STUDIS-3

StudentInnen-Verwaltungs-System

am Institut für Informatik der Universität Bern

Projektarbeit

im Rahmen des Studiums der Informatik
an der
Universität Bern

von Thomas F Hofmann

Betreut von

Prof. Dr. Oscar Nierstrasz
Dr. Markus Lumpe

April 1999

Inhalt

1 Einleitung 2

2 Requirement-Spezifikation 3

2.1 Anforderungen an das System 3
2.1.1 Übersicht der wichtigsten Funktionalitäten: 3

2.2 Zur derzeitigen Lösung 4
2.2.1 Die wichtigsten Entitäten des Datenmodells: 4
2.2.2 Funktionen 4

2.3 Begründung für den Neuentwurf der zugrundeliegenden Tabellenstruktur 5

2.4 Anforderungen an die Produkteigenschaften von STUDIS-3 5
2.4.1 Funktionalität 5
2.4.2 Fehlerbehandlung und Toleranz 5
2.4.3 Erweiterbarkeit 5

3 Design Dokumente 6

3.1 UML-Diagramme 6

3.2 DB-Struktur 6

3.3 Tabellen STUDIS-2 (soweit für die Migration relevant) **..... 6**

4 Programmcode 7

4.1 Überblick über die enthaltenen Units 7
4.1.1 Uebersicht 7
4.1.2 Beschreibung 7

4.2 Pendenzen 11
4.2.1 Reglemente 11
4.2.2 Definieren neuer Abschlüsse 11
4.2.3 PropertiesDialog der Abschluss-Struktur-Nodes 11
4.2.4 Auswertung 11
4.2.5 Erweiterung der Leistungstabelle um Spalte für Anzahl E 11
4.2.6 Sicherheit 12
4.2.7 Statistiken 12
4.2.8 Errorlog 12
4.2.9 Anleitung / Hilfe 12
4.2.10 Administratormodus 12

5 Erfahrungen bei der Entwicklung von STUDIS-3 13

6 Anhang 15

1 Einleitung

Am Institut für Informatik wird eine Datenbankapplikation benützt, um die Daten der InformatikstudentInnen zu verwalten. Das betrifft einerseits Personendaten wie Adresse, etc. und andererseits Leistunggsdaten, also Testate zu Veranstaltungen innerhalb des Studienbetriebes und Noten zu abgelegten Prüfungen.

Ende Semester wird mit dem Programm jeweils für jedeN StudierendeN ein sogenannter Leistungsausweis (LA) ausgedruckt, worin der aktuelle Stand der absolvierten Leistungen verzeichnet ist. Das System vergleicht dabei jeweils den angestrebten Abschluss und dessen Anforderungen mit den bisher erbrachten Leistungen.

Mit den verwalteten Daten können auch Übersichtstabellen und Statistiken erstellt werden.

STUDIS-2 ist die 2. Version des Systems. Es basiert auf einer Paradox-DB, die Oberfläche wurde mit der Paradox-Entwicklungsumgebung erstellt. Da der Entwickler von STUDIS-2 nicht mehr verfügbar ist, kann STUDIS-2 mit vertretbarem Aufwand nicht mehr gewartet und erweitert werden.
Besonders bei Änderungen der Reglemente, die die Struktur der möglichen Abschlüsse definieren, ist das bestehende System nur mühsam zu benützen.

Meine Aufgabe bestand darin, eine neue Applikation zu erstellen, die die selben Funktionalitäten erfüllen und zudem wartbar und erweiterbar sein soll. Als Entwicklungsumgebung war Borlands Delphi vorgesehen, das eine schnelle Entwicklungsumgebung, Objektorientierung und gute Datenbankanbindung bietet.

Thomas Hofmann, April 1999

hofmann@iamexwi.unibe.ch

2 Requirement-Spezifikation

2.1 Anforderungen an das System

Das STUDIS-System wird am Institut für Informatik der Uni Bern zur administrativen Verwaltung benützt. Es werden damit Personendaten der StudentInnen, Daten über Testate und Noten zu Vorlesungen verwaltet. Leistungsausweise, Notenübersichtsblätter, Statistiken können damit ausgedruckt werden.

2.1.1 Übersicht der wichtigsten Funktionalitäten:

Verwaltung der Personenstammdaten (MatrikelNr., Name, Adresse, Studienfächer, etc.)
- Import der Daten je Fakultät
- Manuelle Eingaben / Ergänzungen
- Nachschlagen von Personendaten

Verwaltung der Leistungsnachweise (Testate, Noten)
- TPN-Import ("Vorlesungsschluss")

Verwalten reglementarischen Vorschriften bezüglich der möglichen Abschlüsse im Fach Informatik (im Haupt- oder Nebenfach)

Erstellen von Leistungsausweisen
- Ausdruck je Ende Semester mit dem Stand der erbrachten Leistungen
- LA 1, LA 2, auch für NF-Studierende
- Berechnungen bzgl. der jeweils gültigen Reglemente
- somit Verwaltung von Reglementen, Anforderungen für mögliche Abschlüsse

Erstellen von Statistiken
- verschiedene Übersichten
- tabellarischer Aushang je Prüfung

Das bestehende System ("STUDIS-2") ist nicht mehr wartbar. Deshalb soll ein neues System mit den gleichen Funktionalitäten erstellt werden ("STUDIS-3").
Dabei soll besonders auf Erweiterbarkeit geachtet werden (→ Änderungen der Reglemente, andere Vorlesungen, etc.)

STUDIS soll im Prüfungssekretariat auf einem PC unter MS-Windows laufen.

STUDIS-2 ist eine Paradox-Applikation unter Windows. Das neue Frontend soll mit Borland Delphi 2.0 erstellt werden. Damit könnten die bestehenden Tabellen weiterverwendet werden.
Weitere Vorteile mit Delphi: einfache Erstellung des GUI, gute Datenbankanbindung, objektorientierte Programmierung.

2.2 Zur derzeitigen Lösung

vgl. Dokumentation von Armin Birkl und Programmcode von STUDIS-2

2.2.1 Die wichtigsten Entitäten des Datenmodells:

StudentIn
Leistung (Note/Testat)
schriftliche Arbeit
Reglement
Abschluss
Äquivalenz (von Leistungen bzgl. eines bestimmten Abschluss)

weitere Entitäten:

Anrede, Status, Ziel, etc. → Hilfstabellen für die Personenstammdaten (PSD.DB)

Struktur, Aequilist → zur Darstellung der Reglementsdaten (RGLMT.DB)

2.2.2 Funktionen

Importieren von Personenstammdaten (SDX-Import)
Importieren von Daten über absolvierte Leistungen: Testate, Noten (TPN-Import)
Definieren von Reglementen, Abschlüssen

Erzeugen von
- Leistungsausweisen (je StudentIn)
- Notenblättern für Aushang (je Prüfung)
- Statistiken (je Vorlesung, Semester, ...)
- Listen der schriftl. Arbeiten, Leistungen, Reglemente, Abschlussdaten
- Administrationslisten

Hilfsfunktionen dazu
- Überprüfung der Importdaten (Matr.Nr., Reglm., Leistg., ...)
- Überprüfung, ob Anforderungen von Abschlüssen erfüllt sind
- Gülitigkeit der Reglemente, Zeitlimite der Testate
- Berechnen von Notendurchschnitten, ...
- Definieren von äquivalenten Leistungen
- Verwaltung von besonderen Ausnahmen

- Users, Passwörter verwalten

2.3 Begründung für den Neuentwurf der zugrundeliegenden Tabellenstruktur

Getrennte Verwaltung der verschiedenen Entitätstypen.
Entkoppelung von Stammdaten und temporären Hilfsdaten (PSD.DB)

Keine redundanten, doppelt geführte Tabellen (TPN / TPNSAR, etc.)

Klare Strukturierung von Referenzen (Leistung / Aequivalenz / Aequivalenzliste)

Änderung des Importformats der Personenstammdaten:
Neu werden die Personendaten in einem Textfile von den betreffenden Fakultäten geliefert. Die einzelnen Felder sind durch Tabulatoren getrennt, wobei die erste Zeile die Feldnamen enthält.

2.4 Anforderungen an die Produkteigenschaften von STUDIS-3

2.4.1 Funktionalität

Die bestehenden Funktionalitäten sollen wieder zur Verfügung stehen.

Studierende, die nicht mehr immatrikuliert sind (Abschluss, Abbruch, ...) sollen aus dem System entfernt werden können.

Vereinfachung beim Nachschlagen von Personendaten.
Beim Ausdrucken der verschiedenen Reports sollen mehr Optionen zur Verfügung stehen (Deckblatt, etc.)

Übernahme der bestehenden Daten muss gewährleistet sein.

