Entwicklung und Ergebnisse eines allgemeingültigen Fachkonzeptes für die Prüfungsverwaltung an Hochschulen

Thorsten Spitta, Jörn Mordau
Universität Bielefeld
Fakultät für Wirtschaftswissenschaften[1]

Zusammenfassung:
Mit „Eckdatenverordnungen“ oder auch neuen Studienmodellen sollen überlange Studienzeiten verringert werden. Beide „Lösungen“ - nehmen wir einmal an, sie seien dies - erfordern eine bessere administrative Unterstützung der Dekanate und vor allem der Prüfungsämter, als derzeit existierende Software sie bietet. Die Autoren sehen die Hochschulen in der Pflicht, hier für Abhilfe zu sorgen. Der vorliegende Aufsatz versteht sich als ein Beitrag hierzu. Es wird der Rahmen für den fachlichen Entwurf einer universell verwendbaren Prüfungsverwaltungssoftware skizziert. Die hier vorgestellten Teile umfassen vor allem das Datenmodell und Auszüge aus einer formalen Spezifikation, mit der das Datenmodell verifiziert wurde.

1 Problemstellung und Überblick

Die Verwaltung von Prüfungsämtern in Hochschulen ist ein Softwareproblem, das nach Wissen der Autoren vielen Hochschulen Schwierigkeiten bereitet. Im Zuge staatlich verordneter Änderungen von Prüfungsordnungen vergrößert sich das Problem noch. Ab einer bestimmten Anzahl von (Zwischen-)Prüfungen ist es mit vertretbarem Personalaufwand nicht mehr möglich, ein Prüfungsamt ohne Software zu verwalten, weshalb wir von einem *Software*problem sprechen. Zwei Wege, das Problem zu lösen, werden praktiziert:

1. Ein Hochschulangehöriger schreibt ein Programmsystem, das so lange zur Zufriedenheit aller genutzt werden kann, wie der Entwickler die Software selbst wartet.
2. Es wird ein Standardpaket eingesetzt, das einer Releasepflege durch eine professionelle Organisation unterliegt. Unseres Wissens gibt es nur ein solches Paket, das System POS der Firma HIS (Hochschul-Informationssystem) GmbH, deren Gesellschafter zu einem Drittel der Bund und zu zwei Dritteln die Länder sind. Das System wird nach Aussage der Fa. HIS von ca. 10 Fakultäten produktiv genutzt, darunter auch der, der die Autoren angehören.

Zumindest wegen der Personenabhängigkeit von 1. halten wir den zweiten Weg für den sinnvolleren, zumal er allein den Prinzipien entspricht, die in der Betriebswirtschaftslehre, Wirtschaftsinformatik und Informatik vertreten werden, nämlich ein Standardproblem wiederverwendbar und standardisiert zu lösen. Dies bedeutet nicht, daß wir glauben, das Monopolprodukt HIS-POS könne in der gegenwärtigen Form anderen Fakultäten weiterempfohlen werden. Wir wollen hier jedoch nur am Rande Kritik üben. Unser Ziel ist es, einen Beitrag zu leisten, damit ein verbessertes Standardsystem entstehen kann, möglicherweise als neues Release von HIS-POS. Die Mängel von HIS-POS sind typisch für Software, die direkt beim Anwender entsteht[2]:

- Unnötig komplexe Datenbasis infolge mangelhafter oder unterlassener Datenanalyse [vgl. Vett90: „Das Jahrhundertproblem der Informatik“]. Im Fall HIS-POS fällt vor allem eine hohe Schlüsselredundanz auf, die das System undurchschaubar macht.

[1] Prof. Dr. Th. Spitta, Postfach 10 01 31, 33 501 Bielefeld; {ThSpitta | JMordau}@wiwi.uni-bielefeld.de
[2] Der erste Autor macht diese Aussage auf der Basis langjähriger Entwicklungs- und Wartungspraxis. Wissenschaftliche Untersuchungen zur Art der Fehler in Informationssystemen sind nicht bekannt [vgl. Lehn91, 44f.].

- Speicherung abgeleiteter Daten.
- Mangelhafte oder fehlende Integritätsbedingungen mit der Folge, daß die Datenbasis immer wieder inkonsistent wird. „Reparaturprogramme“, die direkt auf den Daten aufsetzen, erzeugen dann meist neue Inkonsistenzen.
- Die Benutzeroberfläche ist nicht an Vorgängen orientiert, sondern an Datenbeständen. Dies provoziert Fehler durch den Benutzer, die ebenfalls inkonsistente Daten erzeugen.
- Der Wartungsaufwand ist unverhältnismäßig hoch. In einem konkreten Fall betrug allein der Aufwand für einen Releasewechsel mehr als 10 Bearbeitertage.

Die Mängel haben eine gemeinsame Ursache, mit deren Beseitigung *jedes* Softwarehaus ohne intensive Mitarbeit der Hochschulen überfordert sein muß: Es fehlt eine gründlich recherchierte Problemanalyse, die wir in Form von Auszügen aus einem Fachkonzept hier zur Diskussion stellen wollen.

Neben dem Ziel, das Problem des Diskursbereiches einer Lösung näherzubringen, soll ein zweiter Aspekt offengelegt werden, die Entwicklungsmethodik. Es soll nachvollziehbar werden, wie die vorgelegten Ergebnisse entstanden sind. Während die gegenwärtige wissenschaftliche Diskussion beherrscht wird von Beiträgen zum fachlichen Entwurf im Großen, insbesondere der Prozeßanalyse [vgl. CoYo90, FeSi90, Öst95, Sche94], sehen wir ein Defizit in der Anwendung von Methoden zur Spezifikation und deren Einbindung in die Methoden des Grobkonzeptes. Es gibt keinen Mangel an Spezifikationsmethoden [vgl. BaWö84, Jon86, Spiv89], wohl aber in deren Nutzung. Erst bei der detaillierten Spezifikation findet man einige Abstraktionsmöglichkeiten und Strukturen, die zu einer Vereinfachung von Arbeitsergebnissen des Grobkonzeptes und damit zur Qualitätsverbesserung eines Systems führen. Einige methodische Ergebnisse meinen wir, verallgemeinern zu können, da es sich hier nicht um Beispiele, sondern um die Entwicklung eines repräsentativen Systems handelt.

Für welche Problemklasse soll ein Prüfungsverwaltungssystem repräsentativ sein? Ein solches System bietet in einem sehr kleinen Rahmen viele Fragestellungen betrieblicher Informationssysteme, und ist ein beliebtes [vgl. z.B. Vett90, Fi92] und u.E. auch gut geeignetes Beispiel, Datenmodelle zu lehren. Folgende Charakteristika betrieblicher Informationssysteme finden sich in einem solchen System:

- Datenbasis mit Stamm-, Bewegungs- und verdichteten Daten (z.B. Student, Prüfung, Notenverteilung),
- Mehrbenutzerfähigkeit,
- Arbeitsabläufe als Transaktionen, deren Konsistenz in der Datenbasis gesichert werden muß (*Geschäftsprozesse*, [vgl. FeSi95], z.B. Noten eintragen für 500 Klausuren),
- Verdichtung und/oder Auslagerung von Vorgängen nach einer gewissen Zeit (Prüfungen zurückliegender Semester),
- Mandantenfähigkeit (Einsatz für mehrere Fakultäten),
- Zusammensetzung aus Teilsystemen:
  - administrativ (Immatrikulationsbüro und Prüfungsamt),
  - regelbasiert (Prüfungsordnung),
- Benutzeroberfläche sowohl für Massenvorgänge (z.B. Anmeldungen) als auch Einzelfälle (z.B. Stammdatenpflege) und Abfragen,
- Zugriffsschutz wegen sensibler Daten,
- Schnittstelle zu Standardsystemen wie Textverarbeitung (Bescheinigungen u.ä.).

Die Programmierung einer Lösung für eine *bestimmte* Fakultät oder Prüfungsordnung übersteigt u.E. drei Bearbeitermonate nicht. Es handelt sich also um ein sehr kleines System. Ein gründlich recherchiertes Fachkonzept als Basis für eine langfristig stabile Lösung erfordert jedoch ein Mehrfaches dieses Aufwandes. In den folgenden Abschnitten werden der Entstehungsprozeß (*Projektorganisation*), die verwendete *Methodik* und beispielhafte *Ergebnisse*

des Fachkonzeptes dargestellt. Die Ergebnisse bestehen aus einem vollständigen Datenmodell, einer Liste der Arbeitsabläufe („Geschäftsprozesse“) und Beispielen aus Prozeßbeschreibungen und Spezifikationen.

2 Projektorganisation

Die bisherigen Ergebnisse wurden in zwei aufeinander folgenden Projektseminaren mit Studenten der Betriebs- und Volkswirtschaftslehre im Hauptstudium erzielt. Im ersten Semester handelte es sich um 33, im zweiten um 23 Teilnehmer. Die didaktischen Aspekte der Veranstaltung gehören nicht hierher, wohl aber die Organisation der Erstellung von Ergebnissen.

Alle Teilaufgaben waren durch die Lehrenden vordefiniert und wurden in Wiederholung zur Vorlesung noch einmal methodisch erläutert. Jede Teilaufgabe wurde durch drei Untergruppen konkurrierend erarbeitet, im Plenum vorgetragen und diskutiert. Danach hatten die Untergruppen einer Teilaufgabe eine gemeinsame Lösung zu erarbeiten, die wir hier als *Ergebnis* bezeichnen.

Wir halten diese „Projektorganisation“ für qualitätserhöhend, da ein Problem bearbeitet wurde, bei dem Studenten Betroffene sind und genügend Kenntnisse über den Diskursbereich besitzen. Die hochschulspezifische Organisation des Projektes hat mehr Ähnlichkeiten mit Praxisprojekten als Unterschiede [vgl. MaOp85]. Im einzelnen:

Tab. 1: Entwicklung unter Einbeziehung der Benutzer im Lehrbereich und im Fachbereich eines Betriebes.

