Die funktionale Essenz von HDMS-A[‡]

Oscar Slotosch[*]
F. Nickl[**]
S. Merz[**]
Heinrich Hußmann[*]
Rudolf Hettler[*]

18. Februar 1994

Zusammenfassung

Dieser Bericht ist das Ergebnis einer Fallstudie zur Spezifikation von großen Softwaresystemen. Im Rahmen der Fallstudiengruppe "HDMS-A" des KORSO-Projektes wurde angelehnt an das Beispiel des Deutschen Herzzentrums Berlin eine elektronische Patientenakte konzipiert und in der algebraischen Spezifikationssprache SPECTRUM spezifiziert. Die funktionale Essenz enthält die formalen Spezifikationen der, für die Computerunterstützung, wesentlichen Abläufe in einer vereinfachten Sicht auf das Krankenhaus. Sie ist die Grundlage für eine SOLL-Spezifikationen des Gesamtsystems.

Dieser Bericht ist eng mit den drei Berichten [Het93, Huß93, Nic93] verbunden: Das hier angewendete Vorgehen beim Erstellen großer Spezifikationen wird in [Huß93] beschrieben. Die dabei verwendete Kombination aus ER-Diagrammen und SPECTRUM-Spezifikationen wird in [Het93] beschrieben. Die Semantik der Ablaufdiagramme wird in [Nic93] beschrieben.

[‡]Diese Arbeit wird vom Bundesministerium für Forschung und Technik als Teil des Verbundprojekts "KORSO - Korrekte Software" gefördert

[*]Institut für Informatik der Technischen Universität München, D-80290 München

[**]Institut für Informatik der Ludwig-Maximilians-Universität München, Leopoldstraße 11b, D-80802 München

Inhaltsverzeichnis

1 Einleitung

Dieser Bericht beschreibt die funktionale Essenz des HDMS-A Systems zur Verwaltung einer elektronischen Patientenakte. Eine Beschreibung des KORSO-Fallbeispiels HDMS-A findet sich in [LCFW92] und [CKL93]. Der Begriff der funktionalen Essenz sowie das methodische Vorgehen, das in diesem Beispiel zur Entwicklung der funktionalen Essenz angewendet wurde, ist in [Huß93] beschrieben. Die folgenden Bemerkungen dienen dem besseren Verständnis dieses Berichts und vor allem der in ihm enthaltenen SPECTRUM-Spezifikationen.

- Abschnitt 2.1 beschreibt das Gesamtdatenmodell des spezifizierten Systems in Form eines Entity-Relationship Diagramms und tabellarischer Entitybeschreibungen. Diese Informationen können, wie in [Het93] beschrieben, in eine formale SPECTRUM-Spezifikation übersetzt werden. Es wird angenommen, daß die bei der Übersetzung des E/R-Diagramms entstehende Spezifikation den Namen DB trägt. DB stützt sich auf die Spezifikationen der einzelnen Entitytypen ab (siehe Abbildung 1).

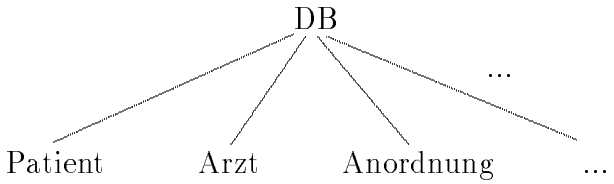

Abbildung 1: Stützdiagramm der SPECTRUM-Spezifikation des Gesamtdatenmodells

- Die in 2.2 beschriebenen externen Dokumente sind zwar keine Entities des zu spezifizierenden Systems, können aber genauso beschrieben werden. Sie werden ebenfalls dem in [Het93] beschriebenen Übersetzungsmechanismus unterworfen.

- Die Datenflußdiagramme dienen zum Verständnis der einzelnen Teilabläufe. Sie können, wie in [Nic93] beschrieben, zusammen mit den hier gegebenen Spezifikationen der elementaren Systemtransaktionen und Umweltfunktionen in eine strombasierte Spezifikation übersetzt werden, die das dynamische Verhalten des Systems beschreibt. Ausgehend von dieser Spezifikation können Beweisverpflichtungen an die Konsistenz der Spezifikation der elementaren Transaktionen mit der Ablaufbeschreibung durch die Datenflußdiagramme generiert werden. Beim Prüfen dieser Beweisverpflichtungen sind uns einige Inkonsistenzen in den Spezifikationen aufgefallen.

- Zur Behandlung der Zeit werden in dieser Fallstudie folgende Annahmen getroffen. Es gebe eine Sorte

```
sort DateTime;
```

die zur Beschreibung der Tageszeit zusammen mit dem Kalenderdatum dient. Zur Bestimmung der aktuellen Zeit nehmen wir eine in die Datenbank "eingebaute" Uhr an. Technisch gesehen bedeutet dies, daß die Datenbank neben den im E/R-Schema von Abbildung 2 gegebenen Entitytypen einen zusätzlichen Entitytyp Clock speichert. Dieser Entitytyp wurde aus Gründen der Übersichtlichkeit in Abbildung 2 weggelassen. Zu jedem Zeitpunkt enthalte die Datenbank genau eine Entity des Typs Clock, in der die aktuelle Zeit gespeichert ist. Zur Fortschreibung der Zeit wird eine elementare Transaktion

```
tick: Db → Db;
```

angenommen. Das Auslesen der aktuellen Zeit aus der Datenbank wird durch eine Funktion

```
datetime: Db → DateTime;
```

bewerkstelligt. Auf der Sorte DateTime gebe es ferner eine totale Ordnung

```
.before.: DateTime × DateTime → Bool   prio 6;
```

zum Vergleich zweier Zeiten sowie ein Prädikat

```
is_today : DateTime × Db → Bool;
```

zum Überprüfen, ob ein Datum von heute ist.

- Über den in [BFG+93a] und [BFG+93b] definierten Sprachumfang von SPECTRUM hinaus wird in den Spezifikationen ein `where`-Konstrukt mit der Bedeutung

```
<exp> where <defns>  ≡
letrec <defns> in <exp> endlet
```

verwendet. Außerdem wird sowohl das `letrec`-Konstrukt als auch das `where`-Konstrukt zur Einführung von Tupeln von Bezeichnern verwendet, zum Beispiel

```
letrec (a, b) = <Tupel-Exp> in <exp> endlet
```

Dieses Konstrukt kann natürlich durch Verwendung geeigneter Projektionsfunktionen auf "reines" SPECTRUM zurückgeführt werden.

- Die in dieser Arbeit angegebenen Spezifikationen stützen sich teilweise auf Spezifikationen aus der Standard Library von SPECTRUM, die in [BFG+93b] angegeben ist (natürliche Zahlen, Listen, ...). Für Listen wird darüberhinaus ein Enthaltenseins-Prädikat

```
isin:  α::EQ ⇒ α × List α → Bool;
  axioms α::EQ ⇒ ∀ x,y:α, s: List α in
    isin(x,[]       ) = false;
    isin(x,cons(y,s) = (x==y ∨ isin(x,s));
  endaxioms;
```

angenommen.

2 Logisches Datenmodell des Gesamtsystems

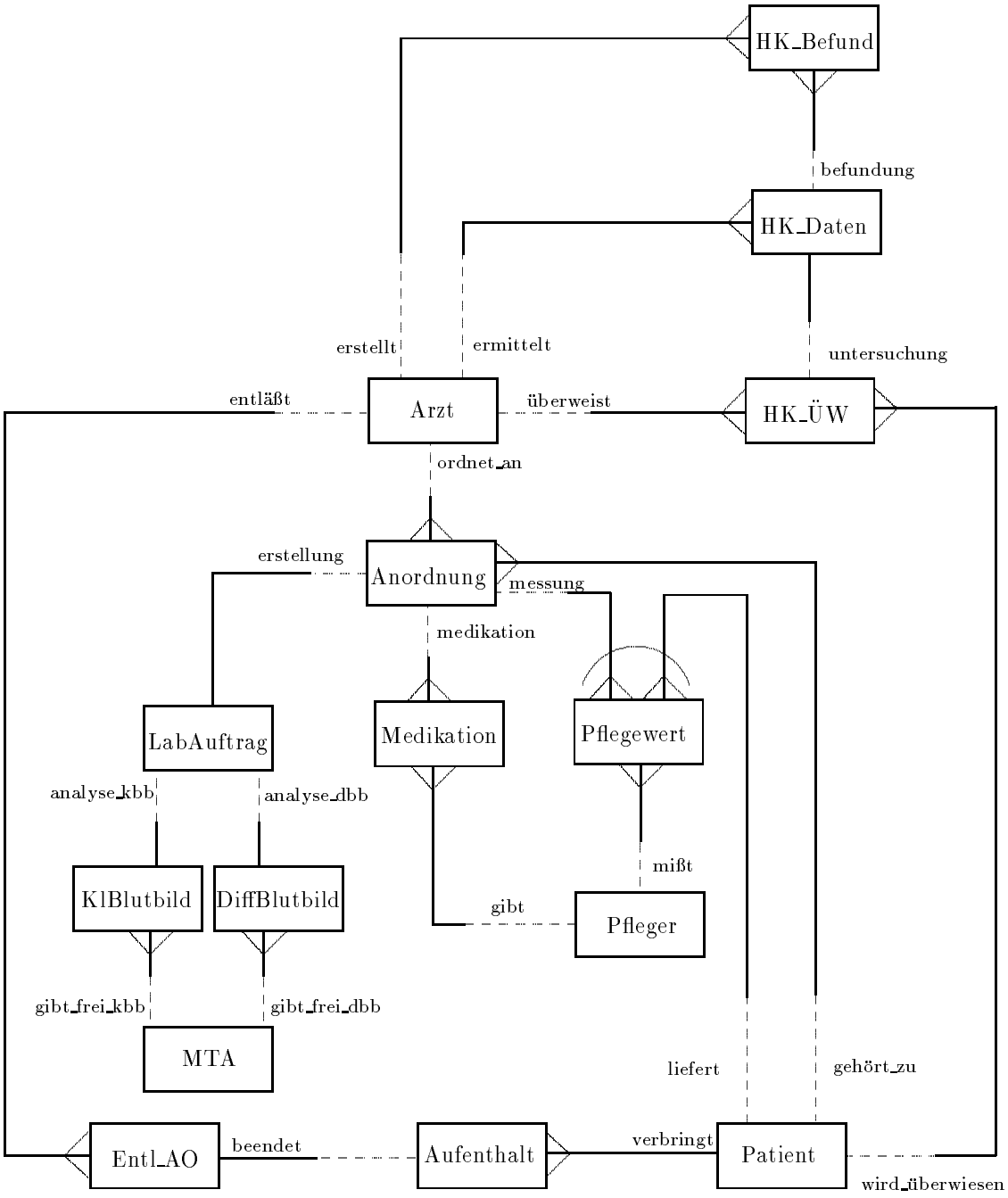

Abbildung 2: Gesamt-ER-Modell

2.1 Attributstruktur der Entities

Entity Patient

PatId:	PatId	(*mandatory; primary key*)
Name:	Name	(*mandatory*)
Geschlecht:	Geschlecht	(*mandatory*)
GebDatum:	DateTime	(*mandatory*)
GebOrt:	Ort	(*mandatory*)
Adresse:	Adresse	
Hausarzt:	Hausarzt	
Körperdaten:	KörperDaten	
Station:	Station	
Zimmer:	Zimmer	