2.4.2 Fehlerbehandlung und Toleranz

Plausibilitätscheck bei Import, Eingaben (Format, Wertsbereiche, Datum 2000, ...)
Toleranz bei falschen Inputs (ErrorLogs, ...)

2.4.3 Erweiterbarkeit

Zukünftige Reglementsänderungen müssen einfach integriert werden können.

Schön wäre evtl. OLE-2 Export der Reports → unter WinWord nachbearbeiten

3 Design Dokumente

vgl. referenzierte Seiten im Anhang

3.1 UML-Diagramme

A-1	Überblick über die funktionale Gruppen	
A-2	Data / Migration:	DB-Schnittstellen und Migration von den bestehenden Daten
A-3	Personendaten:	Verwaltung der personenbezogenen Daten und der Leistungen
A-4	Reglementsdaten:	Verwaltung der Reglemente und der entsprechenden Abschlüsse
A-5	Import:	von Personendaten und erbrachten Leistungen
A-6	Leistungsausweise:	Überprüfung des Stand der Leistungen je StudentIn

3.2 DB-Struktur

A-7	ER-Diagramm Übersicht
A-8	ER-Diagramm Details Personendaten
A-9	Tabellenüberblick

3.3 Tabellen STUDIS-2 (soweit für die Migration relevant)

B-1	Abschluss.db
	Aequilst.db
B-2	Aequival.db
	Leistung.db
B-3	PSD.db
B-4	Remark.db
	Rglmt.db
B-5	(Sar.db)
	Struktur.db
B-6	TPN.db
B-7	(TPNSar.db)
	Anrede.db
B-8	Fach.db
	(Faculty.db)
	Kategori.db
B-9	Status.db
	Ziel.db
B-10	Calc.db

4 Programmcode

vgl. „Studis3 Code.doc“

4.1 Überblick über die enthaltenen Units

4.1.1 Uebersicht

Main	{MainForm}
StudiDataMod	{SDM: TDataModule}
Studi	{StudiDlg}
Leistung	
Taten	
Auswertung	{AuswertungsForm}
Personen	{PersDataForm}
Reglement	{RegleForm}
Abschluss	
AbschlussForm	{AbschForm}
Migration	{MigrationForm}
MigrationDataMod	{MigDM: TDataModule}
Node	
PropertiesDlg	{PropDlg}
ImpMainForm	{ImpMainForm}
ProgrDlg	{ProgressDlg}
RawImpThr	
ProcessThr	
TatenImpThr	
Tokenizers	
NeuDialog	{NeuDlg}
SearchDlg	{SuchDlg}

4.1.2 Beschreibung

Main **{MainForm}**

Hauptformular mit Auswahl der Programmbereiche:

<Personendaten>	öffnet 'PersDataForm' → Verwaltung der personenbezogenen Daten
<Reglemente>	öffnet 'RegleForm' → Reglementsverwaltung
<Import>	öffnet 'ImpMainForm' → Importieren der Personen- & Leistungsdaten je Semester
<Migration>	öffnet 'MigrationForm' → Uebertragen der Tabellen von STUDIS-2
<Leistungs Ausweise>	öffnet 'AuswertungsForm' → Generieren der Leistungsausweise (LA)
<anmelden>	Passwortkontrolle → not yet implemented

Bei der Kreierung werden aus dem 'STUDIS-3.ini'-File die Pfade des alten und des neuen DB-Verzeichnis in den Variablen 'ROOT_PATH' und 'OLD_ROOT_PATH' eingelesen.

StudiDataMod {SDM: TDataModule}
DataModule, das alle Table-, Query-, etc. Variablen enthält und den andern Units zur Verfügung stellt.
Allgem. Methode 'TryOpenTable(XX)' mit Erfolgsresultat.
Je Table eine Initialisierungsmeth. 'MakeXXTable'.
Die konkreten Pfade werden bei der Kreierung per 'ROOT_PATH' und 'OLD_ROOT_PATH' gesetzt.

Studi {StudiDlg}
TStudi ist die Objektkapselung eines Personenrecords mit allen vorhandenen Attribute aus der PersonenStammDaten-Tabelle ('StudiTable').
Load- & Save-Methoden.
Zusätzlich 'TatenListe': alle erbrachten Leistungen

TStudiDlg ist das Detailanzeige-Formular für einen Studi.
TheStudi ist der aktuelle Studi zum Dialog.
SetToStudi & SetToDlg stellen die Werte der Dialogfelder entspr. TheStudi ein bzw. umgekehrt.

Personen {PersDataForm}
Formular zur Auflistung aller Studierenden.
Bei DetailBtnClick oder OnDoubleClick auf dem DBGrid wird das Detailanzeige-Formular (TStudiDlg) geöffnet.
Bei SearchBtnClick wird der SuchDlg geöffnet, bei erfolgter Auswahl eines Suchkriteriums wird der Cursor auf den besten Match gesetzt und die Ansicht nach dem Suchkriterium geordnet.

SearchDlg {SuchDlg}
Auswahldialog für Suchkriterien im PersDataForm.
ModRes für ausgewähltes Kriterium, Value für den gesuchten Wert.

Leistung
TLeistung ist die Objektkapselung eines Records aus 'Leistung.db';
Definitionen der möglichen Leistungen (Vorlesungen, Praktika, etc.).
TLeistung wird referenziert von den Reglements-Nodes und von den 'Taten'.

Taten
TTat ist die Objektkapselung eines Records aus 'PersLeist.db';
Erbrachte Leistungen: welcher Studi hat wann welche Leistung (LeistId) mit welcher Note erbracht.
Testate werden als Leistungen mit der Note 0.0 dargestellt.

Auswertung {AuswertungsForm}
Berechnung Leistungsausweise
CheckStudi traversiert den zum betr. Studi passenden Abschluss-Struktur-Baum und vergleicht mit dessen TatenListe, welche Leistungen des angestrebten Abschlusses schon erfüllt sind.

Reglement {RegleForm}
Darstellung eines einzelnen Reglements (RegName, RegId). Hat eine Liste mit den zugehörigen Abschlüssen (Abschlist).

Das RegleForm listet alle Abschlüsse auf, die zum aktuellen Reglement gehören. (Bisher nur für 1 Reglement implementiert, mit der Dropdown-Liste kann dann das Reglement ausgewält werden.)
Bei NeuerAbschBtnClick wird ein NeuDlg geöffnet zum abfragen von Id, Name, etc. Wenn die zurückgegebene AbschlussId noch unbenutzt ist, wird damit ein neuer Abschluss generiert mit der

aktuellen Reglemets-Id. Damit wird ein Abschluss-Formular geöffnet, wo der Abschluss dann definiert werden kann.
Load- & Save-Methoden für die Abschlüsse.

NeuDialog {NeuDlg}
Abfrage von 4 Werten zur Erzeugung eines neues Abschluss.
(ev. generisch machen: auch für Reglemente, Leistungen)

Abschluss
Darstellung eines einzelnen Abschluss. Hat RegId, AbschId, AbschName.
In der StrukturL(iste) wird die Struktur aus für den Abschluss geforderten Leistungen als linkslinearer Baum aus TNode-Elementen dargestellt. Gespeichert wird diese Liste in 'StruktTable', die RootId ist die Id der Root-Node (Node mit dem Abschluss selbst als Leistung): Load- & SaveStrukturList.
MigrateFromCalc transformiert die Struktur der Abschlüsse aus
STUDIS-2 in die neue Form und speichert sie so.

Node
Mit TNode-Objekten wird ein einzelner Knoten in einer Abschluss-Struktur dargestellt. Dafür hat jedes eine eigene NodeId (Primärkey in Struktur.db) und Referenzen auf Nachbarn: Parent, Sibling, Child.
Knoteninhalt sind: LeistId, NodeText (gemäss Leistung der Node),
AnzNoten, AnzTestate (von nachfolgendem Subtree gefordert),
WahlNote, WahlTestat (wahlfreie / obligatorisch Leistung).
Error zeigt Status an.
Wenn der Abschluss äquivalente Leistungen definiert, wird eine Node mit LeistId = ALTERNATIVE eingeschaltet, an die die entsprechenden Leistungen gehängt werden.

AbschlussForm {AbschForm}
Darstellung der Baumstruktur eines Abschluss mit einer TTreeView-Komponente. Die TTreeNodes des TreeView haben als Data-Pointer eine Referenz auf ein TNode-Objekt, dessen Strukturreferenzn mit UpdateNode() gesetzt werden. Mit SetTreeView() werden die Nodes in der StrukturListe als TreeView dargestellt.
Load & Save werden vom Abschluss (TheAbsch) durchgeführt.
Für TAbschluss.SaveStrukturList() wird AbschForm.NewIds() zur Verfügung gestellt zum Bereinigen der NodeIds (wieder sequentielle Nummerierung nach Inserts & Deletes).