Merkmal	*Student Universität*	*Mitarbeiter Fachabteilung*
zeitliche Beteiligung	nur punktuell	
Qualifikation	stark streuend	
methodische Fähigkeiten	gering	
Problemkenntnis	hoch	
Betroffenheit	hoch	
Engagement	stark streuend	
personelle Kontinuität	keine	stark
Kapazität Projektgruppe	hoch	gering
Möglichkeit konkurrierender Sichten	gut	sehr gering
Betroffenheit nach Projektabschluß	sehr gering	sehr hoch

Diese Synopse kann nur gelten, wenn ein Projekt aus dem jeweiligen Erfahrungshintergrund der Benutzer zu bearbeiten ist. Der Dozent entspricht in einem Praxisprojekt der Rolle der DV-Projektmitglieder und des Projektleiters in einer Person. Hier gibt es kaum Analogien.

Der Wechsel aller „Entwickler“ zwischen den Semestern ist zwar eine speziell hochschultypische Gegebenheit, es entstand jedoch auch hierdurch ein Entwicklungsschritt, der in Praxisprojekten stattfinden *sollte* und der Qualitätssicherungsmaßnahme *Review* entspricht [vgl. Wall90, 146ff.]. Alle in der ersten „Phase“ (Semester) erarbeiteten Ergebnisse mußten von der Folgegruppe nachvollzogen werden.

Die in Abschnitt 4 vorgestellten Ergebnisse wurden unter Bedingungen erzeugt, die eine hohe Qualität ermöglichen. Unsere Qualitätsmaßstäbe für das Fachkonzept sind *Funktionserfüllung*, *Vollständigkeit*, *Konsistenz* und *Redundanzfreiheit*.

Nun sichert eine noch so ausgefeilte Projektorganisation für sich noch keine Produktqualität. Hierzu müssen eine geeignete Entwicklungsmethodik und auch Werkzeuge kommen.

3 Entwicklungsmethodik

Die Entwicklungsmethodik wurde dem relativ einfachen Problem angemessen gewählt. Das Problem ließ erwarten, daß benötigt würden:

- wenige, leicht zu ermittelnde *Prozeßketten*,
- ein übersichtliches *Datenmodell*,
- *Funktionen* überwiegend als einfache Änderungs- oder Abfragefunktionen der Datenbasis,
- sehr wenige *Schnittstellen* zur Einbettung in die Umwelt.

Für komplexe Systeme konzipierte Ansätze wie etwa OMT (object modelling technique) von Rumbaugh [Rum+91] oder die semantische Objektmodellierung SOM von Ferstl und Sinz [FeSi90] wurden verworfen und statt dessen eher „traditionell" orientiert nach den methodisch ähnlichen Ansätzen von Spitta [Spi89] und Scheer [Sche94] vorgegangen. In beiden Ansätzen werden zunächst Sichten auf Daten, Funktionen und organisatorische Rollen isoliert modelliert und dann stufenweise zu Objekttypen und Prozeßketten integriert. Die Integration ist eher informal und muß durch Entwurfsdisziplin geleistet werden. Dies ist für die Entwicklung größerer Systeme problematisch [vgl. FeSi95], erscheint jedoch für ein kleines System wie das vorliegende unkritisch.

Der fachliche Entwurf eines Systems wird hier in zwei Schritten ausgeführt, dem *Fachkonzept* (auch *Grobkonzept*) und der *Spezifikation* (auch *Feinkonzept*). Ob man diese Ergebnisse einzelnen Phasen zuordnet oder als Teilschritte einer Phase sieht [s. PaSi94, 37], ist nachrangig. Wichtig ist jedoch, daß der verfeinernde zweite Schritt *Rückwirkungen* auch auf wesentliche Strukturen des ersten hat, man die Phasen also keinesfalls als reine Sequenz sehen darf. Dies impliziert, daß nicht streng hierarchisch entwickelt werden kann. Der Aspekt der Rückbezüge bekommt bei der hier dargestellten erstmalig angewendeten Vorgehensweise ein besonderes Gewicht.

Das folgende Bild gibt einen Überblick, in welchem Entwicklungsschritt welche Ergebnisse entstehen und wie sie ineinandergreifen.

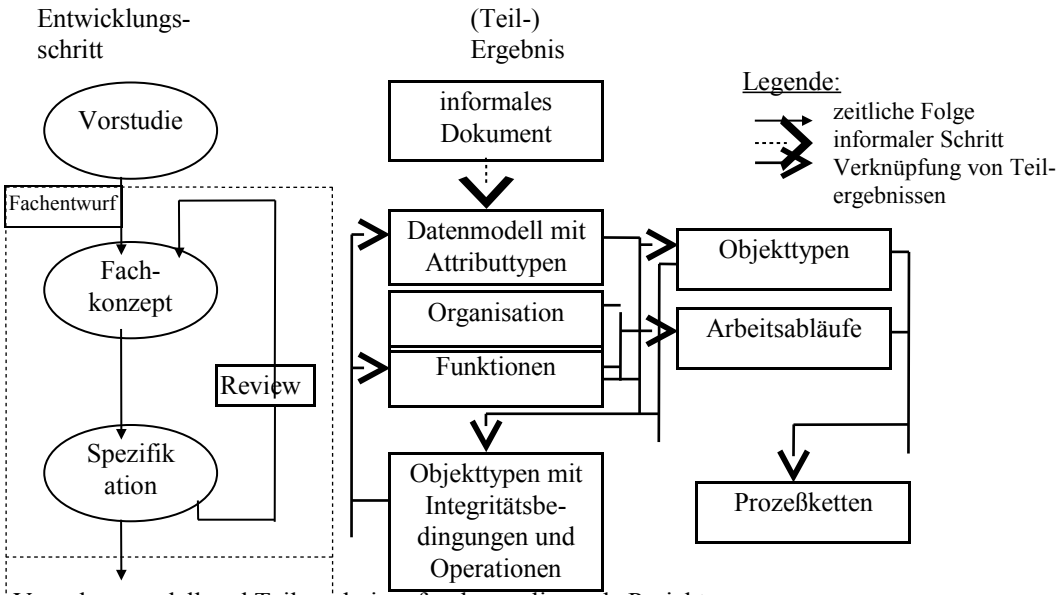


Abb. 1: Vorgehensmodell und Teilergebnisse für das vorliegende Projekt

Wesentliche Ergebnisse aus dem ersten Schritt (Fachkonzept) für die vorliegende Problemstellung sind:

- Grafische *Arbeitsabläufe* der „Geschäftsprozesse",
- ein *Datenmodell*, das als Entity-Relationship-Modell (ERM) *und* als Relationenmodell (RM) ausgeführt wird,
- ein *Datenkatalog* mit Typisierung aller Attribute.

Im zweiten Schritt (Spezifikation) wurden die nur grob skizzierten Funktionen detailliert spezifiziert, und zwar mit der formalen, mengenorientierten Spezifikationssprache **Z** [vgl. Dill90], die strukturell durchaus verträglich ist mit dem ebenfalls mengenorientierten Relationenmodell [vgl. BaHa91]. **Z** erlaubt eine von Programmiersprachen unabhängige Modellierung der Relationen und ihrer Attribute in Form sog. *Schemata*, innerhalb derer zulässige Operationen und Integritätsbedingungen als Prädikate definiert werden.

Ein Datenmodell sichert viele Integritätsbedingungen. Warum wird darüber hinaus noch formal spezifiziert? Die Integrität der Datenbasis ist der Hauptschwachpunkt des Systems HIS-POS und u.W. auch vieler anderer Informationssysteme. Dagegen ist das Problem der Integritätsbedingungen bereits hier nicht mehr trivial und keinesfalls durch ein Datenmodell gelöst.

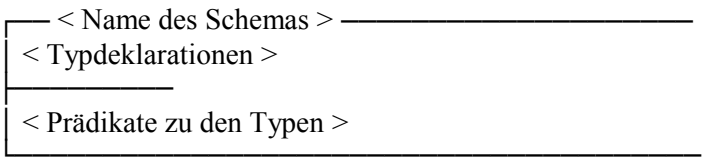
Durch Anwendung des Relationenmodells oder eines erweiterten ERM wie SERM (strukturiertes ERM) [Sinz88] werden viele Integritätsbedingungen gefunden, jedoch nicht alle. Erst eine formale Spezifikationen zwingt zum exakten Durchdenken des Sachverhaltes, deckt Inkonsistenzen auf und leitet zu Abstraktionen hin, die man mit informalen, am konkreten Problem orientierten Beschreibungen nicht findet. Formale Spezifikationen wirken qualitätserhöhend, weil Fachprobleme einer frühzeitigen Abstimmung zugeführt werden, die sonst der Programmierer implizit löst.

Unsere bisherigen Erfahrungen zeigen sogar, daß auch ein widerspruchsfreies Relationenmodell erst nach Fertigstellung einer Spezifikation vorliegen kann (Beispiele hierzu in Abschnitt 4.4). Die Spezifikation wirkt in zwei gegensätzliche Richtungen: Zum einen bereitet sie vorausschauend die programmiersprachliche Implementation des Systems vor, zum anderen korrigiert und ergänzt sie rückblickend die Ergebnisse des Fachkonzeptes. Die Spezifikation liefert:

- *Integritätsbedingungen* zwischen Attributen und Relationen. Dies sind die Invarianten der Zustände der Datenbasis, also das, was sich bei Änderungsoperationen nicht ändern darf.
- Zustandsänderungen bei *Operationen* auf Relationen.