Entity Aufenthalt

AufhFolgeNr:	Nat	(*mandatory; primary key*)
PatId:	PatId	(*mandatory; primary key*)
Kostenträger:	Kostenträger	(*mandatory*)
AufnahmeDatum:	DateTime	

Entity Arzt

ArztId:	ArztId	(*mandatory; primary key*)
Name:	Name	(*mandatory*)
Adresse:	Adresse	(*mandatory*)
Dienstbez:	Dienstbez	(*mandatory*)
Station:	Station	
Eintritt:	DateTime	(*mandatory*)
Austritt:	DateTime	
Rolle:	Rolle	(*mandatory*)

Entity Pfleger

PflegerId:	PflegerId	(*mandatory; primary key*)
Name:	Name	(*mandatory*)
Adresse:	Adresse	(*mandatory*)
Dienstbez:	Dienstbez	(*mandatory*)
Station:	Station	
Eintritt:	DateTime	(*mandatory*)
Austritt:	DateTime	
Rolle:	Rolle	(*mandatory*)

Entity Anordnung

PatId:	PatId	(*mandatory; primary key*)
AONr:	AOId	(*mandatory; primary key*)
Erstellung:	DateTime	(*mandatory*)
Komment:	Text	−−*Kommentare z.B. Zeitpunkt der Messung*
Bb_erwünscht:	Bool	−−*normales Blutbild*
DiffBb_erw:	Bool	−−*differenziertes Blutbild*
Therapie:	Medizin	−−*Therapiebogen*
VWerte:	Messungen	−−*angeordnete Vitalwerte*

Entity HK_ÜW (HK-Überweisung)

HKAOId:	HkId	(*mandatory; primary key*)
Erstellung:	DateTime	(*mandatory*)
Komment:	Text	

Entity Entl_AO (Entlassungsanordnung)

AufhFolgeNr:	Nat	(*mandatory; primary key*)
PatId:	PatId	(*mandatory; primary key*)
EntlassungsDatum:	DateTime	(*mandatory*)
EntlassungsDiagnose:	Diagnose	(*mandatory*)
EntlassungsZiel:	EntlZiel	(*mandatory*)

Entity Pflegewert (Vitalwerte, Ein- und Ausfuhr)

WertId:	PfId	(*mandatory; primary key*)
Datum:	DateTime	(*mandatory*)
Wert:	Mess_Wert	(*mandatory*)

Entity Medikation (gegebene Medikamente)

MedId:	MedId	(*mandatory; primary key*)
Datum:	DateTime	(*mandatory*)
Dosis:	Werte	(*mandatory*)

Entity LabAuftrag (Laborauftrag)

ANr:	AuftragNr	(*mandatory; primary key*)
ErstellungsZeitpkt:	DateTime	(*mandatory*)
Differw:	Bool	(*mandatory*)
Laboreingang:	DateTime	−−*wird beim Eingang ins Labor belegt*

Entity MTA

MTAId:	MTAId	(*mandatory; primary key*)
Name :	Name	(*mandatory*)
Adresse:	Adresse	(*mandatory*)
Einstellung:	DateTime	(*mandatory*)
Ausstellung:	DateTime	

Entity KlBlutbild

ANr:	AuftragNr	(*mandatory; primary key*)
Status:	Status	(*mandatory*)
Wert:	KlBlutbildWert	*−−kann UNDEF sein,*
		−−falls Status = nichtverwertbar
MTAId:	MTAId	(*mandatory*) *−−Kennung der MTA,*
		−−die den Wert freigibt
Zeitpunkt:	DateTime	*−−Zeitpunkt des Eintrags in die Datenbank*

Entity DiffBlutbild

ANr:	AuftragNr	(*mandatory; primary key*)
Status:	Status	(*mandatory*)
Wert:	DiffBlutbildWert	*−−kann UNDEF sein,*
		−−falls Status = nichtverwertbar
MTAId:	MTAId	(*mandatory*) *−−Kennung der MTA,*
		−−die den Wert freigibt
Zeitpunkt:	DateTime	*−−Zeitpunkt des Eintrags in die Datenbank*

Entity HK_Daten

HKAOId:	HkId	(*mandatory; primary key*)
HKP:	Text	
Anfangszeit:	DateTime	(*mandatory*)
Endezeit:	DateTime	
Druckkurven:	Druckkurven	
Röntgenfilm:	Röntgenfilm	

Entity HK_Befund

BefundId:	BefundId	(*mandatory; primary key*)
Befundungsdatum:	DateTime	(*mandatory*)
Befund:	Befund	(*mandatory*)
Befundbrief:	Befundbrief	

2.2 Externe Dokumente

Vertrag

PatId:	PatId	
Name:	Name	
Geschlecht:	Geschlecht	
Vertragsdatum:	DateTime	
Ort:	Ort	
Kostenträger:	Kostenträger	
TextVariante:	TextVariante	

AufnahmeNachricht

PatId:	PatId	
Name:	Name	
AufnahmeDatum:	DateTime	

Blutprobe

ANr:	AuftragNr	(*mandatory*) --*Etikett für das Labor*
Differw:	Bool	(*mandatory*) --*Etikett für das Labor*
Name:	Name	--*Etikett zum Blutabnehmen*
Zimmer:	Zimmer	--*Etikett zum Blutabnehmen*
Blut:	Blut	

EntlassungsNachricht

PatId:	PatId	
Name:	Name	
EntlassungsDatum:	DateTime	
EntlassungsZiel:	EntlZiel	

3 Aufnahme

3.1 Allgemeine Bemerkungen

Folgende Elemente aus der Ist-Ablaufbeschreibung werden aus folgenden Gründen in die Essenz nicht übernommen:

- Wegen nicht essentieller Funktionalität:
  Etiketten, Rena-Folien, Aufnahmekarte.
- Da außerhalb der Systemgrenzen:
  Vertrag durchlesen, unterschreiben;
  Einweisungsschein, Kostenübernahmeschein, Aufnahme- und Verhandlungsumschlag, Krankengeschichte-Formular.
- Durch Rationalisierungseffekte (auch anderer Bereiche) unnötig:
  Stationskarte, Entlassungsschein, Aufnahmekarten-Kartei.

Dagegen wurde ein zentraler Bereich in den Ist-Abläufen implizit gelassen und muß hinzugefügt werden:

- Patienten-Identifikation

3.2 Anmerkungen zu den einzelnen Aktivitäten

Identifikation

In den bisherigen Abläufen wird zunächst der Vertrag ausgestellt. Da aber in der neuen Fassung die Patienten-Identifikation eine zentrale Rolle spielt (z.B. wird der Vertrag der Kostenstelle wohl getrennt von den restlichen (elektronisch übermittelten) Daten zugeschickt werden und muß deshalb zugeordnet werden), ist als erster Schritt die Festlegung einer neuen oder bereits bekannten Patienten-Id nötig.

Es ist dieser Teil der Aufnahme-Prozedur, der darüber entscheidet, ob ein Patient als "neu" behandelt oder mit einer bereits vorhandenen Krankengeschichte identifiziert wird. Es wird vorgeschlagen, eine Identifikation nur zuzulassen, wenn die bei der Aufnahme ermittelten Attribute mit den gespeicherten Attributen in folgenden Werten übereinstimmen: Name (einschließlich Vorname), Geschlecht, Geburtsdatum, Geburtsort. Es wäre aber durchaus denkbar, noch weitere Kriterien hinzuzufügen, z.B. eine als Bitmap gespeicherte Unterschriftsprobe. Um solche Änderungen zu erleichtern, wurde das Identifikations-Kriterium in einer eigenen Funktion ("identisch") gekapselt.

Vertrags-Schreibung

Da kein Mustervertrag vorliegt, wird hier angenommen, daß man für das Schreiben des Vertrags folgende Daten braucht: Name, Geschlecht, Geburtsdatum, Geburtsort, Kostenträger. Des weiteren wird eine nicht näher spezifizierte Angabe zur Beschreibung eventueller Textvarianten angenommen.

Da der Behandlungsvertrag eine sehr wichtige Rolle für die Zulässigkeit aller weiteren Aktivitäten hat, wird sorgfältig darauf geachtet, nur Patienten aufzunehmen, die den Vertrag auch wirklich unterschrieben haben. Zu diesem Zweck werden drei elementare Transaktionen eingeführt: Start ist die Anmeldung, die immer entweder von einer Aufnahme (nach unterschriebenem Vertrag) oder einer Nicht-Aufnahme, d.h. Anmeldungs-Rücknahme gefolgt wird. Wollte man diesen Aspekt weniger betonen, könnte man Identifikation und Aufnahme natürlich auch zusammenfassen und den möglichen Abbruch auf Implementierungsebene (Transaktionskonzept) regeln.

