schwierigkeiten, die durch die statische Festlegung von Variablennamen bzw. durch die Vorgabe, immer auf vollständige P-File-Einträge zugreifen zu müssen, entstehen können, wurde z.B. bei der Beschreibung des Entwurfsmodellobjektes Personalakte im Kapitel zum dritten betracheten Fallbeispiel eingegangen (siehe Abschnitt 5.4.3, Seite 114f.). Diese Erläuterungen sollen hier nicht wiederholt werden.

Des weiteren wäre es bei einer zukünftigen Erweiterung von ProSet sinnvoll, Operationen bereitzustellen, die das Erzeugen und das Löschen von P-Files aus ProSet-Programmen heraus ermöglichen. Bisher ist das Erzeugen und das Löschen von P-Files nur mit Hilfe von externen Werkzeugen möglich. Durch die Möglichkeit, P-Files direkt aus ProSet-Programmen heraus zu erzeugen, könnte z.B. die Installation eines Prototyps vereinfacht werden. Es wäre möglich, ein ProSet-Programm zu implementieren, das sämtliche P-Files, die von einem Prototyp benötigt werden, erzeugt und initialisiert. Das Löschen von P-Files aus ProSet-Programmen heraus könnte beispielsweise genutzt werden, um die Deinstallation eines Prototyps zu vereinfachen. Es wäre möglich, ein ProSet-Programm zu implementieren, das alle nicht mehr benötigten P-Files löscht, wenn die mit einem Prototyp nicht mehr gearbeitet werden soll.

Bei einer abschließenden Betrachtung, ob es sinnvoll war, die Sprache ProSet zur Erstellung von Prototypen auf der Basis von Objectory-Modellen zu nutzen, sollte zwischen den einzelnen Fallbeispielen differenziert werden. So wäre zur Erstellung des Prototyps des Systems zur Bibliotheksverwaltung eine objektorientierte Sprache oder eine objektorientierte Erweiterung von ProSet besser geeignet gewesen als die verwendete ProSet-Version, weil die fehlende Möglichkeit, Vererbungsbeziehungen in ProSet-Programmen zu verwenden, die Erstellung des Prototyps behindert hat. Bei den anderen betrachteten Fallbeispielen bereitete die Erstellung der ProSet-Prototypen hingegegen keine Schwierigkeiten. Die ProSet-Prototypen konnten sicher schneller implementiert werden als vergleichbare Programme in konventionellen Programmiersprachen, wie z.B. Pascal oder C. Die Verwendung der Sprache ProSet zur Erstellung von Prototypen hat sich hier als sinnvoll erwiesen.

Sinnvoll wäre noch eine abschließende Bewertung der Güte der Prototypen. Da bei dieser Arbeit jedoch kein Maß zur Bestimmung der Güte eines Prototyps vorgegeben wurde, wird diese Bewertung in der Diplomarbeit nicht vorgenommen.

Des weiteren wäre eine Betrachtung, inwieweit erstellte ProSet-Prototypen die Kooperation zwischen Anwendern und Entwicklern unterstützen können, interessant. Die Betrachtung der Kooperation zwischen Anwendern und Entwicklern eines Software-Systems ist jedoch kein Bestandteil der Aufgabenstellung dieser Diplomarbeit und wurde deshalb nicht betrachtet. Eine solche

Betrachtung könnte Thema nachfolgender Studien oder Diplomarbeiten sein. Hier wäre die Frage interessant, inwiefern Prototypen, die im Rahmen des Prototyping mit ProSet erstellt werden, dazu beitragen, die Anforderungen an ein System bzw. an eine Erweiterung eines Systems zwischen Anwendern und Entwicklern festzulegen.

Ein weiteres Thema für weiterführende Betrachtungen könnte die Automatisierung der Abbildung von Objectory-Modellen auf ProSet-Programme sein. Hier könnten Möglichkeiten zur automatischen Generierung von ProSet-Code oder zumindest von ProSet-Coderahmen auf der Basis von Objectory-Modellen untersucht werden.

Abschließend soll noch auf die Hilfsmittel eingegangen werden, die für die Arbeit mit ProSet benutzt wurden. Dabei handelt es sich um die folgenden Komponenten:

- der ProSet-Compiler in der Version 0.6,
- der in der Diplomarbeit von Kappert [Kap95] vorgestellte H-PCTE-Server,
- die graphische ProSet-Compiler-Benutzeroberfläche xpst,
- die Hilfsprogramme xpfed und xpft zum Anlegen, Bearbeiten und Löschen von P-Files.