Da die Formalismen von **Z** nicht allgemein bekannt sein dürften, fügen wir hier kurz einige Erläuterungen mit einem Beispiel an:

Ein *Schema* faßt Aussagen zu deklarierten Objekten zusammen. Diese können sich auf einzelne oder mehrere Relationen (Cluster, Teilsysteme) sowie auf Operationen beziehen. Schemata können in andere integriert werden, so daß auch komplexe Spezifikationen knapp zusammengefaßt werden können. Formal wird das Schema durch einen Rahmen vom erläuternden Text abgehoben und soll in diesen eingebunden werden. **Z** zielt also auf eine Folge von formalen und informalen Bestandteilen einer Spezifikation ab. Manchmal werden auf diese Weise scheinbar einfache natürlichsprachliche Aussagen konstruiert, hinter denen aber sehr komplexe (und auch komplex zu implementierende!) Integritätsbedingungen stehen. Ein Schema ist folgendermaßen aufgebaut:



Beispiel:

Unter der Annahme, daß Datentypen für die Attribute und die Relationen `person` und `adresse` bereits spezifiziert sind, wäre eine Relation für mehrere Adressen einer Person folgendermaßen zu schreiben:

```
┌──Person_AdresseRel──────────────────────────────────────────────────┐
│ person_adresse :   P PERSON_ADRESSE                  (P = Potenzmenge)
├─────────────
│ PRIM_KEY_OF    person_adresse {Pers-Nr, Adr-Nr}      (primary key)
│ REQUIRED       person_adresse {AdrKnz}               (REQUIRED= keine Nullwerte)
│ ALT_KEY_OF     adresse {Pers-Nr, AdrKnz}             (alternate key)
│ ∀pa : person_adresse • pa.AdrKnz = 'HeimSem'
│                      ⇒ ¬∃ pa1 : person_adresse •
│            pa1.Pers-Nr = pa.Pers-Nr ∧ pa1.Adr-Nr ≠ pa.Adr-Nr
└─────────────────────────────────────────────────────────────────────┘
```

Die Relation `person_adresse` drückt die Beziehung zwischen Personen und Adressen aus. Wenn nicht zwischen Heimat- und Semesteradresse unterschieden wird, d.h. das Attribut `AdrKnz` den Wert `'HeimSem'` annimmt, dann gibt es für die betreffende Person keine weitere Adresse mehr. Man vermeidet durch diese Festlegung z.B., daß der Programmierer die Unbestimmtheit dadurch „löst“, daß er die Semsteradresse in die Heimatadresse kopiert. Dies müßte man wissen, wenn eine SQL-Abfrage an das fertige System ein richtiges Ergebnis liefern soll.

Man beachte die unterschiedliche Bedeutung der Begriffe: Eine einzelne Beziehung zwischen Person und Adresse ist ein Objekt des *Objekttyps* PERSON_ADRESSE. Haben wir es mit einer bestimmten Menge von Objekten des Typs PERSON_ADRESSE zu tun, stellt diese die *Relation* `person_adresse` dar. Weitere Formalien zu **Z** werden in Abschnitt 4.4 erläutert.

4 Ergebnisse

Folgende Ergebnisse werden dargestellt:

- die Liste der Arbeitsabläufe („Geschäftsprozesse“),
- das Datenmodell als zentrales und wichtigstes Teilergebnis,
- der Dialogentwurf als Prozeßkette,
- Beispiele zur Spezifikation.

5 Arbeitsabläufe *(Geschäftsprozesse)*

Die Arbeitsabläufe werden hier lediglich als Aufzählung ohne nähere Kommentierung dargestellt. Sie bilden noch keinen generalisierten Soll-Ablauf, sondern geben im wesentlichen den Istzustand in unserer Fakultät wieder. Hier sei auf die Ergebnisse des bayerischen Verbundprojektes „Optimierung von Universitätsprozessen“ verwiesen [SiKru95]. Für Außenstehende muß lediglich erläutert werden, daß die Anmeldung zu Prüfungen durch Formulare erfolgt, die über einen Belegleser dem HIS-POS-System zur Verfügung gestellt werden und daß die Klausurorganisation im Hauptstudium noch nicht über HIS-POS abgewickelt wird.

Im Vorgriff auf das Datenmodell wird den Arbeitsabläufen zugeordnet, welche Entitätstypen in welchen Abläufen erzeugt oder geändert werden. Unterstrichene Abläufe sind sogenannte *lange Transaktionen*, das sind längere Arbeitsabläufe im Dialog, die unterbrechbar ge-

staltet sein müssen. ‘,’ bedeutet UND, ‘|’ ODER, ‘-’ zeigt an, daß keine Daten erzeugt oder geändert werden. Die Abläufe:

	Operation	Entitätstyp
Allgemeine Abläufe		
• Ändern von Prüfungsleistungen	ändern	LEISTUNG
• Bestätigen von Leistungen (z.B. für Bafög)	-	
• Anerkennen von Leistungen	anlegen	ANERKENNUNG, LEISTUNG
• Bearbeiten Exmatrikulationen	ändern	STUDIENVERLAUF
• Pflegen Stammdaten	anlegen | ändern	PERSON | FACH | VERANSTALTUNG | und „Prüfungsordnung“
• [Erstellen Statistiken]	-	
Modular verwendbare Abläufe		
• Bearbeiten Nichterscheinen	ändern	LEISTUNG
• Vorbereiten Klausuren intern	anlegen	PRÜFUNG, DOZENT, PRÜFRAUM
• Bearbeiten Anmeldelisten	anl. | änd.	LEISTUNG
Vorgänge Grundstudium		
• Vorbereiten Klausuren extern	-	
• Abwickeln Klausuranmeldungen	(s. Bearbeiten Anmeldelisten)	
• Auswerten Klausuren	ergänzen	LEISTUNG
• Vorbereiten mündliche Ergänzungsprüfung	(s. Vorbereiten Klausuren intern)	
• Auswerten mündliche Ergänzungsprüfung	(s. Auswerten Klausuren)	
• Erstellen Vordiplom (Batchprozeß)	ändern	STUDIENVERLAUF
• Vorbereiten punktuelle Prüfung	(s. Vorbereiten Klausuren intern)	
Vorgänge Hauptstudium		
• Vorbereiten Diplomklausuren	(s. Vorbereiten Klausuren intern)	
• Anmeldungen Diplomklausuren	(s. Bearbeiten Anmeldelisten)	
• Organisieren Prüfer	-	
• Abwickeln Diplomklausuren	(s. Auswerten Klausuren)	
• Anmeldungen Diplomarbeit	anlegen	HAUSARBEIT
• Annehmen/Zurückweis. Diplomarbeitsthema	ändern	HAUSARBEIT
• Überwachen Fristen Diplomarbeit	-	
• Abwickeln Bewertung Diplomarbeit	ändern	HAUSARBEIT
• Erstellen Diplomzeugnis (Batchprozeß)	(s. Erstellen Vordiplom)	

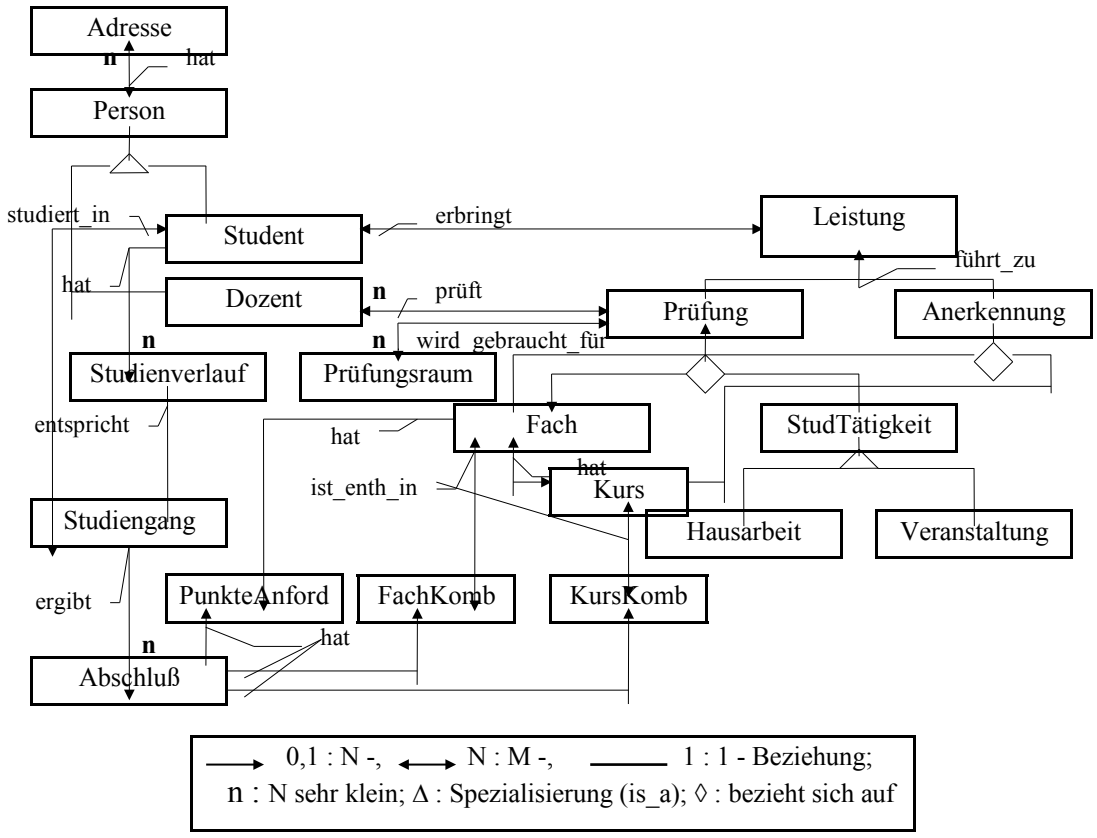
Man sieht, daß die stark an den Istabläufen unserer Fakultät orientierten „Geschäftsprozesse“ Vereinheitlichungen und Abstraktionen vor allem zwischen Grund- und Hauptstudium zulassen. Dies wurde durch die Aufnahme von Verweisen auf Abläufe statt Wiederholungen von Entitätstypen angedeutet. So ist die Klausurorganisation in Grund- und Hauptstudium strukturell und - wie sich im folgenden Punkt zeigen wird - sogar auf der Ebene der Attribute identisch. Darüberhinaus bearbeiten alle Prüfungsarten (Klausur, mündliche Prüfung und deren Varianten wie die „Ergänzungsprüfung“) dieselben Entitätstypen, weil `PrüfArt` lediglich ein Attribut von PRÜFUNG ist.