Stammdaten-Vervollständigung

Die sogenannte Stammdaten-Befragung aus der Ist-Beschreibung hat in der Essenz nur noch eine sehr bescheidene Rolle der Vervollständigung der Stammdaten (um Anschrift und Hausarzt, evtl. weitere Einträge könnten noch dazukommen). Von Bedeutung ist diese Funktion als "Bestätigung" vor der Archivierung des Patienten; es wird ein Vermerk in der Datenbank aufbewahrt (Entity Aufenthalt), der den Vertragsabschluß dokumentiert und das Datum des Vertragsabschlusses enthält.

Die Körperdaten werden erst auf der Station (zusammen mit der Krankengeschichte, die aber nicht in die Datenbank kommt) erfaßt. Bei dieser Gelegenheit wird auch die Information über Station und Zimmer festgelegt, die in der Ist-Analyse noch nicht vorkommt.

Archivierung

Die Aufnahmekarten-Kartei wird ersetzt durch die Eintragung der Aufnahme-/ Entlassungsgeschichte in das zentrale Patientenarchiv (Entities Aufenthalt und Entl_AO). In der Soll-Spezifikation kann es aus Gründen der Zuverlässigkeit und Performance interessant sein, wieder einen lokalen, für die Identifikation und Vertragsabwicklung ausreichenden lokalen Speicher in der Aufnahme zu haben.

Für die Aufenthalts-Folgenummern wird ein Standard-Mechanismus zur Erzeugung neuer Schlüssel (genkey) benutzt. Wesentlich hierbei ist, daß die Funktion, die den neuen Schlüssel erzeugt, keine spezifische Information über die Daten des Patienten zur Verfügung hat. Damit sind Implementierungen ausgeschlossen, die über bestimmte Schlüssel-Codierungen erlauben, vertrauliche Patientendaten aus dem Schlüssel wiederzugewinnen.

Benachrichtigung der Kostenstelle

Nach Auskunft der TU Berlin kann man davon ausgehen, daß die notwendigen Daten an die Kostenstelle elektronisch übermittelt werden. Die Erstellung des Entlassungsscheins und ähnliche Aktivitäten wurde deshalb nicht berücksichtigt. Für eine Übergangsphase kann man sich auch vorstellen, daß entsprechende Komponenten an dieser Schnittstelle für die Erstellung der bisherigen Dokumente sorgen.

Zusammenhang zum Datenmodell der Ist-Analyse

Die Spezifikationen STAMMDATEN, PAT_ID und KÖRPER_DATEN wurden in angepaßter Form ins Datenmodell, d.h. in die oben angeführten Entity-Beschreibungen, übernommen. Allerdings wurden folgende Umbezeichnungen vorgenommen:

- Id zu PatId,
- Patid zu Patient.

In die Stammdaten wurde folgendes neue Feld aufgenommen:

- Geburtsort (für eine bessere Identifizierung)

Es gibt keinen Grund mehr, die Stammdaten zu einem eigenen strukturierten Attribut zu machen, deshalb wurden Stammdaten und Patientendaten fusioniert. Für die Körperdaten wird genau die Sorte und die Spezifikation aus der Ist-Analyse übernommen. Die Aufenthaltsgeschichte wurde zu einer eigenen Entität Aufenthalt (entstanden aus der alten Aufnahmekarten-Kartei).

3.3 Schnittstelle zum Datenmodell

Relevanter Ausschnitt aus dem E/R-Modell:

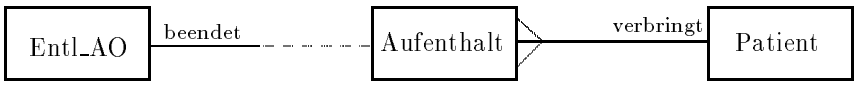

Abbildung 3: Teildatenmodell Aufnahme

Im Diagramm kaum darstellbare, aber dennoch wichtige Integritätsbedingungen sind folgende Aussagen:

- Für jeden Patienten gibt es maximal einen Aufenthalt, der noch nicht (durch eine EntlassungsAO) beendet ist.
- Es darf keine zwei Patientendatensätze mit verschiedenen Identifikatoren geben, die nach dem Identifikationskriterium (Name, Geschlecht, Geburtsdatum, Geburtsort) gleich sein müßten.

Beide Integritätsbedingungen werden durch folgendes Pädikat formalisiert:

```
C: Db → Bool;
C strict total;

axioms ∀ db in
  C db = ( ∀ ah1,ah2,p.
            verbringt db (p,ah1) ∧ ¬(∃eao. beendet db (eao,ah1)) ∧
            verbringt db (p,ah2) ∧ ¬(∃eao. beendet db (eao,ah2)) ⇒ ah1=ah2 )
         ∧ (∀ pi,pi',n,g,d,o.
            identisch(pi,u,g,d,o,db)∧identisch(pi',u,g,d,o,db) ⇒ pi=pi' );
          -- Für die Hilfsfunktion identisch siehe Abschnitt 3.5
endaxioms;
```

3.4 Datenflußdiagramm

Das Datenflußdiagramm zeigt die Abläufe im System und die nötigen externen Abläufe. Der Patient wird in Abhängigkeit seiner Unterschrift unter den Vertrag aufgenommen oder nicht.

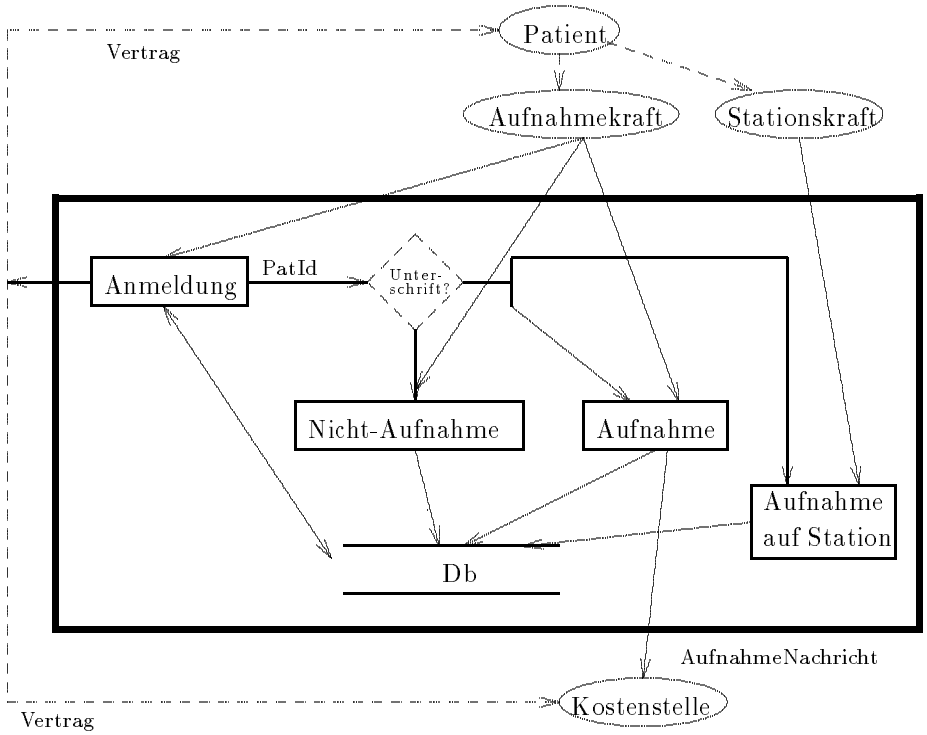

Abbildung 4: DFD Aufnahme

3.5 Schnittstelle zur Datenbank (Hilfsfunktionen)

```
DB_INT = {
enriches DB;

-- Aktuell ist ein Aufenthalt, wenn keine Entlassungsanordnung dazu vorliegt.
istAktuell: Db × Aufenthalt →  Bool;
istAktuell strict;
axioms ∀ db, ah in
    δ(istAktuell(db, ah)) = ah ∈ entAufenthalt(db);
    δ(istAktuell(db, ah)) ⇒
        istAktuell(db, ah) = ¬∃eao. beendet db (eao, ah);
endaxioms;

-- Der (eindeutige!) aktuelle Aufenthalt für einen Patienten.
aktuell: Db × PatId → Aufenthalt;
aktuell strict;
axioms ∀ db, pi, ah in
```

```
    δ(aktuell(db, pi)) = ∃p. p = getPatient(db, pi) ∧
                ∃ ah. verbringt db (p, ah) ∧ istAktuell(db, ah);
    ah = aktuell(db, pi) ⇒ verbringt db (getPatient(db, pi), ah)
                          ∧ istAktuell(db, ah);
endaxioms;

-- Identifikations-Kriterium
identisch: PatId × Name × Geschlecht × Datum × Ort × Db → Bool;
identisch strict;
axioms ∀ db, pi, n, g, d, o in
    δ(identisch(pi, n, g, d, o, db)) = δ(getPatient(pi, db));
    δ(identisch(pi, n, g, d, o, db)) ⇒
    identisch(pi, n, g, d, o, db) =
      let p = getPatient(pi, db) in
        name(p) = attr n ∧ geschlecht(p) = attr g ∧
        gebDatum(p) = attr n ∧ gebOrt(p) = attr o;
endaxioms;
}
```