Auf diese Hilfsmittel wird im folgenden kurz eingegangen:

Der Compiler und der H-PCTE-Server: Auf den Compiler und auf den H-PCTE-Server soll gemeinsam eingegangen werden, weil von außen nicht immer erkennbar ist, ob aufgetretene Schwierigkeiten durch den Compiler oder durch den H-PCTE-Server verursacht werden.

Sowohl der ProSet-Compiler als auch der H-PCTE-Server liefen nach den Fehlerkorrekturen, die parallel zur Bearbeitung des ersten Fallbeispiels durch Mitglieder der ProSet-Entwicklergruppe durchgeführt wurden, relativ stabil. Die folgenden Fehler treten jedoch weiterhin auf:

- Wenn Ausnahmen auftreten, die sich auf persistente Variablen oder persistente Konstanten beziehen, die lokal in Prozeduren deklariert sind, wird das Programm abgebrochen. Ausnahme-Handler werden dabei nicht korrekt ausgeführt (siehe Abschnitt 3.6).
- Wenn beim Versuch, auf eine Datenbank zuzugreifen, ein Deadlock auftritt, wird die Ausnahmemeldung p_fatal_error anstelle der erwarteten Ausnahmemeldung p_deadlock_occured erzeugt (siehe Abschnitt 3.6).
- Die Verwendung von Atomen innerhalb einer persistenten ProSet-Modulinstanz führt zu einem Fehler bei der Ausführung eines Programms, wenn mit Hilfe der Atome auf Einträge in einer endlichen Abbildung zugegriffen werden soll (siehe Abschnitt 5.9).

Des weiteren wäre es sinnvoll, den Persistenzmechanismus bei einer Nachfolgerversion des verwendeten Compilers so zu überarbeiten, daß die Übersetzung umfangreicher Programme, in denen das closure-Konstrukt auf Moduln angewendet wird, in deren Prozeduren persistente Variablen deklariert sind, nicht mit der Fehlermeldung internal assembler error: out of memory abgebrochen wird (siehe Abschnitt 3.6).

Die graphische Oberfläche zum Compiler: Die graphische Oberfläche lief während der gesamten Zeit stabil und hat die Arbeit mit dem ProSet-Compiler insbesondere während der Einarbeitungsphase erleichtert.

Die Hilfsprogramme xpfed und xpft: Auch diese Hilfsprogramme liefen während der gesamten Zeit stabil und konnten einfach bedient werden.

Verwirrend ist bei diesen Programmen noch die Zuordnung von Scroll-Balken zu Anzeigefeldern. Die Scroll-Balken beziehen sich zum Teil auf andere Anzeigefelder, als dies intuitiv erwartet worden wäre.

6.4 Statistische Angaben

Im folgenden sind noch einige statistische Angaben zu den Objectory-Modellen und zu den ProSet-Implementierungen aufgeführt. Diese Angaben sollen die Diplomarbeit vervollständigen. Die Zahlen werden hier nicht interpretiert. Bei einem Vergleich der Zahlen sollten in jedem Fall die Beschreibungen der Arbeiten zu den einzelnen Fallbeispielen (Kapitel 3 bis 5) zusätzlich betrachtet werden.

	erstes Fallbeispiel	zweites Fallbeispiel	drittes Fallbeispiel
Anzahl der Anwendungsfälle	32	12	12
Anzahl der Objekte im Problembereichsmodell	22	17	6
Anzahl der Entitätsobjekte im Analysemodell	21	13	6
Anzahl der Kontrollobjekte im Analysemodell	—	12	12
Anzahl der Schnittstellenobjekte im Analysemodell	—	4	9
Anzahl der Objekte im Entwurfsmodell	—	18	27
Anzahl der ProSet-Moduln	23	18	27
Anzahl der Codezeilen der Implementierung	13602	3398	4959

Tabelle 6.1: Statistische Angaben.

Literaturverzeichnis

[BBP90] Robert Babatz, Manfred Bogen und Uta Pankoke-Babatz, *Elektronische Kommunikation — X.400 MHS*, Friedr. Vieweg & Sohn Verlagsgesellschaft, Braunschweig, 1990.

[BGZ95] Dirk Bäumer, Guido Gryczan und Heinz Züllighoven, "Objektorientierte Software-Entwicklung für Banken — Methodik und Erfahrungen aus einer mehrjährigen Projektpraxis", *OBJEKTspektrum*, (3):45–53, 1995.

[BKKZ92] Reinhard Budde, Karlheinz Kautz, Karin Kuhlenkamp und Heinz Züllighoven, *Prototyping: An Approach to Evolutionary System Development*, Springer-Verlag, Berlin, Heidelberg, New York, 1992.