Es dürfte hiermit gezeigt worden sein, daß das intendierte Softwaresystem einfach gehalten werden kann sowohl von den Funktionen an der Dialogoberfläche als auch von den jetzt zu

spezifizierenden Entitätstypen, die wir im folgenden *Objekttypen* nennen. Warum wir das tun, wird bei der Entwicklung des Datenmodells deutlich.

6 Datenmodell

Zunächst wird das Datenmodell zur Übersicht als Entity-Relationship-Diagramm in einer Notation ähnlich den Bachmann-Diagrammen gezeigt. Es bildet die Objekttypen für das Prüfungsgeschehen und die Prüfungsordnung selbst ab. Alle nicht zum Diskursbereich einer Prüfungsverwaltung gehörenden Beziehungen werden weggelassen, z.B. die Beziehung Dozent ↔ Veranstaltung, da für das Prüfungsgeschehen nur die Beziehung Dozent ↔ Prüfung erforderlich ist. Das Bild stellt keinesfalls die erste von den Studenten erstellte Fassung dar, sondern ein mehrfach aufgrund des Relationenmodells und der Spezifikation *rekonstruiertes* stark abstrahierendes Modell. Es enthält eine Reihe von Generalisierungen, auf die man vermutlich intuitiv nicht kommen wird, z.B. den Objekttypen STUDIENTÄTIGKEIT. Zentraler Objekttyp ist der Typ LEISTUNG mit den relativ komplex verknüpften Objekttypen KURS und FACH. Hierdurch wird es möglich, eine Prüfung sowohl als zeitpunktbezogenes Ereignis (mündlich, Klausur) als auch als Endpunkt einer länger dauernden Tätigkeit zu modellieren. Die Prüfungsordnung ist in den Objekttypen STUDIENGANG, ABSCHLUSS, FACH- und KURSKOMBination abgebildet und erlaubt sowohl ein konventionelles Prüfungssystem als auch ein Kreditpunktesystem mit studienbegleitender Prüfung jeder Lehrveranstaltung.



Legende:

Abb. 2: Objekttyp-Beziehungs-Struktur des Datenmodells

Von links nach rechts werden ähnlich wie im SERM zunehmende Existenzabhängigkeiten dargestellt. Beziehungsbezeichnungen sind in Richtung des weiter entfernt liegenden Pfeiles (↔) oder in Pfeilrichtung (→) zu lesen. Die Kardinalität einiger 1 : N - Beziehungen ist sehr

klein und zudem nur selten > 1. Dies wurde durch ein '**n**' neben der Pfeilspitze kenntlich gemacht. Diese **n** stellen semantisch genau die Sonderfälle dar, die bei einer Software Probleme machen, wenn sie nicht vorgesehen sind:

- Ein Student kann in mehreren Studiengängen studieren (Nebenfächer, Zweitfach),
- es prüfen mehrere Prüfer (keine Beisitzer!),
- ein Student hat mehrere Studienverläufe (Wechsler),
- eine Klausur muß in mehreren Hörsälen geschrieben werden (für die die Software eine Sitzverteilung generiert),
- es sind verschiedene Abschlüsse in einem Studiengang möglich („Y-Modell" der Gesamthochschulen).

Die folgende Aufzählung der Objekttypen zeigt die Attributnamen in relationaler Schreibweise. Die Objekttypen werden unterteilt in *Grundobjekttypen* (auch *Kernentitäten*, [vgl. Vett90]), das sind unabhängige Entitätstypen, ihre Spezialisierungen und davon abhängige Objekttypen, und *Vorgangsobjekttypen*, das sind von Grundobjekttypen existenzabhängige Entitäts- oder Beziehungstypen. Verdichtete Objekttypen werden nicht modelliert, da zunächst nur die primär durch den Benutzer erzeugte Datenbasis spezifiziert werden soll. Nur in genau einem Objekttyp (STUDIENVERLAUF) sind abgeleitete Attribute enthalten, ansonsten sollen solche Attribute zur Laufzeit ermittelt werden. Jede Relation enthält einen - ggf. künstlichen - Primärschlüssel (*Surrogatschlüssel*). Die natürlichen Schlüsselattribute sind durch Schrägdruck als 'alternate key' gekennzeichnet.

(1) Grundobjekttypen:

PERSON (Pers-Nr, PersonArt, Name, VorName, Titel, GebDat, GebName, GebOrtName, GeschlechtKnz) [3]
{**Generalisierung der Subtypen STUDENT und DOZENT**}

STUDENT (Matr-Nr, BafögNr, ErstimmatrDat)
{**statische Daten eines Studenten**}

DOZENT (Mitarb-Nr, PrüfBerechtKnz, BüroRaumBez, BüroTelNr)
{**für das Prüfungsgeschehen relevante Daten eines Dozenten**}

ADRESSE (Adr-Nr, *StaatsKnz*, *Plz*, *OrtsName*, *DetailsAnschrift*)
{**häufig zwei, gelegentlich auch drei Adressen für eine Person; TelNr und EMailAdr sind Attribute eines Beziehungstyps PERSON_ADRESSE**}

STUDIENGANG (StudGang-Nr, *StudGangBez*, StudGangArt, HochschulName, FakultätName)
{**hier relevante Daten der Fakultät/des Fachbereichs**}

ABSCHLUSS (Abschl-Nr, *StudGang-Nr, StudAbschnittKnz, PrüfVersKnz,* MinPunkteAnz, MaxPunkteAnz)
{**Bedingungen für den Abschluß**}

FACH (Fach-Nr, *FachBez*)

KURS (Kurs-Nr, *KursBez*, *KursArt*, VersucheAnz)
{**KURS ist eine zeitunabhängige VERANSTALTUNG**}

FACHKOMB (FachKomb-Nr, Abschl-Nr, PunkteAnz)

KURSKOMB (KursKomb-Nr, Abschl-Nr, Fach-Nr)
{**Zulässige Kombinationen für Fächer und Kurse**}

PUNKTE_ANFORD (Fach-Nr, Abschl-Nr, MinPunkteAnz, MaxPunkteAnz)
{**Bedingungen für ein Fach**}

PRÜFRAUM (Raum-Nr, *RaumBez*, PrüfPlatzHochAnz, PrüfPlatzNiedrigAnz, SitzplatzAnz)
{**für die Klausur-Organisation**}

[3] Für häufig vorkommende benutzerdefinierte Datentypen wird der Typ durch einen Suffix im Attributnamen kenntlich gemacht. Abgekürzte Typen: `-Nr` = Schlüssel, `Nr` = Nummer, `Anz` = Anzahl, `Bez` = Bezeichnung, `Dat` = Datum, `Zeit` = Zeitpunkt, `Knz` = Kennzeichen, `Zust` = Zustand. Andere Typen wie `Name` und `Art` werden nicht abgekürzt.

(2) Vorgangsobjekttypen:

STUDIENVERLAUF	(Matr-Nr, StudGang-Nr, StudGangSemAnz, HauptstudSemAnz, UrlaubsSemAnz, Status, VorExamDat, VorExamNote, AbschlußNote, PrüfVersGrundstudKnz, PrüfVersHauptstudKnz, ÄndDat) {**abgeleitete Daten eines Studenten**}
LEISTUNG	(Leist-Nr, LeistArt, Matr-Nr, Note, VersuchsNr, VorkommnisKnz)
PRÜFUNG	(Prüf-Nr, PrüfArt, Datum, HilfsmittelBez, BeginnZeit, EndeZeit)
ANERKENNUNG	(Anerk-Nr, AnerkArt, Anmerkung)
STUDTÄTIGKEIT	(StudTätigk-Nr, TätigkArt, Kurs-Nr, TätigkName, KreditPktAnz)
VERANSTALTUNG	(Veranst-Nr, SemKnz, Jahr) {**zeitabhängig stattfindender KURS}**
HAUSARBEIT	(Arbeit-Nr, ArbeitTyp, AnmeldeDat, PlanWochenAnz) {**insbesondere Studien-, Staatsexamens- und Diplomarbeiten**}

In der nächsten Verfeinerungsstufe sind Datentypen als statische Integritätsbedingungen zu spezifizieren. Dieser Entwicklungsschritt gehört noch zum Datenmodell.

Objekttypen werden gebildet, indem die Auflistung der Attribute aus dem Datenkatalog um die Einträge *Integritätsbedingungen*, *Operationen* und *Transaktion* erweitert wird. Dahinter stehen folgende Überlegungen: Durch *Typen* werden Integritätsbedingungen auf der Ebene jedes Attributs formuliert. Darüberhinaus gibt es Bedingungen, die sich auf ein Tupel und die sich auf die gesamte Relation beziehen [vgl. Voss94, 513ff.]. Die letztgenannten Integritätsbedingungen sind Gegenstand der Spezifikation.