3.6 Formale Spezifikation der elementaren Transaktionen

```
AUFNAHME_AKTIONEN = {
enriches DB+DB_INT+PATIENT+WERTE+AUFENTHALT;
strict;

identifikation: Name × Geschlecht × Datum × Ort × Db → Db × PatId;
-- Ermitteln der eindeutigen (neuen oder alten) PatId
identifikation total;
axioms ∀ db, pi, n, g, d, o, p in
    δ(getPatient(pi, db)) ∧ identisch(pi, n, g, d, o, db) ⇒
          ∃pi'.identifikation(n, g, d, o, db) = (db, pi')
          ∧ identisch(pi',n,g,d,o,db);
    ¬∃pi. δ(getPatient(pi, db)) ∧ identisch(pi, n, g, d, o, db) ⇒
      identifikation(n, g, d, o, db) = (db', newpi)
        where
          newpi = genkeyPatient(db, (λ pi'. true)) and
          db'   = putPatient(
                    createPatient(attr newpi,
                    attr n, attr g, attr d, attr o,
                    UNDEF, UNDEF, UNDEF, UNDEF, UNDEF), db));
endaxioms;

anmeldung: Name × Geschlecht × Datum × Ort × Kostenträger × Textvariante × Db
           → Db ×  PatId × Vertrag;
axioms ∀ db, pi, n, g, d, o, kt, tv in
   δ(anmeldung(n, g, d, o, kt, tv, db)) =
     ¬(∃pi. identisch(pi, n, g, d, o, db) ∧ ∃ah. ah = aktuell(db, pi));
   δ(anmeldung(n, g, d, o, kt, tv, db)) ⇒
   anmeldung(n, g, d, o, kt, tv, db) = (db', pi, v)
     where
       (db1, pi) = identifikation(n, g, d, o, db) and
       p         = getPatient(pi, db1) and
```

```
       (pi',anr) = genkeyAufenthalt(db, (λ (anr', pi'). pi' = pi)) and
       ah        = createAufenthalt(anr, pi, attr kt, UNDEF) and
       db'       = estverbringt(putAufenthalt(ah, db1), p, ah) and
       v         = createVertrag(attr pi, attr n, attr g, attr d, attr o,
                     attr kt, attr tv);
endaxioms;

aufnahme: PatId × Adresse ×  Hausarzt × Db → Db × AufnahmeNachricht;
-- Patient hat Vertrag unterschrieben.
nicht_aufnahme: PatId × Db → Db;
-- Patient hat Vertrag nicht unterschrieben.
axioms ∀ db, pi, a, ha, ah in
    δ(aufnahme(pi, a, ha, db)) =
        ∃ ah. aktuell(db, pi) = ah ∧ AufnahmeDatum(ah) = UNDEF;
-- Die Bedingung besagt, dass der Patient zur Aufnahme ansteht,
-- aber noch nicht aufgenommen ist.
    δ(aufnahme(pi, a, ha, db)) ⇒
    aufnahme (pi, a, ha, db) = (db', an)
      where
        p   = getPatient(pi, db)  and
        p'  = setHausarzt(setAdresse(p, attr a), attr ha)) and
        db1 = updatePatient(pi, p', db) and
        db' = putAufenthalt(
                setAufnahmeDatum(aktuell(db, pi), datetime(db)), db1) and
        an  = createAufnahmeNachricht(pi, name(p), datetime(db));

    δ(nicht_aufnahme(pi, a, ha, db)) =
        ∃ ah. aktuell(db, pi) = ah ∧ AufnahmeDatum(ah) = UNDEF;
    δ(nicht_aufnahme(pi, a, ha, db)) ⇒
    nicht_aufnahme (pi, db) = db'
      where
        ah  = aktuell(db, pi) and
        p   = getPatient(pi, db) and
        db1 = delAufenthalt(ah, relverbringt(db, p, ah)) and
        db' = if ¬ ∃ah'. verbringt db (p, ah')
              then delPatient(p, db1) else db1 endif;
-- Wenn der Patient nicht schon früher in der Klinik war,
-- wird sein Datensatz wieder gelöscht.
endaxioms;

aufnahmeAufStation: Db × PatId × Grösse × Gewicht × Station × Zimmer → Db;
aufnahmeAufStation strict;
axioms ∀ db, pi, gr, gew, st, zi in
    δ(aufnahmeAufStation(db, pi, gr, gew, st, zi)) = (∃ ah. ah = aktuell(db, pi));
-- Das ist der Zustand eines Patienten–Datensatzes nach der Anmeldung.

    aufnahmeAufStation(db, pi, gr, gew, st, zi)
      = updatePatient(pi,setStation(setZimmer(
          setKörperdaten(getPatient(pi, db), attr körperdaten(gr, gew)),
            attr zi), attr st), db);
endaxioms;
}
```

4 Arzt- und Behandlungs-Abläufe

In diesem Kapitel beschreiben wir die Arbeitsabläufe des Arztes und des Pflegers während der Patientenbehandlung. Sie ergeben sich aus dem Behandlungsablauf der IST-Analyse durch ihre logische Struktur. Weitere Unterschiede zur IST-Analyse sind:

- Anordnungs- und Therapiebogen werden mit den Laboranordnungen zu einer Sorte Anordnung zusammengefaßt
- Die Fieberkurve ist eine Sicht auf mehrere Entities der Datenbank
- Die Tätigkeiten des Übertragens in die Fieberkurve entfallen
- Der NAB entfällt

4.1 ER-Diagramm für den Behandlungs-Ablauf

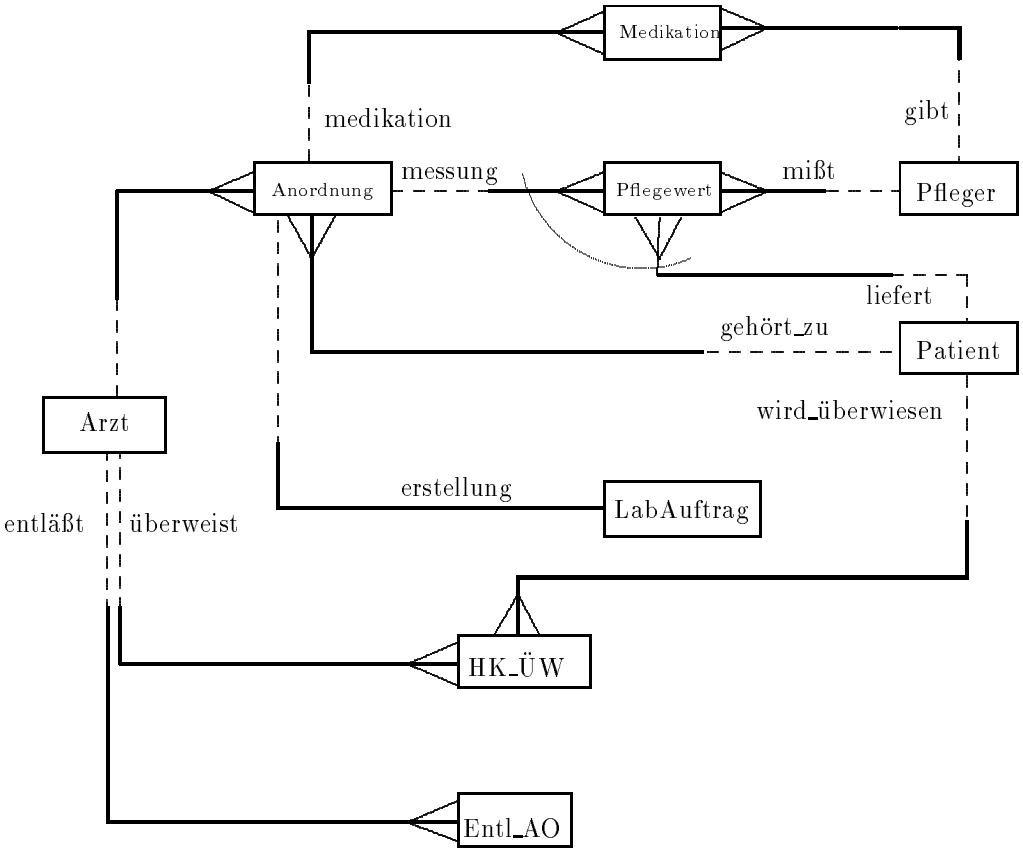

Abbildung 5: Teildatenmodell für Arzt- und Behandlungs-Abläufe

Eine Integritätsbedingung des Teildatenmodells ist: Es darf keine Anordnung ohne Inhalt geben. Dies wird durch folgendes Prädikat gesichert:

```
C_Anordnung : Db → Bool;
C_Anordnung strict total;

axioms ∀ DB : Db in
        C_Anordnung DB = ∀ An : Anordnung . An ∈ entAnordnung( DB ) ⇒
        ( Bb_erwünscht( An ) = attr true          --Blutbild angeordnet
        ∨ Therapie( An ) ≠ attr []                --Medikamente angeordnet
        ∨ VWerte ( An ) ≠ attr [] );              --Pflegewerte angefordert
endaxioms;
```

Eine weitere Bedingung ist die Übereinstimmung des Fremdschlüssels PatId aus der Anordnung mit der PatId des Patienten, zu dem diese Anordnung gehört. Dies wird durch folgendes Prädikat gesichert:

```
C_gehört_zu : Db → Bool;
C_gehört_zu strict total;

axioms ∀ DB : Db in
        C_gehört_zu DB = ∀ An : Anordnung, Pat : Patient .
                An ∈ entAnordnung( DB ) ∧
                Pat ∈ entPatient( DB ) ∧
                gehört_zu DB( An, Pat ) ⇔ PatId( An ) = PatId( Pat );
endaxioms;
```

4.2 Datenflußdiagramm für den Arzt-Ablauf

Das Datenflußdiagramm beschreibt die dem Arzt bei der Visite gebotene Systemunterstützung. Dabei stehen dem Arzt die elementaren Transaktionen 'Entlassen', 'Uberweisen' und 'Behandeln' zur Verfügung. Die Systemgrenze bilden dabei die zu den Patienten gehörigen medizinischen Daten und Pflegeanweisungen, die der Arzt bei der Ausführung dieser Transaktionen eingeben muß.