Migration {MigrationForm}
Migration der Daten von STUDIS-2 zur neuen Applikation.
Das MigrationForm stellt Buttons zur Verfügung, um die nötigen Schritte einzeln oder zusammen auszulösen. Im Memo-Feld werden Erfolgsmeldungen ausgegeben.
Mit CreateTablesBtnClick werden die neuen Tabellen initialisiert mit je der entsprechenden 'MakeXXTable'-Methode vom StudiDataMod.
Mit SideTablesBtnClick werden die Daten der Nebentabellen kopiert.
Mit ReglementeBtnClick werden die Reglementsdaten in die neue Struktur konvertiert und übertragen. (vorl. nur 1 Regl. gl.zeitig, vgl. SetReglementBtnClick)
Mit PSDBtnClick werden die Personendaten geholt.
Mit LeistungenBtnClick werden die erbrachten Leistungen übertragen.

MigrationDataMod {MigDM: TDataModule}
DataModule für die Migration. Öffnet die Tabellen der alten Applikation.

PropertiesDlg {PropDlg}

Eigenschafts-Dialog für TNode's.
Mit SetToDlgValues resp. SetToNodeValues werden From und referenzierte Node einander angepasst.
LeistList beinhaltet alle vorhandenen Leistungen, die für die Node angewählt werden kann.
Mit NeueLeistBtnClick könnten neue Leistungen definiert werden.

ImpMainForm {ImpMainForm}
Import der neuen Personendaten und der erbrachten Leistungen.

Mit PersImpBtnClick wird das Textfile vom Dekanat geöffnet (OpenFile), der ProgressDialog geöffnet und der Rohimport ausgelöst (ProgrDlg.StartPersImport).
Mit ProcessBtnClick ('process RawTable') wird ein ProgressDialog geöffnet und mit dessen StartProcessing die Umwandlung des RawTables ins relationale DB-Schema gestartet.

Mit LeistImpBtnClick wird ein ProgressDialog geöffnet. Mit dessen StartLeistImport werden die neuen Noten und Testate eingetragen.

ProgrDlg {ProgressDlg}
Mit StartPersImport wird der RawImportThread gestartet und die Fortschrittsanzeige initialisiert.
Mit StartProcessing wird der ProcessThread gestartet.
Mit StartLeistImport wird der TatenImportThread gestartet.

RawImpThr
In der Execute-Methode wird der RawTable generiert und die Personendaten werden mit ImportRawData zeilenweise übertragen. Dazu wird ein TPersTokenizer benützt.

ProcessThr
In der Execute-Methode werden aus den RawTable-Zeilen allen Seitentabellen die neuen Werte angehängt. Die Personendaten werden in Studi-Objekte gepackt (MakeStudis) und im StudiTable eingetragen oder korrigiert (SaveStudis).

TatenImpThr
In der Execute-Methode werden mit ImportTaten die neuen erbrachten Leistungen im PersLeistTable eingetragen.

Tokenizers
TTokenizer ist die abstrakte Oberklasse, die für TPersTokenizer (Import der Personendaten) und für TLeistTokenizer (Import der erbrachten Leistungen) spezialisiert wird.

4.2 Pendenzen

4.2.1 Reglemente

zZ wird bei der Migration der alten Daten nur ein einzelnes Reglement übertragen. Die Methode 'TMigrationForm.MigReglemente' muss um einen Loop über alle vorhandenen Reglemente erweitert werden. Zusätzlich muss jedes migrierte Reglement in 'RegleTable' eingetragen werden.

Dazu kommen im RegleForm:
Auswahl des angezeigten Reglements mit der ComboBox.
Möglichkeit zum Definieren eines ganzen neuen Reglements.
Löschen einzelner Abschlüssen aus einem Reglement.

4.2.2 Definieren neuer Abschlüsse

Eventuell Copy/Paste-Funktionalität für einzelne Nodes. Sinnvoller wäre wohl die Möglichkeit, einen ganzen Abschluss aus einem (anderen) Reglement übernehmen zu können, um diesen modifiziert ins aktuelle Reglement einzufügen.
(zZ kann ein existierender Abschluss im NeuDialog ausgewählt werden; die Baumstruktur muss aber neu wieder eingegeben werden.)

4.2.3 PropertiesDialog der Abschluss-Struktur-Nodes

Je nach Art des angezeigten Nodes können gewisse Felder ausgeblendet werden, die nicht gebraucht werden. Blattknoten brauchen keine Angaben über die Anzahl Noten, die von Nachfolgern verlangt wird; Rootknoten können nicht wahlfrei sein; etc.

4.2.4 Auswertung

Das Erzeugen der Leistungs-Ausweise (LA) ist noch nicht komplett.

Mit der implementierten Testfunktion wird erst die Liste der erbrachten Leistungen eines einzelnen Studierenden mit der Baumstruktur des Abschluss verglichen. Der Baum wird rekursiv durchlaufen (postorder) und zu jedem Knoten die entsprechenden Leistungen in der 'TatenListe' des Studi gesucht. Die gefundenen werden mit Note, etc. in einer Stringliste aufgeführt. Für jeden Knoten der Abschluss-Struktur wird gezählt, wieviele Noten, Testate unterhalb vorhanden sind. Die Strings können einfach für die Erstellung der LA benutzt werden.

Was fehlt:
Laden des entsprechenden Reglementes je Studi.
Erzeugung von Leistungen, die aus Teilleistungen bestehen (Knoten) und nicht importiert werden.
Berechnung der Notenschnitte, etc. je Knoten.
Ausgabe im Form von Leistungsausweisen.
Möglichkeit, alle anfallenden LA automatisch zu erzeugen.
Nachbearbeitungsoption der LA.
Ausdrucken der LA.

4.2.5 Erweiterung der Leistungstabelle um Spalte für Anzahl E

Um die standardisierte Aufwand-Vergleichseinheit 'E' auf den LA ausgeben zu können, muss der LeistungTable um eine Spalte erweitert werden. Die entsprechenden Angaben müssen eingegeben werden können.

4.2.6 Sicherheit

Um das System mit Passwortschutz auszustatten, können die dafür vorgesehenen Eingabefelder auf dem MainForm benützt werden. Zusätzlich muss noch ein Verwaltungsdialog eingebaut werden (User eintragen, entfernen; Passwörter ändern). Die entsprechenden Daten müssen gespeichert werden, zB in einer zusätzlichen Tabelle.
Die Tabellen sollten mit Passwortschutz versehen werden.

4.2.7 Statistiken

Statistiken über Noten von bestimmten Vorlesungen, über Anzahl teilnemender Studierenden, etc. können aus den bestehenden Tabellen extrahiert werden. Dazu braucht es zusätzlich eine Unit mit den entsprechenden DB-Zugriffs- und Berechnungs-Methoden und ein Formular für die geeignete Darstellung.

4.2.8 Errorlog

Für die Migration und den Import sollte ein Errorlog geführt werden, um alle aufgetretenen Fehler aufzuzeichnen.
Evtl. auch ein allgemeines Log mit Aufzeichnungen zum Ablauf, etc.

4.2.9 Anleitung / Hilfe

Für den Einsatz des Programms wäre ein Hilfesystem oder eine ausführliche Anleitung angebracht.

4.2.10 Administratormodus

Die Migration gehört nicht zum normalen Gebrauch des Systems. Dies sollte in einem Administratormodus geschehen.

5 Erfahrungen bei der Entwicklung von STUDIS-3

Das StudentInnen-Verwaltungs-System 'STUDIS-3' soll das bestehende System 'STUDIS-2' ablösen, das am Institut für Informatik benützt wird. Mit dieser Datenbank-Applikation wird am Ende jedes Semesters für jedeN StudierendeN ein sogenannter Leistungsausweis (LA) ausgedruckt, worin der aktuelle Stand der absolvierten Leistungen verzeichnet ist. Ebenso dient das Programm der SekretärIn am Institut zum Nachschlagen von Personendaten. Es können Statistiken erstellt werden über Anzahl TeilnehmerInnen von Veranstaltungen, über die erreichten Noten, Notenschnitte, etc. Für den Aushang der erreichten Noten einer Prüfung können entsprechende Listen ausgedruckt werden.

Da der Entwickler von STUDIS-2 nicht mehr verfügbar ist, kann STUDIS-2 mit vertretbarem Aufwand nicht mehr gewartet und erweitert werden. Meine Aufgabe bestand darin, eine neue Applikation zu erstellen, die die selben Funktionalitäten erfüllen und zudem wartbar und erweiterbar sein soll. Als Entwicklungsumgebung war Delphi von Borland vorgesehen.