Unter *Operation* werden lediglich Namen von Operationen eines Objekttyps genannt, die spezifiziert werden müssen. Die standardmäßigen Operationen werden nur genau einmal für das gesamte System spezifiziert, das sind *anlegen*, *ändern* und *löschen* als datenverändernde Operationen.

Transaktion benennt diejenigen Objekttypen, mit denen zusammen der betrachtete Objekttyp auf Tupelebene eine logische Einheit bildet. So wird spezifiziert, daß eine ADRESSE nur aufgrund einer PERSON, sei sie STUDENT oder DOZENT, angelegt werden kann. Dies schließt nicht aus, daß eine neue PERSON auf eine existierende ADRESSE Bezug nimmt.

Mit dieser groben Beschreibung der Datentypen (Attribute) und Objekttypen ist das Datenmodell abgeschlossen. Es kann lediglich noch um Mengengerüste erweitert werden, die eine frühzeitige Abschätzung von Implementierungsentscheidungen erlauben.

7 Dialogentwurf als Prozeßkette

Während jeder Objekttyp über Standardoperationen verfügt (anlegen, lesen, ändern, löschen), werden in der Spezifikation diejenigen Operationen gesucht, die die zu implementierenden Prozesse auslösen oder bewirken. Dies geschieht über einen ersten groben *Dialogentwurf*, der als Basis für einen Prototyp dienen kann. Die Funktionen der Dialogoberfläche wirken auf die Objekttypen, genauer: sie lösen Standardoperationen aus. Dabei wirken Funktionen auf Grundobjekttypen und auf Vorgangsobjekttypen verschieden.

Funktionen auf Grundobjekttypen bewirken, was man in der betrieblichen Praxis *Stammdatenpflege* nennt. Die damit verbundenen Prozesse sind ausschließlich Dialogtransaktionen, deren Auslöser fallweise eintretende Ereignisse sind. Angelegt oder geändert wird *ein* logisches Tupel über die unter Transaktion genannten Objekttypen. Massenänderungen von Stammdaten kommen im vorliegenden System nicht vor.

Funktionen auf Vorgangsobjekttypen bilden zum Teil *lange Transaktionen*, für die Zwischenzustände festgehalten werden müssen. Sie werden teilweise im Dialog nur ausgelöst und laufen als Batchprozeß ab. In allen Fällen entspricht jedoch eine elementare Funktion genau

einem elementaren oder aggregierten Vorgangsobjekttyp, niemals mehreren. Außerdem bezieht sie sich im Normalfall auf *alle oder viele Tupel*, nur in Sonderfällen auf einzelne (z.B. korrigieren einer Note).

Während die ändernden Operationen auf Grundobjekttypen weitgehend durch die Integritätsbedingungen spezifiziert sind, müssen die Operationen auf Vorgangsobjekttypen umfassender spezifiziert werden. Es folgt die Auflistung der Prozesse als Dialogbaum mit Zuordnung der dabei veränderten Objekttypen:

Prozesse (Sollkonzept)	angelegter/geänderter Objekttyp
Grundobjekttypen:	
• **pflegen Stammdaten**	
- Personaldaten	STUDENT (PERSON und ADRESSE sind für den Benutzer nicht als Objekttypen erkennbar)
	DOZENT
- Prüfungsordnung	STUDIENGANG
	FACH
	FACHKOMB, KURSKOMB
- Lehrangebot	KURS
	VERANSTALTUNG
Vorgangsobjekttypen:	
• **eröffnen Semester**	
- anlegen Veranstaltungen	VERANSTALTUNG
- pflegen Prüfer	DOZENT (Funktion in *pflegen Stammdaten*)
- fortschreiben Studienverlauf (Anstoß Batchprozeß)	STUDIENVERLAUF
• **vorbereiten Prüfungen**	
- anlegen Prüfung	PRÜFUNG
- pflegen Prüfer	DOZENT (Funktion in *pflegen Stammdaten*)
- pflegen Räume	PRÜFRAUM
- anlegen schriftliche Arbeit	HAUSARBEIT
• **bearbeiten Prüfungsanmeldung**	
- anlegen Teilnehmer	LEISTUNG
- einsehen Stammdaten	-
- pflegen Anerkennung	ANERKENNUNG
- generieren Raumbelegung	-
• **bearbeiten Prüfungsergebnisse**	
- eintragen Noten	LEISTUNG
- generieren Noten / Zeugnisse	STUDIENVERLAUF
- überwachen Termine	-
• **abschließen Semester** (archivieren / verdichten zurückliegende Vorgangsdaten)	

8 Spezifikation

Mit den Spezifikationsbeispielen beziehen wir uns auf das Anmeldeverfahren für schriftliche Prüfungen. Sie sind in der Prozeßkette `bearbeiten Prüfungsanmeldung` angesiedelt. Dabei haben sich Studenten für Prüfungen gemeldet, die innerhalb einer bestimmten Zeitperiode stattfinden. Ob diese Anmeldung online durch die Studenten, über einen Belegleser oder mittels Tastatureingabe des Sachbearbeiters erfolgt, ist hier nicht von Interesse.

9 Objekttyp ANMELDUNG als einfaches Beispiel

Die aufbereiteten Anmeldungen sind vom Objekttyp ANMELDUNG, den es im bisherigen Datenmodell (s. Abschnitt 4.2) nicht gibt:

ANMELDUNG (Matr-Nr, Prüf-Nr, Fach-Nr)

Die zugehörige Relation `angemeldet?`[4], in der ausschließlich Anmeldungen für den vorgesehenen Prüfungszeitraum erfaßt sind, stellt sich folgendermaßen dar:

```
┌──AnmeldungRel──────────────────────────────────────┐
│ angemeldet? :      P ANMELDUNG                     │
├────────────                                        │
│ PRIM_KEY_OF    angemeldet {Matr-Nr, Prüf-Nr}       │
└────────────────────────────────────────────────────┘
```

Die Konsistenz der vorliegenden Anmeldedaten ist zu überprüfen. Das Ergebnis sind die zugelassenen und die unzulässigen Anmeldungen. Die Relation `zulässig!` sei ebenfalls eine Menge von Datensätzen des Typs ANMELDUNG,

```
┌──ZulAnmeldungRel──────────────────────────────────┐
│ zulässig!    :      P ANMELDUNG                    │
├──────────                                         │
│ PRIM_KEY_OF    zulässig{Matr-Nr, Prüf-Nr}         │
└───────────────────────────────────────────────────┘
```

während die Relation `unzulässig!` von einem anderen Objekttyp UNZUL-ANMELD mit einem zusätzlichen Attribut zur Kommentierung des Ablehnunggrundes ausgeht.

UNZUL-ANMELD (Matr-Nr, Prüf-Nr, Fehlermeldung, Fach-Nr)

```
┌──UnzulAnmeldungRel─────────────────────────────────┐
│ unzulässig! :      P ANMELDUNG                     │
├──────────                                          │
│ PRIM_KEY_OF    unzulässig{Matr-Nr, Prüf-Nr, Fehlermeldung}
└────────────────────────────────────────────────────┘
```

10 Zulässigkeitsprüfungen als komplexes Beispiel

Nachfolgend werden zwei Zulässigkeitsprüfungen spezifiziert, und zwar
- ob die angemeldeten Prüfungen tatsächlich stattfinden,
- ob sie zu dem vorgesehenen Abschluß passen.

Neben diesen beiden Konsistenzprüfungen müssen noch weitere Zulässigkeiten geklärt werden, z.B. ob der angemeldete Student überhaupt immatrikuliert ist, ob er seine angestrebte Prüfung schon endgültig bestanden oder nicht bestanden hat, oder ob die zu prüfenden Kurse und Fächer in den vorgesehenen Prüfungsrahmen der Prüfungsordnung passen. Alle diese Prüfungen laufen in ähnlicher Weise ab. Man erhält jedesmal eine Ausgabetabelle vom Typ **P** ANMELDUNG (versehen mit dem Suffix `_ok`) und eine Ausgabetabelle vom Typ **P** UNZUL-ANMELD (Suffix `_err`):

```
zulässig!   = prüf-nr_ok! ∩ abschluß_ok! ∩ ...

unzulässig! = prüf-nr_err! ∪ abschluß_err! ∪ ...
```

Die Relation `unzulässig!` kann einen Studenten mehrfach enthalten, wenn auf ihn mehrere Fehlermeldungen zutreffen. Daher ist `Fehlermeldung` Schlüsselbestandteil.

[4] Ein Fragezeichen hinter einem Bezeichner kennzeichnet Eingabedaten, ein Ausrufezeichen Ausgabedaten.

Um die Einbindung der zulässigen Anmeldedaten in das Datenmodell zu verfolgen, betrachte man die im Anhang beigefügte Abbildung des Datenmodells in relationaler Form. Die Datensätze in der Tabelle `zulässig!` werden in die Tabellen

leistung : **P** LEISTUNG
leistung-prüfung : **P** LEISTUNG_PRÜFUNG
leistung_fach : **P** LEISTUNG_FACH

wie nachfolgend beschrieben integriert. ANMELDUNG ist also ein temporärer Objekttyp und deshalb nicht im Datenmodell enthalten.

Für jeden Datensatz aus `zulässig!` wird ein neuer Datensatz in `leistung` und in `leistung_prüfung` eingefügt. Ein neuer Datensatz in `leistung_fach` wird nur dann angelegt, wenn ein zu prüfender Kurs mehreren Fächern eines Studiengangs zugerechnet werden kann. Aus diesem Grunde enthält der Objekttyp ANMELDUNG das Attribut `Fach-Nr.`

Aus der Tabelle `zulässig!` können die zugehörigen Werte für die `Matr-Nr` in `leistung` und die `Prüf-Nr` in `leistung_prüfung` übertragen werden. Die `LeistArt` erhält den Wert `` `geprüft´ `` (statt `` `anerkannt´ ``), die `Note` und die `VersuchsNr` bekommen vorläufig einen Nullwert, das `VorkommnisKnz` die Charakterisierung `` `angemeldet´ ``. Liegen die Korrekturergebnisse zu einem späteren Zeitpunkt vor, so können die Attribute `Note`, `VersuchsNr` und `VorkommnisKnz` aktualisiert werden. Die `Fehlermeldung` in der Tabelle `unzulässig!` kann etwaigen Einsprüchen entgegengehalten werden.