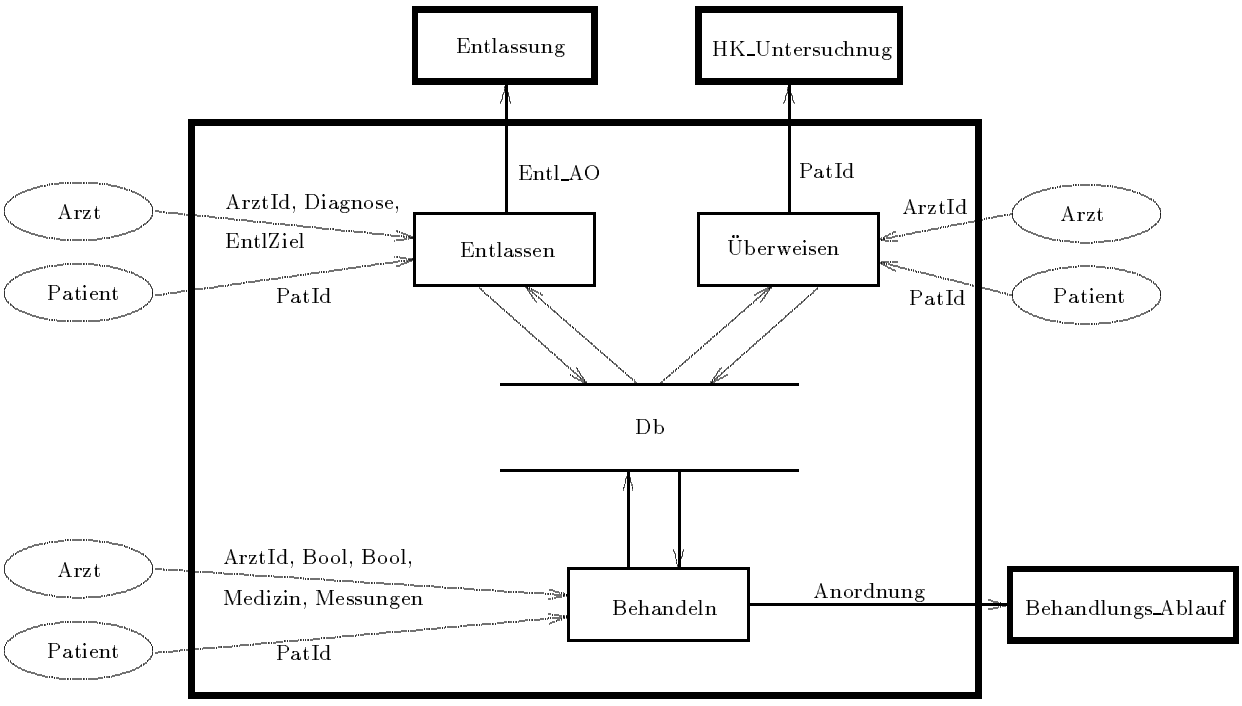

Abbildung 6: DFD Arzt-Ablauf

4.3 Datenflußdiagramm für den allgemeinen Behandlungs-Ablauf

Der Behandlungsablauf gliedert sich in drei Teile. Das Datenflußdiagramm des allgemeinen Behandlungs-Ablaufs beschreibt das Ausführen der angeordneten medizinischen Behandlung durch den Pfleger und stellt den Zusammenhang zum Labor-Ablauf dar. Die Messung der Vitalwerte erfolgt in zwei eigenen Abläufen.

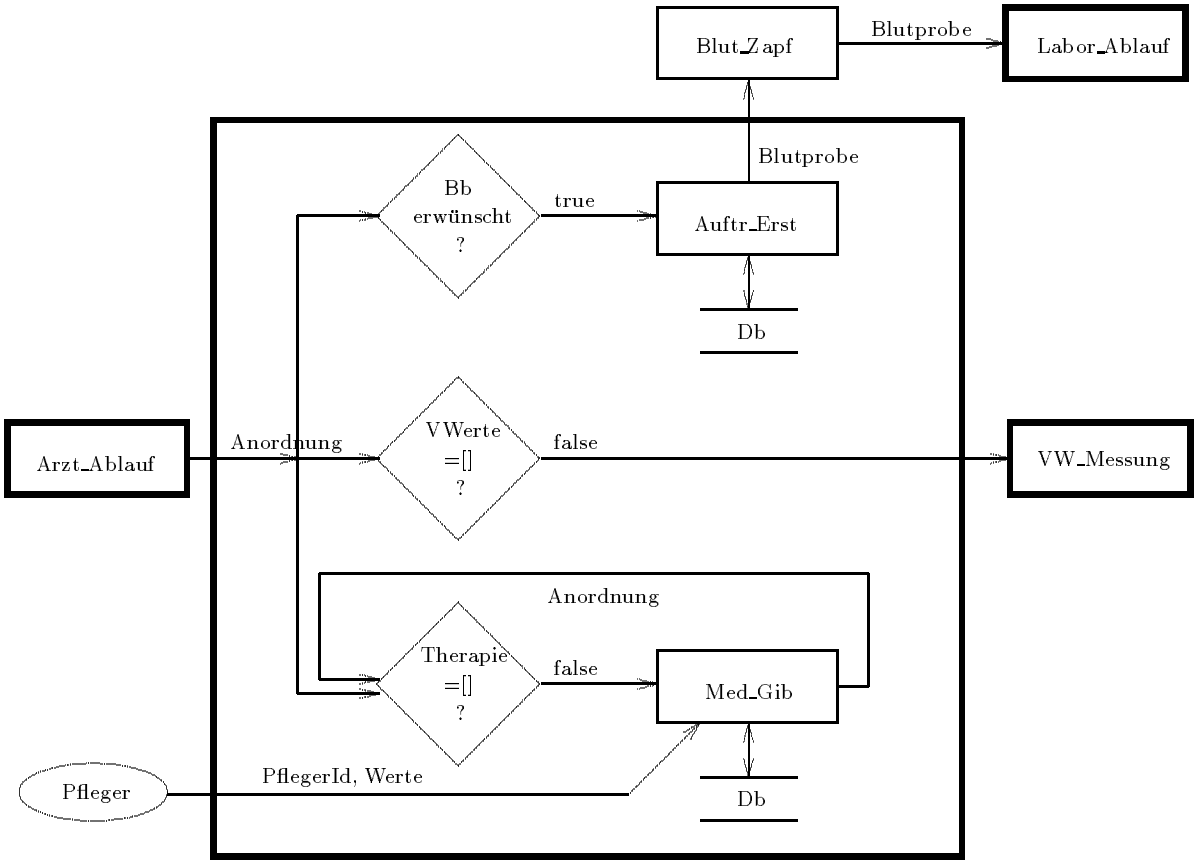

Abbildung 7: DFD angeordnete und Routine-Behandlung

4.4 Datenflußdiagramm für Vitalwertmessung auf Anordnung

Dieses Datenflußdiagramm zur Vitalwertmessung beschreibt die unterschiedlichen Messungsarten, die auf Anordnung des Arztes vorgenommen werden. Die gestrichelten Funktionen stellen dabei Umweltfunktionen dar.

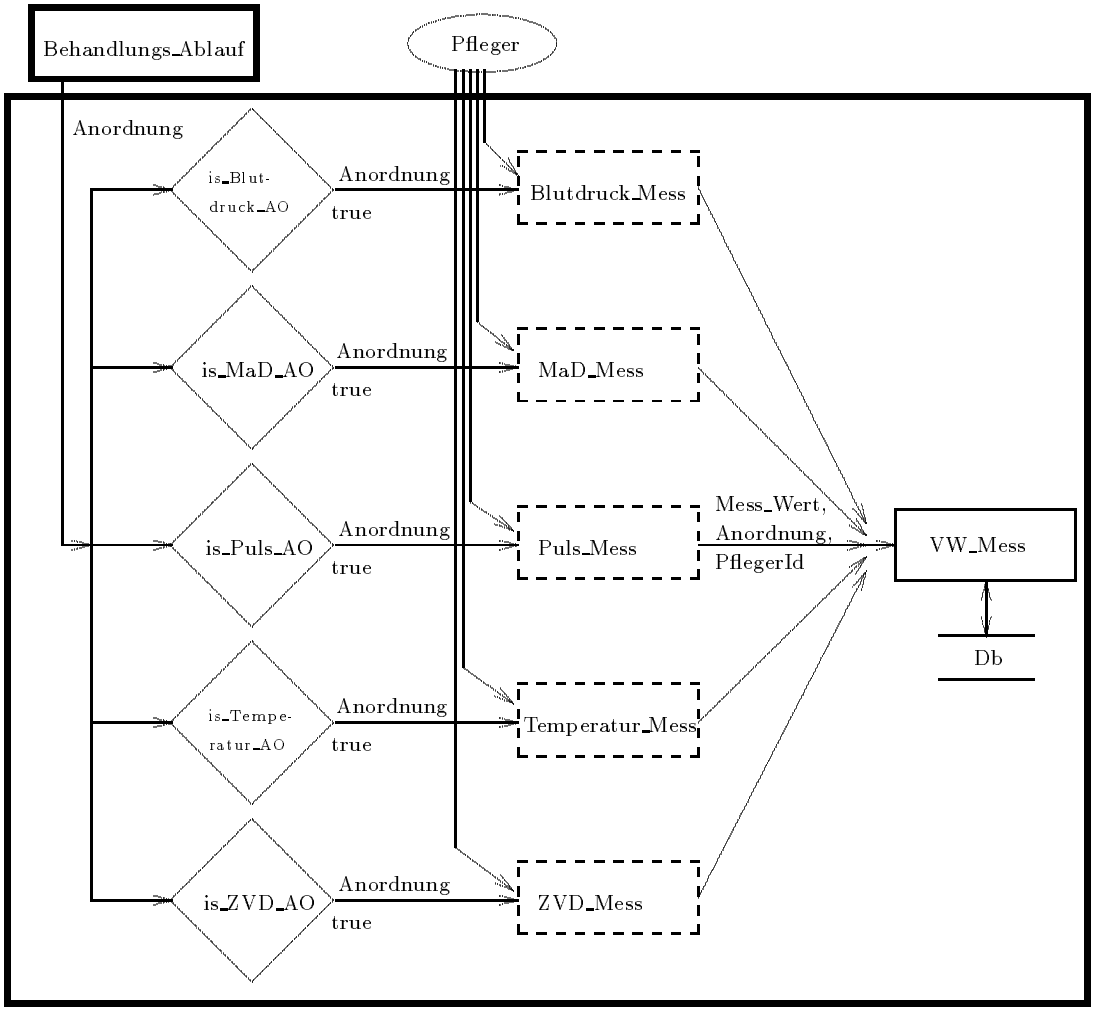

Abbildung 8: DFD Vitalwert-Messung auf Anordnung

4.5 Datenflußdiagramm für Vitalwertmessung aus Routine

Dieses Datenflußdiagramm der Vitalwertmessung beschreibt die Messungen, die der Pfleger aufgrund seiner Erfahrung ohne Anordnung durchführen darf.