[BP92] Walter R. Bischofberger und Gustav Pomberger, *Prototyping-Oriented Software Development — Concepts and Tools*, Springer–Verlag, Berlin, Heidelberg, New York, 1992.

[DF89] Ernst-Erich Doberkat und Dietmar Fox, *Software Prototyping mit SETL*, Leitfäden und Monographien der Informatik, Teubner-Verlag, Stuttgart, 1989.

[DFG+92] Ernst-Erich Doberkat, Wolfgang Franke, Ulrich Gutenbeil, Wilhelm Hasselbring, Ulrich Lamers und Claus Pahl, "ProSet — A Language for Prototyping with Sets", In N. Kanopoulos, Hrsg., *Third International Workshop on Rapid System Prototyping*, Seiten 235 – 248, 1992.

[DFH+95] Ernst-Erich Doberkat, Wolfgang Franke, Wilhelm Hasselbring, Claus Pahl, Hans-Gerald Sobottka und Bettina Sucrow, "ProSet — Prototyping with Sets Language Definition", Technischer Bericht, Lehrstuhl Software-Technologie, Fachbereich Informatik, Universität Dortmund, Juli 1995.

[DFKS93] Ernst-Erich Doberkat, Wolfgang Franke, Udo Kelter und Wolfgang Seelbach, "Verwaltung persistenter Daten in einer Prototyping-Umgebung", In *Requirements Engineering '93: Prototyping*, Seiten 147–164, Teubner, 1993.

[Flo84] Chistiane Floyd, "A Systematic Look at Prototyping", In Reinhard Budde, Karin Kuhlenkamp, Lars Mathiassen und Heinz Züllighoven, Hrsg., *Approaches to Prototyping*, Seiten 1 – 18, Berlin, Heidelberg, New York, Tokyo, 1984, Springer-Verlag.

[GAJ+94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek und Vaidy Sunderam, *PVM 3 User's Guide and Reference Manual*, September 1994.

[GLV92] Sophie Gamerman, Catherine Lanquette und Fernando Vélez, "Using Database Applications to Compare Programming Languages", In F. Bancilhon, C. Delobel und P. Kanellakis, Hrsg., *Building an Object-Oriented Database System — The Story of O_2*, Kapitel 13, Morgan Kaufmann, San Mateo, California, 1992.

[GZ92] Guido Gryczan und Heinz Züllighoven, "Objektorientierte Systementwicklung: Leitbild und Entwicklungsdokumente", *Informatik-Spektrum*, 15:264–272, 1992.

[JCJÖ93] Ivar Jacobson, Magnus Christerson, Patrik Jonsson und Gunnar Övergard, *Object-Oriented Software Engineering*, Addison Wesley, 1993, Revised fourth printing.

[Kap95] Christian Kappert, “Integration von Persistenzkonzepten in eine Prototyping-Sprache und Realisierung mit Hilfe eines Nicht-Standard-Datenbanksystems”, 1995.

[KGZ95] Klaus Kilberth, Guido Gryczan und Heinz Züllighoven, *Objektorientierte Anwendungsentwicklung — Konzepte, Strategien, Erfahrungen*, Vieweg, 1994.

[KLSZ92] Antoinette Kieback, Horst Lichter, Matthias Schneider-Hufschmidt und Heinz Züllighoven, “Prototyping in industriellen Software-Projekten”, *Informatik-Spektrum*, 15:65–77, 1992.

[KR70] Brian W. Kernighan und Dennis M. Ritchie, *The C-Programming Language*, Prentice Hall, Englewood Cliffs, New Jersey, second edition, 1983.

[Ory94a] Objectory AB, *Objectory – Introduction*, 3.5 edition, 1994.

[Ory94b] Objectory AB, *Objectory – Requirements Analysis*, 3.5 edition, 1994.

[Ory94c] Objectory AB, *Objectory – Robustness Analysis*, 3.5 edition, 1994.

[Ory95] Objectory AB, *Objectory SE – Reference Manual*, 3.6 edition, 1995.

[Ste93] Wolfgang Stein, “Objektorientierte Analysemethoden — ein Vergleich”, *Informatik-Spektrum*, 16:317–332, 1993.

[Tan92] Andrew S. Tanenbaum, *Computer-Netzwerke*, Wolfram's Verlag, Attenkirchen, second edition, 1992.

[Wil93] George Wilkie, *Object-Oriented Software Engineering: The Professional Developer's Guide*, Addison Wesley, 1993.