(1) Kontrolle der Prüfungsnummer

```
┌── Check-PrüfNr ──────────────────────────────────────────────────────────
| Ξ DB                                                                  [5]
| angemeldet? :      P ANMELDUNG
| prüfNr_ok!  :      P ANMELDUNG
| prüfNr_err! :      P UNZUL-ANMELD
├──────────
| prüfNr_ok! =       {anm: angemeldet?; p: prüfung | anm.Prüf-Nr = p.Prüf-Nr ∧
|                    (∀lp: leistungen-prüfung • lp.Prüf-Nr ≠ anm.Prüf-Nr) • anm }
| {pe: prüfNr_err! • (Matr-Nr, Prüf-Nr, Fach-Nr) pe} = angemeldet? \ prüfNr_ok!
|
| ∀ pe: prüfNr_err! • pe.Fehlermeldung = 'die angegebene Prüfung wird nicht geschrieben'
└──────────────────────────────────────────────────────────────────────────
```

Bevor die Anmeldungen überprüft werden können, müssen die in naher Zukunft stattfindenden Prüfungen bereits Bestandteil der Datenbasis sein. Allerdings gibt es zu diesen Prüfungen noch keine Leistungssätze und damit auch keine Sätze in der Tabelle `leistung_prüfung`. Die Anmeldesätze mit korrekten Prüfungsnummern werden in der Relation `prüfNr_ok!` gesammelt. Prüfungsnummern sind dann korrekt, wenn sie in der Tabelle `prüfung` vorliegen und für diese Prüfungen eben noch keine Leistungseinträge existieren.

Die Kombination (`Matr-Nr, Prüf-Nr, Fach-Nr`) reduziert das Objekt `pe` auf drei anstelle von vier Attributen. Die auf drei Attribute projizierte Relation `prüfNr_err!` entspricht so der Differenzmenge zwischen allen Anmeldungen und den zulässigen Anmeldungen bezüglich des Abgleichs zwischen den Prüfungsnummern.

(2) Kontrolle des Abschlusses

[5] Das Kürzel *DB* verweist auf die im Anhang vorgestellte Datenbasis. Der Buchstabe Ξ besagt, daß sich der Zustand der Datenbasis nicht verändert (lesender Zugriff). Eine Änderung wird mit Δ gekennzeichnet.

Zur Überprüfung, ob der Student sich auch tatsächlich zu Prüfungen angemeldet hat, die in seinen Studienabschluß passen, Müssen wir Relationen zwischen Relationen betrachten. Das Datenmodell im Anhang zeigt, daß sich über die Verbindungslinien Beziehungen aufbauen lassen. Zum Beispiel gilt:

$$\forall\ s: \text{student} \bullet \exists\ ss: \text{student\_studgang};\ stg: \text{studiengang};\ a: \text{abschluß} \bullet s.\text{Matr-Nr} = ss.\text{Matr-Nr} \wedge ss.\text{StudGang-Nr} = stg.\text{StudGangNr} \wedge stg.\text{StudgangNr} = a.\text{StudgangNr}$$

Über direkte Beziehungen, die über Attribute mit gleichem Wertebereich hergestellt werden, gelangt man zu indirekten Beziehungen und kann sich somit weitere sinnvolle Zusammenhänge erschließen. Im vorliegenden Fall wird die Beziehung zwischen Student und Abschluß und der Zusammenhang zwischen Prüfung und passendem Abschluß offengelegt. Die folgenden verbalen Ausführungen zum nächsten Schema zeigen die Grenzen von Prosa als Darstellungsmittel der Spezifikation komplexer Sachverhalte.

Ein Student strebt mit seinem Studium einen Abschluß an, der entweder ein Teilabschluß wie das Vordiplom oder aber die erfolgreiche Beendigung des Gesamtstudiums sein kann. Weiterhin kann er das Studienziel in einem oder in mehreren Studiengängen verfolgen. Werden mehrere Studiengänge studiert, so werden gleichzeitig mehrere Abschlüsse angestrebt.

Um diese Zusammenhänge zu formalisieren, führen wir als Typ eine *binäre Relation* ein, die wir mit 'X ↔ Y' kennzeichnen. Im Schema `Check-Abschluß` (vgl. S. 16) sind die Beziehungen zwischen Student und Abschluß bzw. zwischen Prüfung und Abschluß entsprechend deklariert (siehe die *Deklarationen* `Dk3` und `Dk4`)[6]. Die genaue Definition der beiden binären Relationen erfolgt durch die *Prädikate* `Pr1` und `Pr2` im unteren Teil des Schemas.

Aus den Kardinalitäten der direkten Beziehungen lassen sich auch die Kardinalitäten der indirekten Beziehungen bestimmen. In unserem Fall sind die binären Relationen zwischen Student und Abschluß und zwischen Prüfung und Abschluß keine Funktionen, da sowohl ein Student als auch eine Prüfung mehreren Abschlüssen zugerechnet werden kann. Das Bild eines Studenten `s` oder einer Prüfung `p` kann also mehr als ein Element enthalten.

Formal beschreibt man das Bild eines Elementes aus dem Definitionsbereich einer Relation mit `rel_Bin !{x}!`, wobei `rel_Bin` für den Namen der binären Relation und das `x` für ein bestimmtes Element aus dem Definitionsbereich der Relation steht. Im Fallbeispiel gilt:

```
# student-abschl_Bin !{s}!≥ 1 ∧  # prüfung-abschl_Bin !{p}! ≥ 1
```

Jeder Student hat mindestens einen Abschluß zum Ziel und jede Prüfung ist die Station zu mindestens einem Abschluß. Die Anmeldung zu einer Prüfung ist bezüglich des Abschlusses dann konsistent, wenn das Bild der angegebenen Prüfung und das Bild des vorliegenden Studenten mindestens einen gemeinsamen Abschluß enthalten.
Bei genauerer Untersuchung des Modells läßt sich feststellen, daß bei einer Aufteilung des Studiums in Grund- und Hauptstudium auch einem Studenten, der nur einen Studiengang studiert, zwei Abschlüsse zugeordnet werden, nämlich der Abschluß des Grundstudiums und des Hauptstudiums. Jeder dieser beiden Abschlüsse ist zudem an eine bestimmte Prüfordnungsversion gekoppelt, die aus dem `STUDIENVERLAUF` über die Attribute `PrüfVersGrundstudKnz` und `PrüfVersHauptstudKnz` abgelesen werden kann. Der Prüfungskandidat kann sich nun in drei verschiedenen Zuständen befinden:

1. Er ist nur für Prüfungen des Grundstudiums zugelassen. In dieser Situation gilt für seinen Datensatz `stv` bezüglich des `STUDIENVERLAUF`s : `stv.VorExamDat = null ∧`

[6] Mit Dk_i und Pr_j wird im folgenden Text auf Teile des Schemas `Check-Abschluß` verwiesen.

`stv.HauptStudSemAnz = 0`. Die `HauptStudSemAnz` kann erst dann größer als der Wert 0 sein, wenn die Zulassung zum Hauptstudium besteht.

2. Er ist nur für Prüfungen des Hauptstudiums zugelassen. In dieser Situation gilt für den Datensatz `stg` bezüglich des `STUDIENVERLAUF`s: `stv.VorExamDat ≠ null`. Das Vorexamen ist also vorhanden, daher gilt aufgrund einer Integritätsbedingung auch `stv.HauptStudSemAnz > 0`.
3. Er hat das Grundstudium noch nicht abgeschlossen, ist aber dennoch berechtigt, auch Prüfungen des Hauptstudiums zu absolvieren. In diesem Fall gilt die Bedingung:
   `stv.VorExamDat = null ∧ stv.HauptStudAnz > 0`.

Zusammengenommen gilt dann die stets wahre Aussage:

```
┌── Zulassung_Grund-/Hauptstudium ──────────────────────────
│ stv  :      studienverlauf
├──────────
│    ( stv.VorExamDat = null ∧ stv.HauptStudSemAnz = 0 )
│ ∨  ( stv.VorExamDat ≠ null )
│ ∨  ( stv.VorExamDat = null ∧ stv.HauptStudAnz > 0 )
└──────────────────────────────────────────────────────────
```

Man beachte, daß sich die ∨ - verknüpften Aussagen gegenseitig ausschließen. Die binäre Beziehung `student-abschl_Bin` in *Pr2* enthält *alle* Kombinationen `s→a` zwischen einem Studenten und einem Abschluß, die den aufgeführten Bedingungen entsprechen. Um für die jeweilige Zulassungsbeschränkung bezüglich des Grund-/Hauptstudiums den richtigen Abschluß zu finden, wird für jeden Spezialfall der richtige Abschluß ausgewählt:

```
┌── Abschluß_Grund-/Hauptstudium ────────────────────────────
│ stv :      studienverlauf
│ a   :      abschluß
├──────────
│    ( stv.VorExamDat = null ∧ stv.HauptStudSemAnz = 0
│                          ∧ a.StudAbschnKnz = `grundst´ )
│ ∨  ( stv.VorExamDat ≠ null ∧ a.StudAbschnKnz = `hauptst´)
│ ∨  ( stv.VorExamDat = null ∧ stv.HauptStudAnz > 0
│              ∧ a.StudAbschnKnz ∈ {`grundst´ , `hauptst´} )
└──────────────────────────────────────────────────────────
```

Das Bild einer Student-Studiengang-Prüfungsversion-Kombination ist also entweder ein Grundstudiumabschluß oder ein Hauptstudiumabschluß oder eine Menge aus einem Grund- und einem Hauptstudiumabschluß.