Zunächst ging es darum, den Code der bestehenden Applikation, die in Paradox für Windows 5.0 entwickelt wurde, überhaupt zugänglich zu machen. Das bedeutete umständlicherweise, im Editmodus von Paradox bei jedem Button, Editfeld, Formular, Dialog, etc., zur Codeansicht zu wechseln und, falls eine Funktion oder Prozedur zum Vorschein kam, diese mit Copy-Paste auf ein gemeinsames Dokument zu übertragen (viel Zeit & Geduld ...). Zusätzlich waren Libraries vorhanden, die global benützt werden konnten.

Bei der Analyse des Codes zeigte es sich, dass er sehr wenig kommentiert ist. In der Dokumentation steht über den Code selbst so gut wie fast nichts. Es hat diverse globale Variablen (in den Libraries), bei denen es sehr mühsam ist, herauszufinden, welche Methoden auf sie zugreifen. Viele Codestücke sind im Copy-Paste-Stil geschrieben und sind einander sehr ähnlich.

In der Datenbank hat es neben den eigentlichen (Daten-)Tabellen viele temporären Tabellen, in denen (Zwischen-) Ergebnisse von Berechnungen, etc. gespeichert werden, die aber dauerhaft bestehen bleiben. Die Datentabellen enthalten teilweise verschiedene Felder mit temporären Flags, beispielsweise zum Markieren, ob ein Berechnungsschritt mit dem betreffenden Record schon ausgeführt worden ist oder nicht. Viele Daten sind in mehreren Tabellen redundant vorhanden.
Es scheint, dass durch die Beschränkungen von Paradox viele Verrenkungen nötig gewesen waren, um die nötigen Funktionalitäten zu implementieren. Kurzum, STUDIS-2 behielt viele Geheimnisse für sich.

Delphi (ich benützte Version 2.0) ist eine sehr mächtige Umgebung. Obwohl ich zum ersten Mal damit arbeitete, konnte ich recht schnell damit umgehen. Ich hatte zuvor ein wenig Pascal-Erfahrung. Borlands Object-Pascal scheint mir eine gelungene Erweiterung zur Objektorientierung. Zusammen mit dem visuellen Editieren von Formularelementen können damit sehr schnell Programmoberflächen erstellt werden; man kann sich also grossenteils auf die eigentlichen Funktionalitäten des Programms konzentrieren.
Die Datenbankanbindung ist sehr komfortabel. DB-Elemente wie Tables, Queries, etc. sind in Objekten des Frameworks gekapselt und können vollständig vom Programmcode aus benützt werden. Mit den Queries können SQL-Statements ausgeführt werden, sowohl Abfragen wie auch Updates von Tabellen. Zur Anzeige (und direkten Veränderung) der Tabellendaten stehen DB-Grids zur Verfügung.

Um mich gleichzeitig mit Delphi vertraut zu machen, begann ich Prototypen von einzelnen Programmteilen zu programmieren. Als erstes entwickelte ich ein Modul für den Import von Personendaten. Diese sind bis anhin aus SDX-Dateien vom Dekanat der Universität importiert worden. Neuerdings wurde aber ein Textfile geliefert mit einer Zeile je Record, die Felder durch Semikolons getrennt. Diesbezügliche Ankündigungen gab es keine, das alte Programm, womit der

Datenexport zuvor gemacht worden war, war nicht mehr im Betrieb... Die Konsistenz mit den alten Daten war durch Neunummerierung der Keys und gleichzeitige Veränderung der Werte (zB 'Student/Studentin' statt 'Student/in') stark erschwert.
Die Erweiterung für den Import von erbrachten Leistungen war dann mit kleinem Aufwand verbunden.

Ein zweiter Prototyp betraf die Darstellung von Abschlüssen. In STUDIS-2 wurde dies naheliegenderweise mit einer Baum-Datenstruktur gelöst. Der eigentliche Abschluss ist die Root. Die Teilleistungen, die zur Erfüllung eines Knoten erfordert sind, sind je dessen Children. Die effektiven Leistungen (Noten / Testate) sind die Leaves. Jeder Knoten weiss, vieviele Noten, wieviele Testate von seinen Nachfolgern erfordert sind.
Diese Struktur wurde in einer Tabelle so gespeichert, dass jeder Record einen Knoten ('Teilleistung') darstellt und eine Referenz auf das Reglement, den Abschluss und den Vorgängerknoten ('Leistung') hatte. Für jede Leistung, zu der sowohl ein Testat, als auch eine Note verlangt ist, wurden zwei Leaves gespeichert.
Im Fall, dass von einem Knoten K mit zwei Nachfolgeknoten insgesamt eine bestimmte Anzahl Teilleistungen verlangt wird, aber offen sein soll, von welchem wieviele, musste für jede mögliche Variante der ganze Subtree eingegeben werden.

Ich betrieb einen gewissen Aufwand, um diese Baum-Struktur optisch darstellen zu können. Mit Delphi hatte ich eine visuelle Komponente TTreeView zur Verfügung. Damit können hierarchische Strukturen ähnlich wie in einem Filemanager dargestellt werden.
Ich reduzierte die Redundanz in der Struktur der Abschlüsse durch gleichzeitige Behandlung von Noten und Testaten, zu jeder geforderten Note ist ja auch ein Testat notwendig. Auch die Variantenlösung wäre zu umgehen, mit der Zeit zeigte sich aber, dass der Aufwand insgesamt zu gross war, da Varianten nur in einem einzigen Abschluss vorkommen. Die Überprüfung mit einer einfacheren Struktur wäre machbar. Dazu könnte ein zusätzlicher Knoten als gemeinsamen Vorgänger einschaltet werden. Dieser würde die summarischen Erforderungen prüfen. Die unteren Knoten hätten keine bestimmte Anforderung, sondern je einen Verweis auf den Vorgängerknoten. Die automatische Konvertierung der bestehenden Daten, die ja übernommen werden müssen, wurde sehr umständlich, so dass ich von der Umwandlung der Varianten schliesslich absah. Bei einer späteren Optimierung des Programms könnte dies aufgenommen werden.

Was im alten System eher schwerfällig war, war die Implementierung der Möglichkeit von alternativen (äquivalenten) Leistungen, in den Übergangsreglementen sind solche häufig definiert. In STUDIS-2 gibt es dafür extra eine Tabelle, in der bei jeder Baumtraversierung nachgeschaut werden muss, ob allenfalls eine Äquivalenz existiert.
Ich habe die Baumstruktur der Abschlüsse dahingehend erweitert, dass für solche Äquivalenzen je ein zusätzlicher Knoten („ALTERNATIVE“) eingefügt wird, an welchen alle äquivalenten Leistungen gehängt werden. Bei der Traversierung ist dann klar, dass eine der zwei (oder mehreren) Leistungen genügt.

Das schrittweise Erweitern und Zusammensetzen der Teile war mit Delphi gut unterstützt. Ich konnte so (fast) immer mit einem „working system“ arbeiten. Die Erstellung der Tabellen habe ich in einer Delphiklasse automatisiert, so konnte ich die DB und das Programm auf jedem Stand synchronisieren.

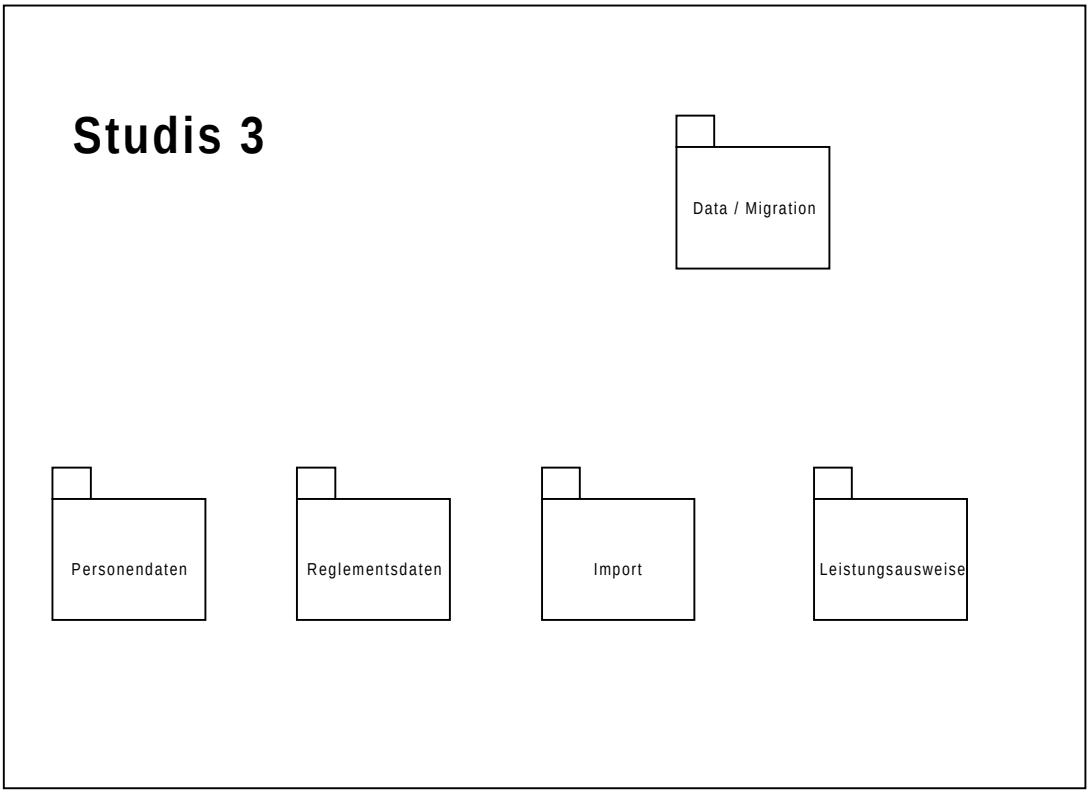
Bei der Migration zeigte sich, dass der Aufwand mit den Erweiterungen und Änderungen, den ich anfangs betrieben hatte, verfrüht gewesen war. Im Sinne von Reingeneering wäre es effizienter gewesen, zunächst die bestehenden Funktionalitäten nachzubilden und erst dann Erweiterungen einzubauen.