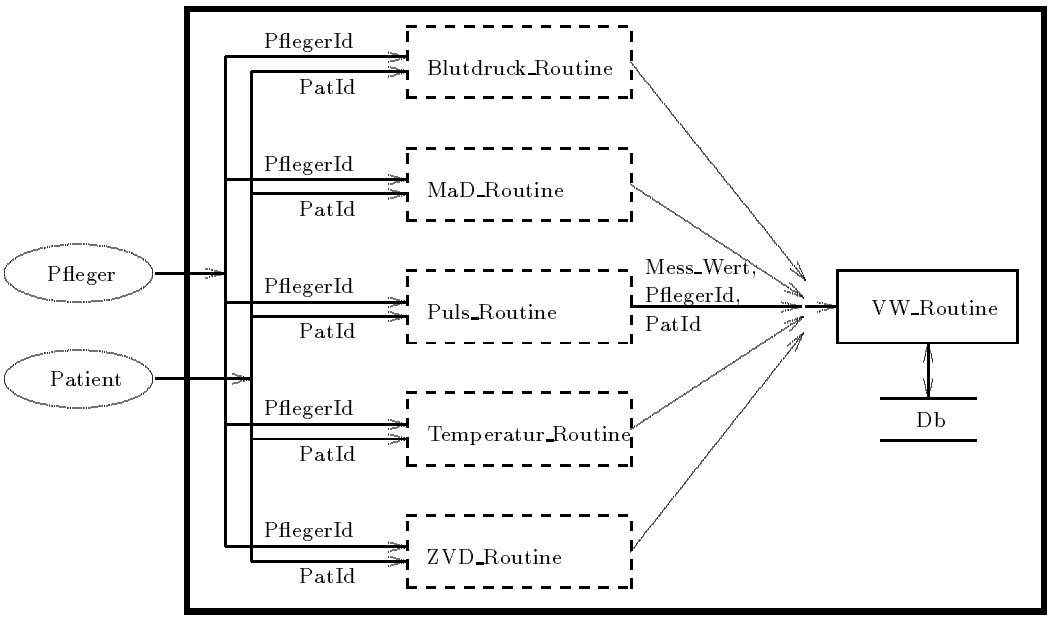

Abbildung 9: DFD Vitalwert-Messung aus Routine

4.6 Formale Spezifikation von Attributsorten und Hilfsprädikaten

Einige Sorten der Attribute des ER-Diagramms sind nicht elementar und werden jetzt spezifiziert:

```
SORTEN = {
        enriches WERTE;

        --Medikationen
        data Med =
                med(    ! Id : MedId,                    --Identifikator für ein Medikament
                        ! Dosis : Werte,
                        ! Beginn_med : DateTime,
                        ! Ende_med : DateTime );


        sortsyn Medizin = List Med

        axioms ∀ M : Med, R,S : Medizin in
                R = cons( M, S ) ⇒ ¬ ( M isin S );
```

```
        endaxioms;
        --damit sind die Medikationen eine endliche Menge

        --Verschiedene Messungsarten für Pflegewerte
        data Messung = Blutdruck
                |       MaD             --Mittlerer arterieller Druck
                |       Puls
                |       Temperatur
                |       ZVD;            --Zentraler Venen Druck

        sortsyn Messungen = List Messung

        axioms ∀ M : Messung, R,S : Messungen in
                R = cons( M, S ) ⇒ ¬ ( M isin S );
        endaxioms;
        --damit sind die Messungen eine endliche Menge

        --Verschiedene Pflegewerte
        data Mess_Wert = mk_Blutdruck( ! syst : Werte, ! dias : Werte )
                |       mk_MaD( ! MaD : Werte)
                |       mk_Puls( ! Puls : Werte)
                |       mk_Temperatur( ! Temperatur : Werte)
                |       mk_ZVD( ! ZVD : Werte);


        --Dieses Prädikat prüft, ob die Sorten der Befehle (Messung) mit denen der gemessenen
        --Ergebnisse (Mess_Wert) übereinstimmen

        paßt_zu : Messung × Mess_Wert ⟶ Bool;
        paßt_zu strict total;

        axioms ∀ Mess : Messung, Wert : Mess_Wert in
                paßt_zu( Mess, Wert ) =
                        ( is_Blutdruck( Mess ) ∧ is_mk_blutdruck( Wert ) ) ∨
                        ( is_MaD( Mess ) ∧ is_mk_MaD( Wert ) ) ∨
                        ( is_Puls( Mess ) ∧ is_mk_Puls( Wert ) ) ∨
                        ( is_Temperatur( Mess ) ∧ is_mk_Temperatur( Wert ) ) ∨
                        ( is_ZVD( Mess ) ∧ is_mk_ZVD( Wert ) );
        endaxioms;
}
```

Formale Spezifikation von Hilfsprädikaten Das Prädikat `erledigt` gibt an, ob eine Anordnung hinreichend bearbeitet ist. Da eine Anordnung eine Liste von zeitlich abgestuften Medikationen enthalten kann (z.B. morgens und abends etwas verabreichen), ist das Prädikat von der Zeit abhängig.

```
ANORDNUG = {
        enriches SORTEN + DB;
        strict;
        erledigt : Anordnung × Db ⟶ Bool;

        axioms ∀ Anord : Anordnung, DB : Db in
```

```
                δ( erledigt( Anord, DB ) ) = Anord ∈ entAnordnung( DB );
                δ( erledigt( Anord, DB ) ) ⇒  erledigt( Anord, DB ) =
                        ( Bb_erwünscht( Anord ) = attr true ⇒
                                ∃ LabAuf : LabAuftrag . erstellung DB ( Anord, LabAuf ) ) ∧
                        ( Therapie( Anord ) ≠ attr [ ] ⇒
                                ∀ Med . Med isin ↓Therapie( Anord ) ⇒
                                        ( Ende_med( Med ) before datetime( DB ) ∨
                                        datetime( DB ) before Beginn_med( Med ) ∨
                                        ( ∀ Medik . Medik ∈ entMedikation( DB ) ∧
                                        medikation DB ( Anord, Medik ) ∧
                                        Id( Med ) = ↓MedId( Medik ) ∧
                                        Dosis( Med ) = ↓Dosis( Medik ) ⇒
                                                ¬ is_today( ↓Datum( Medik ), DB ) ) ) ) ∧
                        ( VWerte( Anord ) ≠ attr [ ] ⇒
                                ∀ PfB . PfB isin ↓VWerte( Anord ) .
                                        ∃ PfW . PfW ∈ entPflegewert( DB ) ∧
                                        messung DB ( Anord, PfW ) ∧
                                        paßt_zu( PfB, ↓Werte( PfW ) );
                        --eine Anordnung ist erledigt, wenn die Laboraufträge geschrieben
                        --sind und die Medikationen gegeben sind (Dauer beachten) und
                        --wenn die befohlenen Werte gemessen sind

        endaxioms;
}
```

Die Hilfsfunktion `behandelbar` stellt sicher, daß der Patient vom Arzt behandelt werden darf. Die Funktion `pflegbar` prüft die Berechtigung des Pflegers, den Patienten zu pflegen. HK_aktive gibt an, ob eine HK-Untersuchung noch nicht beendet ist. Dann ist eine Entlassung oder eine erneute HK-Überweisung unmöglich.

```
HILF = {
        enriches ANORDNUG + AUFNAHME + DB_INT;
        strict;
        total;

        behandelbar : Db × PatId × ArztId → Bool;
        pflegbar : Db × PatId × PflegerId → Bool;
        HK_aktive : Db × PatId → Bool;

        axioms ∀ DB : Db, PatId : PatId, AId : ArztId in
                behandelbar( DB, PatId, AId ) =
                        δ( getPatient( PatId, DB ) ) ∧
                        δ( getArzt( AId, DB ) )∧
                        δ( aktuell( DB, PatId ) );
                --aktuell s. Entlassungsablauf
        endaxioms;
        axioms ∀ DB : Db, PatId : PatId, PfId : PflegerId in
                pflegbar( DB, PatId, PfId ) =
                        δ( getPatient( PatId, DB ) ) ∧
                        δ( getPfleger( PfId, DB ) ) ∧
                        δ( aktuell( DB, PatId ) );
        endaxioms;
```

```
        axioms ∀ DB : Db, PatId : PatId in
                HK_aktive( DB, PatId) =
                        δ( getPatient( PatId, DB ) ) ∧
                        δ( aktuell( DB, PatId ) ) ∧
                        ∃ HKÜW . HKÜW ∈ entHK_ÜW( DB ) ∧
                        ¬∃ HK_D . HK_D ∈ entHK_Daten( DB ) ∧
                        untersuchung( DB )( HKÜW, HK_D );
        endaxioms;
}
```