Jeder Datensatz aus der Relation `angemeldet?` enthält eine `Prüf-Nr` und eine `Matr-Nr`. Für jeden Datensatz bestimmen wir nun das passende `p` aus der Relation `prüfung` und das passende `s` aus der Relation `student`. Wenn nun das Bild von `s` bezüglich der binären Relation `student-abschl_Bin` und das Bild von `p` bezüglich der binären Relation `prüfung-abschl_Bin` übereinstimmen, dann ist die Prüfung bezogen auf den Abschluß korrekt angegeben (vgl. *Pr3* in `Check-Abschluß`). Es folgt die formale Spezifikation des gerade verbal und mit formalen Ausschnitten beschriebenen Sachverhaltes.

Check- Abschluß

Ξ DB

angemeldet?	:	**P** ANMELDUNG	*(Dk1)*
abschluß_ok!	:	**P** ANMELDUNG	*(Dk2)*
abschluß_err!	:	**P** UNZUL-ANMELD	*(Dk3)*
prüfung-abschl_Bin	:	prüfung ↔ abschluß	*(Dk4)*
student-abschl_Bin	:	student ↔ abschluß	*(Dk5)*

prüfung-abschl_Bin = {p:prüfung; pt: prüfung_tätigkeit; stgkt: studtätigkeit;
k: kurs, kbk: kurskomb_kurs; pf: prüfung_fach;
f: fach; fbf: fachkomb_fach; fb: fachkomb; a: abschluß |
(p.Prüf-Nr = pt.Prüf-Nr ∧ pt.StudTätigk-Nr =
stgkt.StudTätigk-Nr ∧ stgkt.Kurs-Nr = k.Kurs-Nr ∧
k.Kurs-Nr = kbk.Kurs-Nr ∧ kbk.KursKomb-Nr =
kb.KursKombNr ∧ kb.Abschl-Nr = a.Abschl-Nr) ∨
(p.Prüf-Nr = pf.Prüf-Nr ∧ pf.FachNr = f.FachNr ∧
f.Fach-Nr = fbf.Fach-Nr ∧ fbf.FachKomb-Nr =
fb.FachKomb-Nr ∧ fb.Abschl-Nr = a.Abschl-Nr) •
p→a } *(Pr1)*

student-abschl_Bin = {s: student; ss: student_studgang; stg: studiengang;
stv: studienverlauf; a: abschluß |
s.Matr-Nr = ss.Matr-Nr ∧ s.MatrNr = stv.MatrNr ∧
ss.StudGang-Nr = stg.StudGang-Nr
∧ stg.StudGang-Nr = stv.Studgang-Nr
∧ stg.StudGang-Nr = a.StudGang-Nr ∧
∧ a.PrüfVersKnz ∈ { stv.PrüfVersGrundStudKnz,
stv.PrüfVersHauptStudKnz }
∧ ((stv.VorExamDat = null ∧
stv.HauptStudSemAnz = 0 ∧
a.StudAbschnittKnz = `grundst´)
∨ (stv.VorExamDat ≠ null ∧
a.StudAbschnKnz = `hauptst´)
∨ (stv.VorExamDat = null ∧
stv.HauptstudSemAnz > 0 ∧
a.StudAbschnKnz ∈ { `grundst´ , `hauptst´ }))
• s→a } *(Pr2)*

abschluß_ok! = {anm: angemeldet?; stv: studienverlauf; p: prüfung; s: student |
anm.Matr-Nr = s.Matr-Nr = stv.Matr-Nr ∧
anm.Prüf-Nr = p.Prüf-Nr
∧ prüfung-abschl_Bin •{p}• ∩ student-abschl_Bin •{s}• ≠ ∅
• anm } *(Pr3)*

{ab: abschluß_err! • (Matr-Nr, Prüf-Nr, Fach-Nr) ab} =
angemeldet? \ abschluß_ok! *(Pr4)*

∀ ab: abschluß_err! • ab.Fehlermeldung = ‘die angegebene Prüfung paßt
nicht zum angestrebten Abschluß’ *(Pr5)*

11 Bewertung der Spezifikationsmethode

In den obigen Beispielen wurde die formale Spezifikation dazu verwendet, die Zustände vor, während und nach den Operationen zu beschreiben. Zustände, die sich nicht verändern, bilden die Invarianten des Datenmodells. Einfache Sachverhalte wie die Zuweisung von Primärschlüsseln oder auch die Kardinalitäten zwischen verschiedenen Datenobjekttypen können auch als entsprechend gekennzeichnete Zusätze in graphische Darstellungsformen eingehen. Komplexere Sachverhalte wie die hier aufgezeigten Zusammenhänge zwischen einer Prüfung, die ein Student ablegt, und dem Abschluß, den er anstrebt, lassen sich dagegen nicht mehr so leicht ins Bildhafte übertragen.

Die formale Strenge der Aussagenlogik hat gegenüber allgemeiner Prosa den Vorteil der Eindeutigkeit. Der Übergang in die ebenfalls eindeutige Sprache der Implementation kann dadurch nach einer gewissen Methodik ablaufen, die unter Umständen eine mehr oder weniger umfangreiche Verifikation mit einschließt. Die Ähnlichkeit einiger Restriktionen aus der **Z**-Syntax zu SQL-nahen Ausdrucksformen sind unschwer auszumachen. Um das Formale aber kommunikationsfähiger und transparenter zu gestalten, sollte man mit erläuterndem Text nicht allzu sparsam umgehen.

Der Rückkopplungseffekt der Spezifikation von Operationen auf das Datenmodell selbst sollte nicht unterschätzt werden. Erst die Untersuchung der Eigenschaften, die sich verändern, liefert ein Verständnis für das, was systemimmanent ist. Vor allem sind es die Operationen, die letztlich bestimmen, was unter dem einen und dem anderen Objekttyp und dessen gegenseitigen Beziehungen genau zu verstehen ist. Viele Begriffe erfahren durch die Spezifikation eine mehr oder weniger leichte Wandlung ihrer ursprünglichen Bedeutung. So ist z.B. in unserem Modell eine Leistung auch dann erbracht, wenn der Student sich lediglich angemeldet, die Prüfung aber noch gar nicht absolviert hat. Die Spezifikation hat also eine nicht zu unterschätzende Bedeutung für die Definition von Begriffen (vgl. hierzu auch die Arbeiten von Ortner [Ort94]).

Wenn sich aus der Untersuchung der Operationen die Invarianten des Modells herauskristallisieren, ist es wichtig, diese Eigenschaften den übrigen Entwicklern mitzuteilen, die andere Operationen untersuchen, um das gesamte Projekt konsistent zu halten. Die Exaktheit und Knappheit der formalen Beschreibung ermöglicht dem geschulten Mitarbeiter, die ihn betreffenden Eigenschaften des Modells schnell zu erfassen und sie mit den Vor- und Nachbedingungen seiner Operationen abzugleichen.

12 Schluß

Mit diesem Diskussionspapier glauben wir, einen Eindruck von der Komplexität des Problems *Standardsoftware zur Unterstützung der Prüfungsverwaltung an Hochschulen* und Lösungsansätze vermittelt zu haben. Aufgrund einschlägiger Erfahrungen mit dem noch für Großrechner konzipierten Produkt HIS-POS halten wir eine grundlegende Neuentwicklung für unabdingbar. Die Ausschnitte aus der Spezifikation zeigen, daß diese nicht trivial sein wird. Die bereits recht umfangreiche Spezifikation und das Datenmodell unserer Arbeitsgruppe kann in eine Entwicklung mit eingebracht werden.

Um das Verhalten eines Softwaresystems auch in seltenen Zustandskonstellationen zu verstehen, muß die Beschreibung eines Datenmodells über die typischen Festlegungen innerhalb eines ER-Modells und die Definition von Relationen hinausgehen. Diese zusätzlichen Integritätsbedingungen erhält man aber erst dann, wenn man die unterschiedlichen Operationen in Betracht zieht, die auf das Datenmodell einwirken.

Die Komplexität des Systems wird zusätzlich dadurch erhöht, daß es Eigenschaften gibt, die sich auf bestimmte Zeitpunkte oder Zeitabschnitte beziehen, die in ihrer Abfolge fixiert sind. Diese zeittypischen Bedingungen liefern ein weiteres Argument für die Notwendigkeit einer exakten Spezifikation. Des weiteren erfordert die Entwicklung eines Standardsystems, das Allgemeine von den Besonderheiten verschiedener Hochschulen und Fakultäten zu trennen. Das dazu nötige Abstraktionsvermögen wird gerade durch den Zwang zur formalen Strenge nachhaltig unterstützt.

Der Zeitaufwand zur Erstellung formaler Beschreibungen ist sicherlich beträchtlich. Er macht sich aber vor allem dann bezahlt, wenn er zu einer Verringerung des Wartungsaufwandes führt, der heute zum überwiegenden Teil aus der späten Korrektur von Fehlern besteht. Allerdings halten wir es aus ökonomischen Gründen für angebracht zu überlegen, an welcher Stelle unbedingt formal spezifiziert werden sollte und wo eine weniger strenge Vorgehensweise vertretbar ist.