Im Ganzen war die Arbeit an diesem Projekt sehr interessant und lehrreich. Der zeitliche Aufwand, den ich immer wieder unterschätzt hatte, war für den Rahmen eines Informatikprojekts gemäss Normalstudienplan sehr hoch.

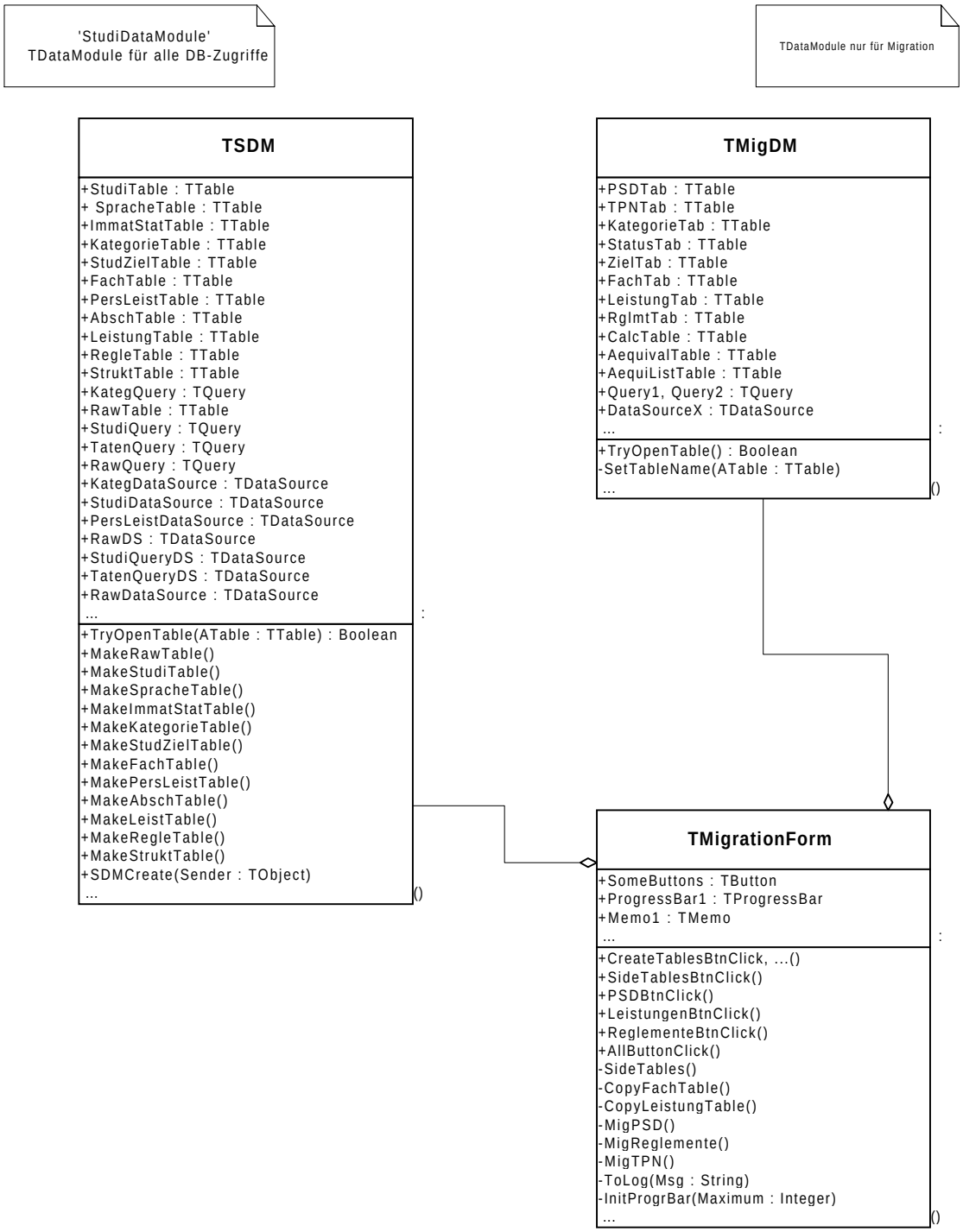
6 Anhang

UML-Diagramme

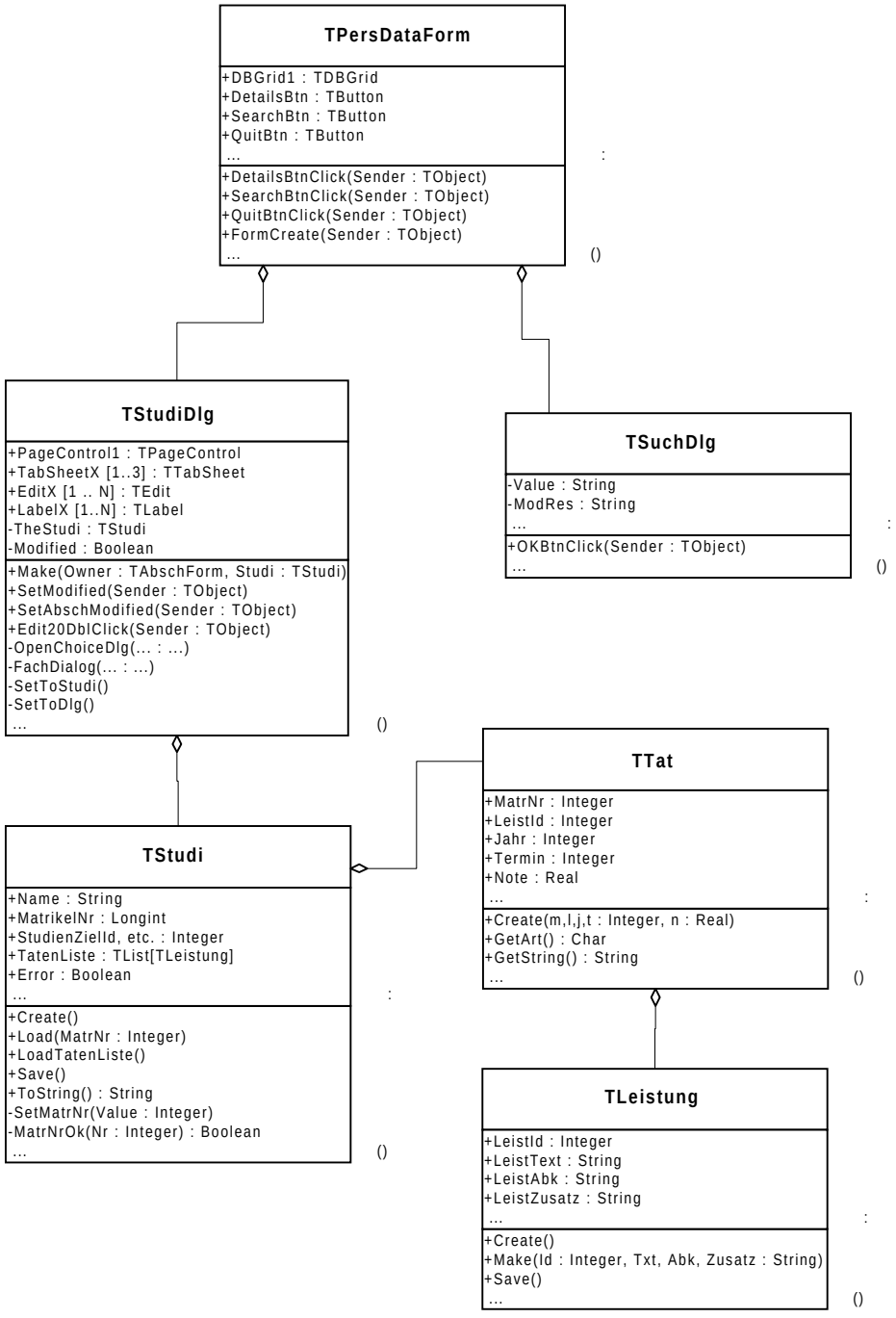
Übersicht über die funktionalen Gruppen



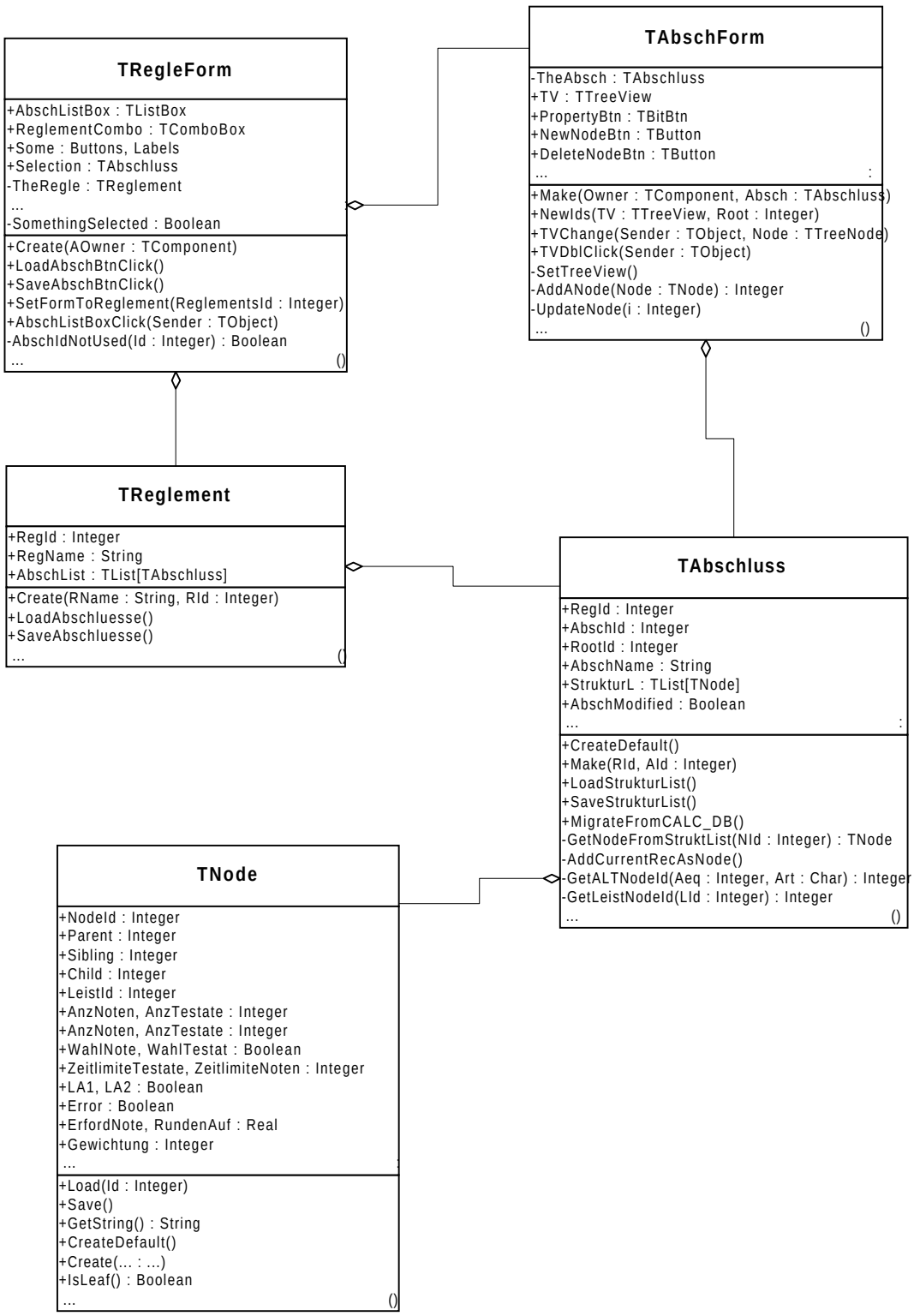
Data / Migration



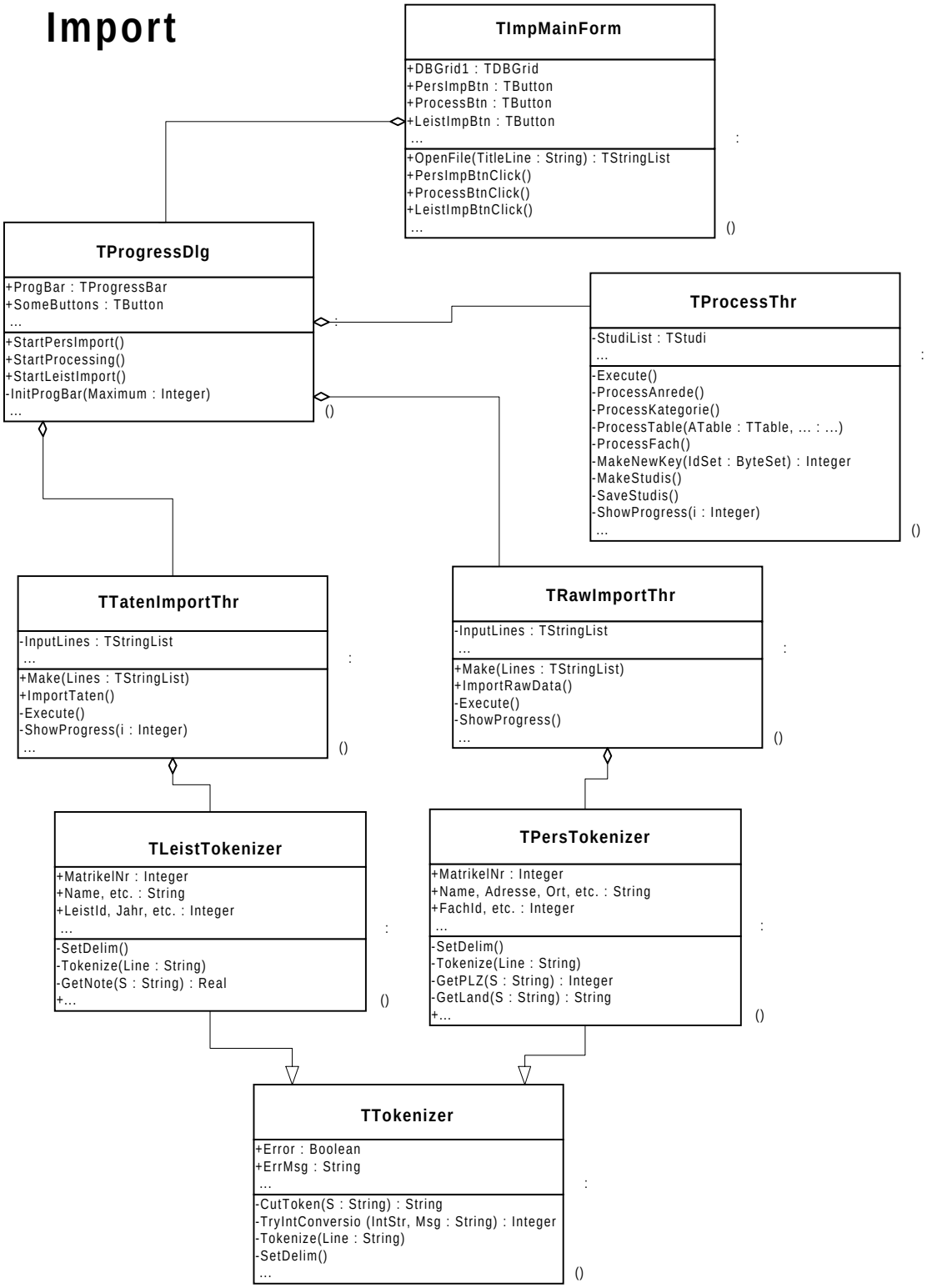
Personendaten



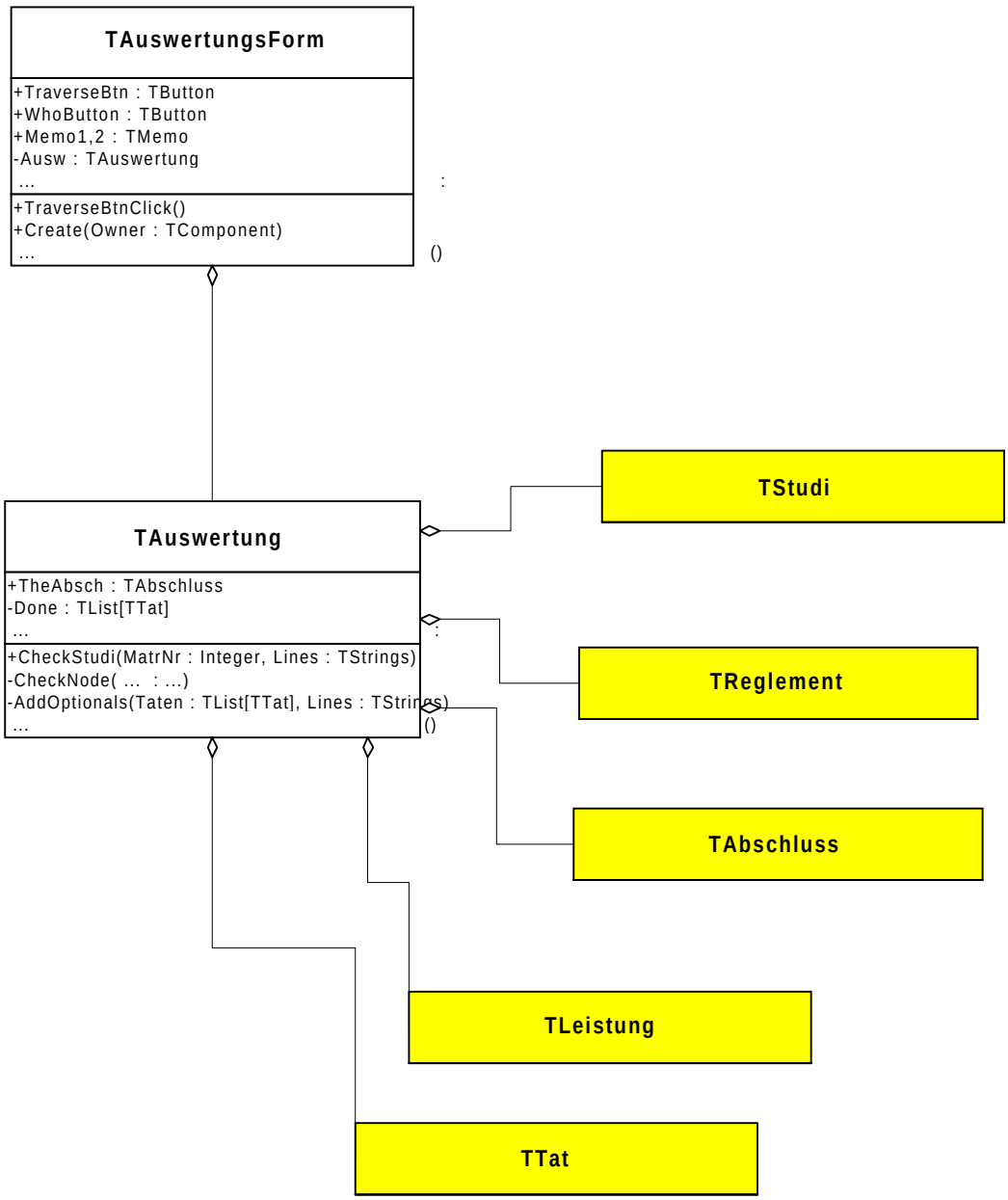
Reglementsdaten



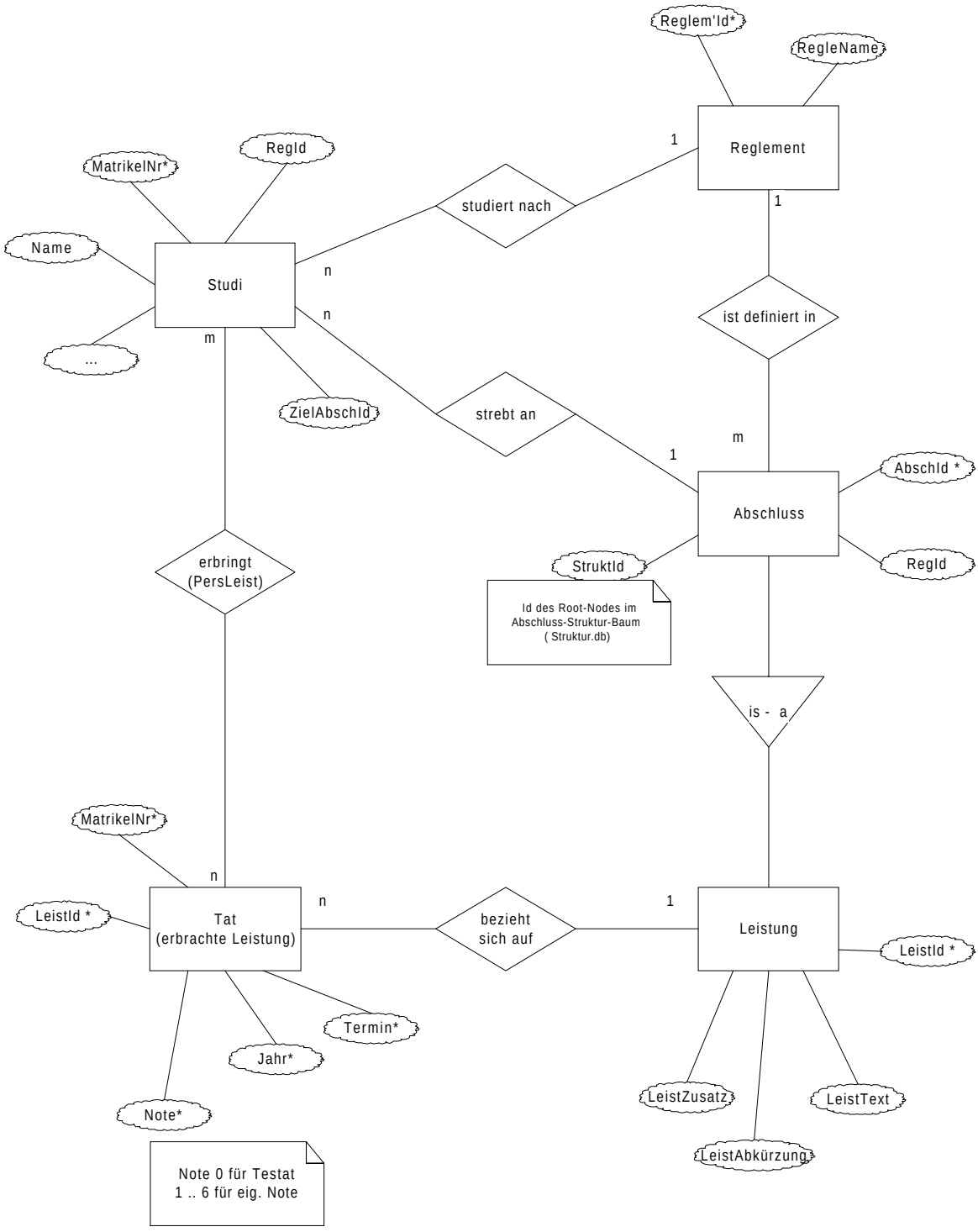
Import


TImpMainForm
+DBGrid1 : TDBGrid
+PersImpBtn : TButton