4.7 Formale Spezifikation des Arztablaufes

Diese Spezifikation beschreibt die drei elementaren Transaktionen `entlassen`, `überweisen` und `behandeln`. Die Inputs dieser Funktionen gibt der Arzt z.B. bei seiner Visite in das System ein.

```
ARZT_ABL =
{ enriches AUFNAHME_AKTIONEN + HILF;
  strict;

  -- 1. Entlassen

  entlassen: Db × PatId × ArztId × Diagnose × EntlZiel → Db × PatId;

  axioms ∀ DB: Db, Pi: PatId, Ai: ArztId, Diag: Diagnose, Ziel: EntlZiel in
    δ(entlassen(DB, Pi, Ai, Diag, Ziel)) =
        behandelbar(DB, Pi, Ai) ∧
        ¬HK_aktive(DB, Pi);
    δ(entlassen(DB, Pi, Ai, Diag, Ziel)) ⇒
    entlassen(DB, Pi, Ai, Diag, Ziel) = (DB', Pi) where
      Aufent = aktuell(DB, Pi) and
      EntlAO = createEntlAO(AufhFolgeNr(Aufent), attr Pi,
               attr datetime(DB), attr Diag, attr Ziel) and
      DB1    = putEntl_AO(EntlAO, DB) and
      DB2    = estbeendet(DB1, EntlAO, Aufent) and
      DB'    = estentläßt(DB2, EntlAO, getArzt(Ai, DB));
  endaxioms;

  -- 2. Überweisen

  überweisen: Db × PatId × ArztId → Db × PatId;

  axioms ∀ DB: Db, Pi: PatId, Ai: ArztId in
    δ(überweisen(DB, Pi, Ai)) =
        behandelbar(DB, Pi, Ai) ∧
        ¬HK_aktive(DB, Pi);
    δ(überweisen(DB, Pi, Ai)) ⇒
    (überweisen(DB, Pi, Ai) = (DB', Pi) where
      HKÜW   = createHK_ÜW(attr genkeyHK_ÜW(DB, λ u. true),
               attr datetime(DB), UNDEF) and
      DB1    = putHK_ÜW(HKÜW, DB) and
      DB2    = estüberweist(DB1, HKÜW, getArzt(Ai, DB)) and
      DB'    = estwirdüberwiesen(DB2, HKÜW, getPatient(Pi, DB)) );
```

```
  endaxioms;

  -- 3. Behandeln

  behandeln: Db × PatId × ArztId × Bool × Bool ×
               Medizin × Messungen → Db × Anordnung;

  axioms ∀ DB: Db, Pi: PatId, Ai: ArztId, Bb_need, Diffbb_need: Bool,
           Mz: Medizin, Mess: Messungen in
    δ(behandeln(DB, Pi, Ai, Bb_need, Diffbb_need, Mz, Mess)) =
        behandelbar(DB, Pi, Ai);
    δ(behandeln(DB, Pi, Ai, Bb_need, Diffbb_need, Mz, Mess)) ⇒
    behandeln(DB, Pi, Ai, Bb_need, Diffbb_need, Mz, Mess) = (DB', Pi) where
      Anord = createAnordnung(attr Pi, attr λ(x,y).y(   -- 2. Komponente
                  genkeyAnordnung(DB, λ(p: PatId, n: Nat) . p == Pi) ),
              attr datetime(DB), UNDEF, attr Bb_need,
              attr diffbb_need, attr Mz, attr Mess) and
      DB1    = putAnordnung(Anord, DB) and
      DB2    = estordnet_an(DB1, Anord, getArzt(Ai, DB)) and
      DB'    = estgehört_zu(DB2, Anord, getPatient(Pi, DB)) );
  endaxioms;
}
```

4.8 Formale Spezifikationen der Behandlungsabläufe

Zuerst werden in PFLEGER_HILF einige Hilfsfunktionen zum Unterscheiden der Anordnungen nach ihrem Inhalt spezifiziert. In PFLEGER_MESS werden die Umweltfunktionen zum Erfassen der Meßwerte recht lose beschrieben. An ihrer schematischen Form erkennt man die leichte Erweiterbarkeit dieser Spezifikation auf andere, in einem Krankenhaus mögliche Messungen.

In PFLEGER_VW_ABL werden die Funktionen zum korrekten Speichern der gemessenen Daten spezifiziert. In PFLEGER_MED_ABL wird das Geben von Medikamenten beschrieben. Dies kann nur auf Anordnung geschehen. Es gibt allerdings Anordnungen, die die dauerhafte Gabe von Medikamenten befehlen.

```
PFLEGER_HILF = {          --enthält Hilfsprädikate zum Unterscheiden von Anordnungen
enriches ANORDNUNG;
strict;
total;

--Blutdruck-Anordnung
is_Blutdruck_AO : Anordnung → Bool;
axioms ∀ AO : Anordnung in
        is_Blutdruck_AO( AO ) =
        ∃ M : Messung . M isin ↓VWerte( AO ) ∧ is_Blutdruck( M );
endaxioms;

--MaD-Anordnung
is_MaD_AO : Anordnung → Bool;
axioms ∀ AO : Anordnung in
        is_MaD_AO( AO ) =
```

```
        ∃ M : Messung . M isin ↓VWerte( AO ) ∧ is_MaD( M );
endaxioms;

--Puls-Anordnung
is_Puls_AO : Anordnung → Bool;
axioms ∀ AO : Anordnung in
        is_Puls_AO( AO ) =
        ∃ M : Messung . M isin ↓VWerte( AO ) ∧ is_Puls( M );
endaxioms;

--Temperatur-Anordnung
is_Temperatur_AO : Anordnung → Bool;
axioms ∀ AO : Anordnung in
        is_Temperatur_AO( AO ) =
        ∃ M : Messung . M isin ↓VWerte( AO ) ∧ is_Temperatur( M );
endaxioms;

--ZVD-Anordnung
is_ZVD_AO : Anordnung → Bool;
axioms ∀ AO : Anordnung in
        is_ZVD_AO( AO ) =
        ∃ M : Messung . M isin ↓VWerte( AO ) ∧ is_ZVD( M );
endaxioms;

}

PFLEGER_MESS = {        --enthält die Umweltfunktionen des Behandlungsablaufs
enriches ANORDNUNG;
strict;
total;

--Blutdruck auf Anordnung:
Blutdruck_Mess : Anordnung × PflegerId → Mess_Wert × Anordnung × PflegerId;
axioms ∀ AO : Anordnung, PfI : PflegerId in
        ∃ MW : Mess_Wert .
                Blutdruck_Mess( AO, PfI ) = ( MW, AO, PfI) ∧ is_mk_Blutdruck( MW );
endaxioms;

--MaD auf Anordnung:
MaD_Mess : Anordnung × PflegerId → Mess_Wert × Anordnung × PflegerId;
axioms ∀ AO : Anordnung, PfI : PflegerId in
        ∃ MW : Mess_Wert .
                MaD_Mess( AO, PfI ) = ( MW, AO, PfI) ∧ is_mk_MaD( MW );
endaxioms;

--Puls auf Anordnung:
Puls_Mess : Anordnung × PflegerId → Mess_Wert × Anordnung × PflegerId;
axioms ∀ AO : Anordnung, PfI : PflegerId in
        ∃ MW : Mess_Wert .
                Puls_Mess( AO, PfI ) = ( MW, AO, PfI) ∧ is_mk_Puls( MW );
endaxioms;

--Temperatur auf Anordnung:
```

```
Temperatur_Mess : Anordnung × PflegerId → Mess_Wert × Anordnung × PflegerId;
axioms ∀ AO : Anordnung, PfI : PflegerId in
        ∃ MW : Mess_Wert .
                Temperatur_Mess( AO, PfI ) = ( MW, AO, PfI) ∧ is_mk_Temperatur( MW );
endaxioms;

--ZVD auf Anordnung:
ZVD_Mess : Anordnung × PflegerId → Mess_Wert × Anordnung × PflegerId;
axioms ∀ AO : Anordnung, PfI : PflegerId in
        ∃ MW : Mess_Wert .
                ZVD_Mess( AO, PfI ) = ( MW, AO, PfI) ∧ is_mk_ZVD( MW );
endaxioms;

--Blutdruck aus Routine:
Blutdruck_Routine : PflegerId × PatId → Mess_Wert × PflegerId × PatId;
axioms ∀ PfI : PflegerId, PatI : PatId in
        ∃ MW : Mess_Wert .
                Blutdruck_Routine( PfI, PatI ) = ( MW, PfI, PatI ) ∧ is_mk_Blutdruck( MW );
endaxioms;

--MaD aus Routine:
MaD_Routine : PflegerId × PatId → Mess_Wert × PflegerId × PatId;
axioms ∀ PfI : PflegerId, PatI : PatId in
        ∃ MW : Mess_Wert .
                MaD_Routine( PfI, PatI ) = ( MW, PfI, PatI ) ∧ is_mk_MaD( MW );
endaxioms;

--Puls aus Routine:
Puls_Routine : PflegerId × PatId → Mess_Wert × PflegerId × PatId;
axioms ∀ PfI : PflegerId, PatI : PatId in
        ∃ MW : Mess_Wert .
                Puls_Routine( PfI, PatI ) = ( MW, PfI, PatI ) ∧ is_mk_Puls( MW );
endaxioms;

--Temperatur aus Routine:
Temperatur_Routine : PflegerId × PatId → Mess_Wert × PflegerId × PatId;
axioms ∀ PfI : PflegerId, PatI : PatId in
        ∃ MW : Mess_Wert .
                Temperatur_Routine( PfI, PatI ) = ( MW, PfI, PatI ) ∧ is_mk_Temperatur(MW);
endaxioms;

--ZVD aus Routine:
ZVD_Routine : PflegerId × PatId → Mess_Wert × PflegerId × PatId;
axioms ∀ PfI : PflegerId, PatI : PatId in
        ∃ MW : Mess_Wert .
                ZVD_Routine( PfI, PatI ) = ( MW, PfI, PatI ) ∧ is_mk_ZVD( MW );
endaxioms;

}

PFLEGER_VW_ABL = {      --enthält die Funktionen zum Abspeichern der Meßwerte
enriches PFLEGER_MESS;
```

```
strict;

nicht_bearbeitet : Messung × Anordnung × Db → Bool;
VW_Mess : Db × Anordnung × PflegerId × Mess_Wert → Db;
VW_Routine : Db × PflegerId × PatId × Mess_Wert → Db;

axioms ∀ Anord : Anordnung, PfB : Messung,
                Wert : Mess_Wert, DB : Db , PfId : PflegerId, PId : PatId in
        δ( nicht_bearbeitet( PfB, Anord, DB ) ) =
                Anord ∈ entAnordnung( DB );
                --ein Pflegebefehl ist nicht bearbeitet, wenn es noch keine
                --passenden Meßdaten gibt.
        δ( nicht_bearbeitet( PfB, Anord, DB ) ) ⇒ nicht_bearbeitet( PfB, Anord, DB ) =
                PfB isin ↓Werte( Anord ) ∧
                ¬ ∃ PfW . PfW ∈ entPflegewert( DB ) ∧
                messung DB ( Anord, PfW ) ∧
                paßt_zu( PfB, ↓Wert( PfW ) );

        δ( VW_Mess( DB, Anord, PfId, Wert ) ) =
                Anord ∈ entAnordnung( DB ) ∧
                δ( getPfleger( PfId, DB ) ) ∧
                pflegbar( DB, ↓PatId( AO ), PfId ) ∧
                ¬ erledigt( Anord, DB ) ∧
                ∃ PfB : Messung . nicht_bearbeitet( PfB, Anord, DB )∧
                paßt_zu( PfB, Wert );
                --Der Pfleger ist für den Patientenbezug der Werte verantwortlich
        δ( VW_Mess( DB, Anord, PfId, Wert ) ) ⇒
        VW_Mess( DB, Anord, PfId, Wert ) = DB' where
                PfW = createPflegewert( attr genkeyPflegewert( DB, λ x. true),
                                        attr datetime( DB ), attr Wert ) and
                DB1 = putPflegewert( DB, PfW ) and
                DB2 = estmessung( DB1, Anord, PfW ) and
                DB' = estmißt( DB2, getPfleger(PfId,DB), PfW );

        δ( VW_Routine( DB, PfId, PId, Wert ) ) =
                δ( getPfleger( PfId, DB ) ) ∧
                δ( getPatient( PId, DB ) ) ∧
                pflegbar( DB, PId, PfId );
        δ( VW_Routine( DB, PfId, PId, Wert ) ) ⇒
        VW_Routine( DB, PfId, PId, Wert ) = DB' where
                PfW = createPflegewert( attr genkeyPflegewert( DB, λ x. true),
                                        attr datetime( DB ), attr Wert ) and
                DB1 = putPflegewert( DB, PfW ) and
                DB2 = estliefert( DB1, getPatient( PId, DB ) ) and
                DB' = estmißt( DB2, getPfleger( PfId, DB ) );
endaxioms;
}

PFLEGER_MED_ABL = {

enriches ANORDNUNG;
strict;
```

```
nicht_gegeben : Med × Anordnung × Db ⟶ Bool;
Med_Gib : Db × Anordnung × PflegerId ⟶ Db × Anordnung;

axioms ∀ AO : Anordnung, DB : Db, M : Med, Dos : Werte, in
        δ( nicht_gegeben( M, AO, DB ) ) =
                AO ∈ entAnordnung( DB ) and
                M isin ↓Therapie( AO );
        δ( nicht_gegeben( M, AO, DB ) ) ⇒
        nicht_gegeben( M, AO, DB ) =
                ¬ ∃ Medik . Medik ∈ EntMedikation( DB ) ∧
                medikation DB ( AO, Medik ) ∧
                MedId( M ) = ↓MedId( Medik ) ∧
                Dosis( M )= ↓Dosis( Medik ) ∧
                is_today( ↓Datum( Medik ), DB );

        δ( Med_Gib( DB, AO, PfId, Dos ) ) =
                AO ∈ entAnordnung( DB ) ∧
                δ( getPfleger( PfId, DB ) ) ∧
                ¬ erledigt( AO, DB ) ∧
                ∃ M : Med . nicht_gegeben( M, AO, DB ) ∧
                Dosis( M ) = Dos;
        δ( Med_Gib( DB, AO, PfId, Dos ) ) ⇒
        Med_Gib( DB, AO, PfId, Dos ) = ( DB', AO' ) where
                NewMed = createMedikation( attr genkeyPflegewert( DB, λ x. true),
                                attr datetime( DB ), attr Dos ) and
                DB1 = putMedikation( DB, NewMed ) and
                DB2 = estmedikation( DB1, AO, NewMed ) and
                DB' = estgibt( DB2, getPfleger( PfId, DB ), NewMed )and
                AO' = if ∀ M : Med . M isin ↓Therapie( AO ) ∧
                        Ende_med( M ) before datetime( DB )
                then    --Dummy-Anordnung zum Terminieren
                        createAnordnung( PatId(AO), attr
                                genkeyAnordnung(DB,λ(p: PatId, n: Nat) . p == ↓PatId(AO)),
                                attr datetime(DB), UNDEF, attr false,
                                attr false, attr [ ], attr [ ])
                else
                        AO
                endif;
endaxioms;
}
```