Das Endprodukt *Prüfungsamtsoftware* wird sicherlich auch dadurch verbessert, daß sich die Anwender verstärkt mit dem Problem *Prüfungsordnung* als standardisierbares Problem auseinandersetzen. Da das Selbstverständnis der Hochschulen auch darin besteht, neue Entwicklungsmethoden aufzuzeigen, ergibt sich gerade in ihren eigenen verwaltungsinternen Bereichen die Möglichkeit, neue Methoden auf die praktische Umsetzbarkeit hin zu überprüfen. Es ist daher sinnvoll, die Analyse und Spezifikation des Problems hochschulintern in Zusammenarbeit mehrerer Fakultäten anzugehen und erst die Implementation von einer externen Organisation vornehmen zu lassen, die den Kontakt mit den Hochschulen nicht aus den Augen verliert. Mit dem vorliegenden Papier wollen wir nicht zuletzt auch bezüglich dieser Option erste Anstöße und Anregungen vermitteln.

Literatur

[BaHa91] *deBarros, R.S.M.; Harper, D.J.*: A Method for the Specification of Relational Database Applications. In: *Nichols, J.E.* (ed.): Z User Workshop. Springer, Berlin - Heidelberg et al. 1991.

[BaWö84] *Bauer, F.L.; Wössner, H.*: Algorithmische Sprache und Programmentwicklung. 2.verb.Aufl., Springer, Berlin - Heidelberg - New York - Tokyo 1984.

[CoYo90] *Coad, P.;Yourdon, E.*: Object-Oriented Analysis. Prentice Hall, Englewood Cliffs / NJ 1990.

[Dill90] Diller, A.: Z - An Introduction to Formal Methods. John Wiley & Sons, Chichester et al. 1990.

[FeSi90] *Ferstl. O.K.; Sinz, E.J.*: Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: Wirtschaftsinformatik 32 (1990) 6, 566-581.

[FeSi95] *Ferstl. O.K.; Sinz, E.J.*: Der Ansatz des Semantischen Objektmodells (SOM) zur Modellierung von Geschäftsprozessen. In: Wirtschaftsinformatik 37 (1995) 3, 209-220.

[Fi92] *Fischer, J.*: Datenmanagement, Oldenbourg, München - Wien 1992.

[Jon86] *Jones, C.B.*: Systematic Software Development Using VDM. Prentice Hall, Englewood Cliffs / NJ 1986.

[Lehn91] *Lehner, F.*: Softwarewartung. Hanser, München - Wien 1991.

[MaOp85] *Mambrey, P.; Oppermann, R.*: Partizipation von Benutzern bei der Softwareentwicklung. In Softwaretechnik-Trends 5(1985) 2, 47-62.

[Öst95] *Österle, H.*: Business Reengineering: Prozeß- und Systementwicklung, Band 1. Springer, Berlin - Heidelberg et al. 1995.

[Ort94] *Ortner, E.*: KASPER - Ein Projekt zur natürlichsprachlichen Entwicklung von Informationssystemen. In: Wirtschaftsinformatik 36(1994) 6, 570-579.

[PaSi94] *Pagel, B.-U.; Six, H.-W.*: Software Engineering, Add.-Wesley, Bonn - Reading/Mass. et.al. 1994.

[Rum+91] *Rumbaugh, J.; Blaha, M.; Premerlani, W. et al.*: Object Oriented Modelling and Design. Prentice Hall, Englewood Cliffs / NJ 1991.

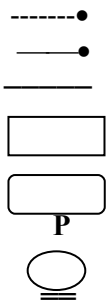
[Sche94] *Scheer, A.-W.*: Wirtschaftsinformatik - Referenzmodelle für industrielle Geschäftsprozesse. 4. Aufl., Springer, Berlin - Heidelberg et al. 1994.

[SiKr95] *Sinz, E.J.; Krumbiegel, J.*: Gestaltung qualitätsgesicherter Universitätsprozesse am Beispiel des prozesses ‘Lehre und Studium’, Diskussionsbeitrag Universität Bamberg Nr, BA-3, März 1995

[Sinz88] *Sinz, E.J.*: Das Strukturierte Entity-Relationsship-Modell (SER-Modell). In: Angewandte Informatik 30(1988) 5, 191-202.
[Spi89] *Spitta, Th.*: Software Engineering und Prototyping. Springer, Berlin - Heidelberg et al. 1989.
[Spiv89] *Spivey, J.M.*: The Z Notation. Prentice Hall, Englewood Cliffs / NJ 1989.
[Vett90] *Vetter, M.*: Aufbau betrieblicher Informationssysteme mittels konzeptioneller Datenmodellierung, 6.Aufl., Teubner, Stuttgart 1990.
[Voss94] *Vossen, G.*: Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme, 2.Aufl., Bonn - Paris et.al. 1994.
[Wall90] *Wallmüller, E.*: Software-Qualitätssicherung in der Praxis, Hanser, München - Wien 1990.

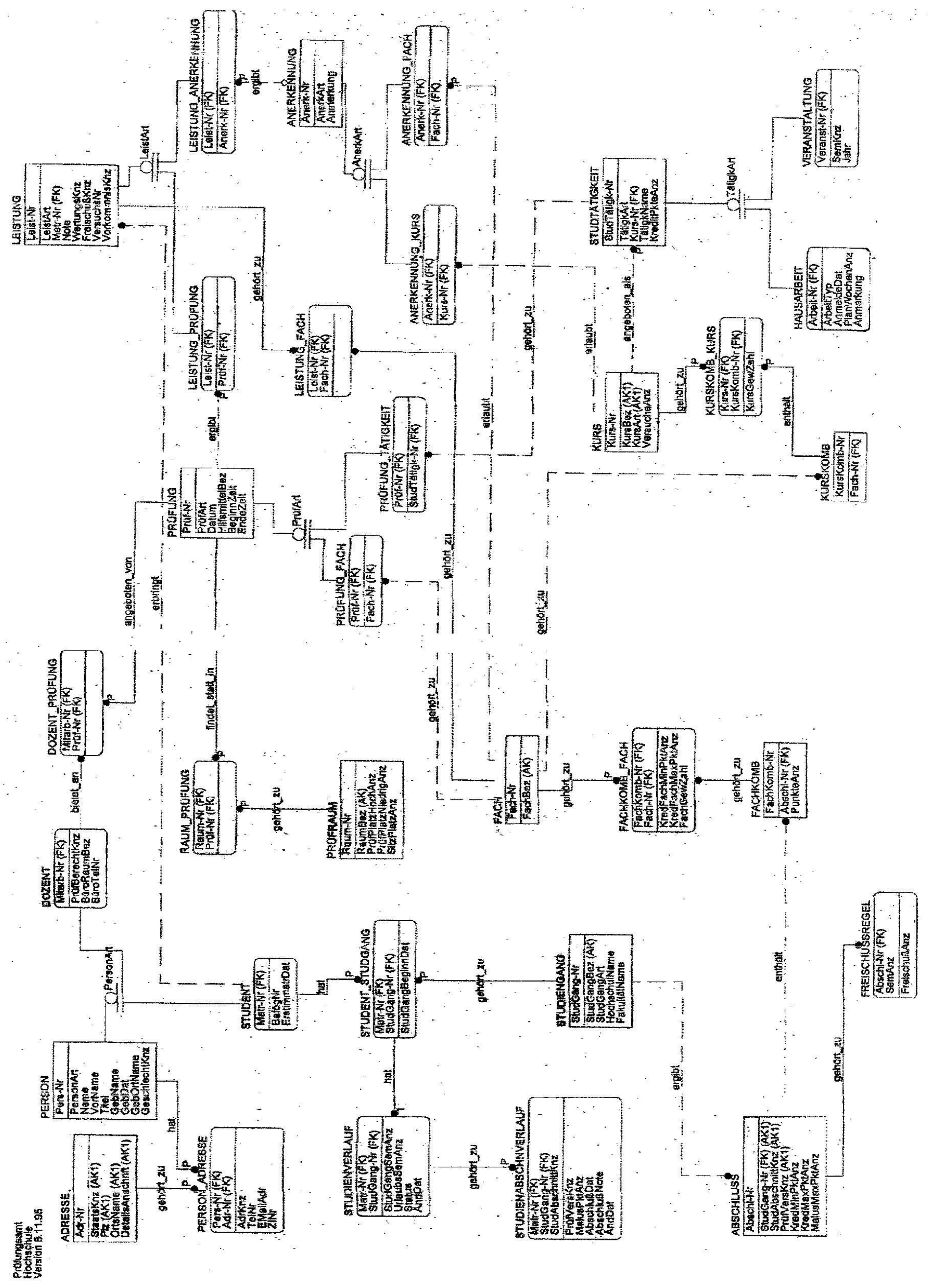
Anhang: Datenmodell Prüfungsamt Hochschule, Version v. 8.11.1995, erstellt und gepflegt mit dem Werkzeug ERWin/SQL

Legende:



1:N -Beziehung (Fremdschlüssel)
1:N -Beziehung (Primärschlüssel)
1:1 -Beziehung
unabhängige Entitätstypen
abhängige Entitätstypen
1 oder N
Spezialisierung

Anhang


LEISTUNG
LEISTUNG_ANERKENNUNG
ANERKENNUNG
ANERKENNUNG_FACH
ANERKENNUNG_KURS
LEISTUNG_PRÜFUNG
LEISTUNG_FACH
STUDTÄTIGKEIT
VERANSTALTUNG
HAUSARBEIT
KURS
KURSKOMB_KURS
KURSKOMB
PRÜFUNG
PRÜFUNG_TÄTIGKEIT
PRÜFUNG_FACH
DOZENT_PRÜFUNG
RAUM_PRÜFUNG
PRÜFRAUM
FACH
FACHKOMB_FACH
FACHKOMB
DOZENT
STUDENT
STUDENT_STUDGANG
STUDIENGANG
FREISCHUSSREGEL
PERSON
ADRESSE
PERSON_ADRESSE
STUDIENVERLAUF
STUDIENABSCHNVERLAUF
ABSCHLUSS