+ProcessBtn : TButton
+LeistImpBtn : TButton
...
+OpenFile(TitleLine : String) : TStringList
+PersImpBtnClick()
+ProcessBtnClick()
+LeistImpBtnClick()
...
TProgressDlg
+ProgBar : TProgressBar
+SomeButtons : TButton
...
+StartPersImport()
+StartProcessing()
+StartLeistImport()
-InitProgBar(Maximum : Integer)
...
TProcessThr
-StudiList : TStudi
...
-Execute()
-ProcessAnrede()
-ProcessKategorie()
-ProcessTable(ATable : TTable, ... : ...)
-ProcessFach()
-MakeNewKey(IdSet : ByteSet) : Integer
-MakeStudis()
-SaveStudis()
-ShowProgress(i : Integer)
...
TTatenImportThr
-InputLines : TStringList
...
+Make(Lines : TStringList)
+ImportTaten()
-Execute()
-ShowProgress(i : Integer)
...
TRawImportThr
-InputLines : TStringList
...
+Make(Lines : TStringList)
+ImportRawData()
-Execute()
-ShowProgress()
...
TLeistTokenizer
+MatrikelNr : Integer
+Name, etc. : String
+LeistId, Jahr, etc. : Integer
...
-SetDelim()
-Tokenize(Line : String)
-GetNote(S : String) : Real
+...
TPersTokenizer
+MatrikelNr : Integer
+Name, Adresse, Ort, etc. : String
+FachId, etc. : Integer
...
-SetDelim()
-Tokenize(Line : String)
-GetPLZ(S : String) : Integer
-GetLand(S : String) : String
+...
TTokenizer
+Error : Boolean
+ErrMsg : String
...
-CutToken(S : String) : String
-TryIntConversio (IntStr, Msg : String) : Integer
-Tokenize(Line : String)
-SetDelim()
...

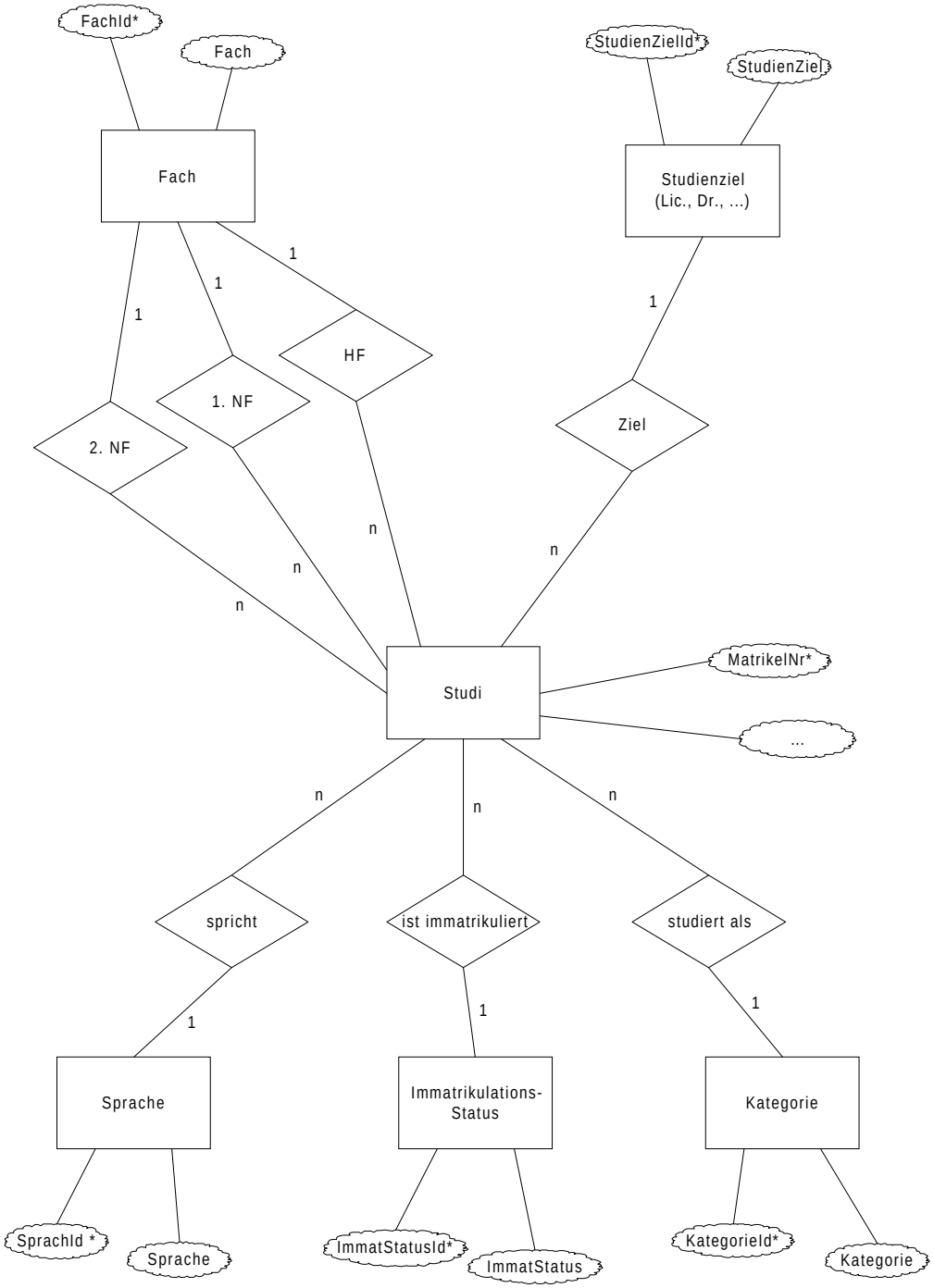
Leistungsausweise



ER-Diagramm : Übersicht



ER-Diagramm : Details Personendaten



Tabellenübersicht

Form:

Name

A	B	C	D

mit A : Feldname
*A : Primary Key
+A : Foreign Key
B : Typ
C : Grösse
D : Wert erfordert (+) oder frei (-)

StruktTable

*NodeId	Integer	0	+
+Parent	Integer	0	+
+Sibling	Integer	0	+
+Child	Integer	0	+
+LeistId	Integer	0	+
AnzNoten	Integer	0	-
AnzTestate	Integer	0	-
WahlNote	Boolean	0	-
WahlTestat	Boolean	0	-
LA1	Boolean	0	-
LA2	Boolean	0	-
LA1Fertig	Boolean	0	-
LA2Fertig	Boolean	0	-
ZeitlimiteTestate	Integer	0	-
ZeitlimiteNoten	Integer	0	-
ErfordNote	Float	0	-
RundenAuf	Float	0	-
Gewichtung	Integer	0	-

RawTable

*MatrikelNr	Integer	0	+
Name	String	24	+
Vorname	String	24	+
+AnredeId	Smallint	0	-
Anrede	String	10	-
BriefAnrede	String	24	-
Adresse_1	String	32	-
Adresse_2	String	32	-
Land	String	4	-
PLZ	Integer	0	-
Ort	String	32	-
GeburtsDatum	String	10	-
HeimatOrt	String	32	-
Tel	String	20	-
+SprachId	Smallint	0	-
Sprache	String	24	-
+StudienZielId	Smallint	0	-
StudienZiel	String	32	-
+HFId	Smallint	0	-
HF	String	30	-
HFSemAnzahl	Smallint	0	-
+NF1Id	Smallint	0	-
NF1	String	30	-
NF1SemAnzahl	Smallint	0	-
+NF2Id	Smallint	0	-
NF2	String	30	-
NF2SemAnzahl	Smallint	0	-
+SemesterId	Smallint	0	-
SemesterBez	String	25	-
+ImmatStatusId	Smallint	0	-
ImmatStatus	String	25	-
+KategorieId	Smallint	0	-
Kategorie	String	75	-
ElternWohnort	String	32	-
StudienJahr	Smallint	0	-

StudiTable

*MatrikelNr	Integer	0	+
Name	String	24	+
Vorname	String	24	+
Adresse_1	String	32	-
Adresse_2	String	32	-
Tel	String	20	-
Land	String	4	-
PLZ	Integer	0	-
Ort	String	32	-
GeburtsDatum	String	10	-
HeimatOrt	String	32	-
ElternWohnort	String	32	-
AnredeId	Smallint	0	-
+SprachId	Smallint	0	-
+StudienZielId	Smallint	0	-
+ImmatStatusId	Smallint	0	-
+KategorieId	Smallint	0	-
+ZielAbschId	Integer	0	-
+RegId	Integer	0	-
+HFId	Smallint	0	-
HFSemAnzahl	Smallint	0	-
+NF1Id	Smallint	0	-
NF1SemAnzahl	Smallint	0	-
+NF2Id	Smallint	0	-
NF2SemAnzahl	Smallint	0	-
InsertDate	DateTime	0	-
UpdateDate	DateTime	0	-
AbschChangeDate	DateTime	0	-
LA1Date	DateTime	0	-
LA2Date	DateTime	0	-

PersLeistTable

*MatrikelNr	Integer	0	+
*LeistId	Integer	0	+
*Jahr	Smallint	0	+
*Termin	Smallint	0	+
*Note	Float	0	+

SpracheTable

*SprachId	Smallint	0	+
Sprache	String	24	+

ImmatStatTable

*ImmatStatusId	Smallint	0	+
ImmatStatus	String	30	+

KategorieTable

*KategorieId	Smallint	0	+
Kategorie	String	75	+

StudZielTable

*StudienZielId	Smallint	0	+
StudienZiel	String	48	+

FachTable

*FachId	Smallint	0	+
Fach	String	48	+
FakultaetsId	Smallint	0	+

AbschTable

*AbschId	Integer	0	+
+StruktId	Integer	0	+
+RegId	Integer	0	+

LeistTable

*LeistId	Integer	0	+
LeistText	String	72	+
LeistAbkürzung	String	5	-
LeistZusatz	String	25	-

RegleTable

*RegleId	Integer	0	+
RegleName	String	96	+