4.9 Formale Spezifikation der Laborauftragserstellung

Die Laboranordnung fällt etwas aus dem Schema der anderen Anordnungen. Ihr wird zunächst ein Laborauftrag zugeordnet, zu dem dann später Laborwerte gespeichert werden (siehe Ablauf Labor). Auf Station werden (per Knopfdruck) zu den Laboranordnungen in der Datenbank Laboraufträge erstellt und gleichzeitig werden reale, leere Blutproben, die mit Etiketten beschriftet sind, vorbereitet (siehe die Spezifikation der elementaren Transaktion `Auftr_Erst` weiter unten). Ein Laborauftrag ist das elektronische Abbild einer Blutprobe.

In einem zweiten Schritt erfolgt dann (eventuell zeitlich und personell ge-

trennt von der Auftragserstellung) die Blutabnahme (siehe die Spezifikation der Umweltfunktion `Blut_Zapf`), bei der aus einer leeren Blutprobe eine gefüllte Blutprobe entsteht, die dann an das Labor geschickt wird.

Der Entitytyp LabAuftrag ist in Abschnitt 2.1 spezifiziert. Um direkt von einer Auftragsnummer schon auf die Station schließen zu können, auf der der Auftrag generiert wurde, verlangen wir zusätzlich noch die Existenz einer strikten Funktion

```
station: AuftragNr → Attr Station .
```

Der Umwelttyp Blutprobe ist ebenfalls in Abschnitt 2.2 spezifiziert. Die Blutprobe ist keine Entität in der Datenbank, sondern ein Objekt der Umwelt, das aber mit Attributen (oder Etiketten) versehen wird, die eine Verbindung zur Datenbank herstellen. Diese Attribute sind die Auftragsnummer und ein boolesches Attribut, welches angibt, ob für die Blutprobe zusätzlich zu der routinemäßigen Bestimmung des kleinen Blutbilds auch noch das Differentialblutbild ermittelt werden soll. Dies sind die Informationen, die im Labor von den Blutproben direkt (ohne Datenbankzugriff) abgelesen werden können. Da die Vorbereitung von Aufträgen und leeren Blutproben zeitlich und personell auseinanderliegen kann von der Blutabnahme, und da bei der Blutabnahme kein Datenbankzugriff erforderlich sein sollte, werden zusätzliche Etiketten 'Name' und 'Zimmer' auf der Blutprobe zur Information für die Blutabnahme vorgesehen. Dabei wird angenommen, daß nie zwei Patienten gleichen Namens (= Vorname und Nachname) im gleichen Zimmer liegen. Die Namens- und Zimmerinformation wird aber nach der Blutabnahme gelöscht (d.h. die entsprechenden Etiketten werden abgetrennt). Somit sind diese Informationen im Labor nicht mehr ersichtlich. (Siehe Spezifikation von `Blut_Zapf`).

Die Sorte "Blut" ist eine nicht näher spezifizierte Sorte zur Beschreibung des Blutinhalts einer Blutprobe. Solange die Blutprobe noch leer ist (zwischen Auftragserstellung und Blutabnahme), ist der Wert des Attributs Blut noch UNDEF.

```
PFLEGER_LAB_ABL =

{ enriches DB + BLUTPROBE;

station: AuftragNr → Station;
-- erlaubt aus einer Auftragsnummer direkt die Station zu ermitteln.

-- Aufträge auf verschiedenen Stationen erhalten somit verschiedene
-- Auftragsnummern.

Auftr_Erst: Db × Anordnung → Db × Blutprobe;
-- Erstellung eines Laborauftrags (in der Datenbank) zu einer Anordnung. Dabei
-- wird zugleich eine ''reale'' leere Blutprobe mit Etiketten erstellt.

-- Auftr_Erst ist eine elementare Transaktion.


Blut_Zapf: Blutprobe → Blutprobe;
```

```
-- Einfüllen des Blutes des auf dem Etikett angegebenen Patienten in
-- eine leere Blutprobe.

-- Blut_Zapf ist eine Umweltfunktion

station strict;  -- station wird ansonsten nicht weiter spezifiziert
Auftr_Erst strict;

Blut_Zapf strict;


axioms ∀ AO: Anordnung, db: Db, bp, bp': Blutprobe in
  δ( Auftr_Erst(db, AO) ) ⇔ AO ∈ entAnordnung(db) ∧
                            Bb_erwünscht(AO) = attr true ∧
                            ¬( ∃ a: LabAuftrag. (erstellung db)(AO, a));

  δ( Auftr_Erst(db, AO) ) ⇒
     Auftr_Erst(db, AO) = (db', bp) where
        pat = getPatient( ↓PatId(AO), db) and
        patname = Name(pat) and
        zimmernr = Zimmer(pat) and
        mark = if DiffBb_erw(AO) == attr true then attr true else attr false endif and
        nr = genkeyLabAuftrag( λ x: Auftragnr.
                            station(x) = Station(getPatient(pid, db), db) )  and
        auftrag = createLabAuftrag( attr(nr), attr(datetime(db)),mark, UNDEF)  and
        db' = esterstellung( putLabAuftrag(auftrag, db), AO, auftrag)  and
        bp = createBlutprobe(attr(nr), mark, patname, zimmernr, UNDEF);

  δ( Blut_Zapf(bp) ) ⇔  Blut(bp) = UNDEF ∧ Name(bp) ≠ UNDEF ∧
                        Zimmer(bp) ≠ UNDEF;
        -- Vorbedingung: die Blutprobe muss leer sein und mit Patientennamen
        -- und Zimmernr. beschriftet sein.

  Blut_Zapf(bp) = bp' ⇒
       ANr(bp) = ANr(bp') ∧ Differw(bp) = Differw(bp') ∧ Blut(bp') ≠ UNDEF
        ∧ Name(bp) = UNDEF ∧ Zimmer(bp) = UNDEF;
        -- bei der Blutabnahme dürfen die für das Labor relevanten Etiketten
        -- nicht verändert werden. Nach der Blutabnahme ist die Blutprobe
        -- gefüllt (hoffentlich mit dem Blut des durch Name und Zimmernr.
        -- angegebenen Patienten - aber dies wird hier nicht spezifiziert) und
        -- das Namens- und Zimmeretikett wird entfernt, da diese Information
        -- im Labor nicht sichtbar sein soll.
endaxioms;

}
```

4.10 Formale Spezifikation einiger Queries

Die funktionale Essenz enthält eigentlich keine Ausgabefunktionen. Dennoch sind diese dem Anwender wichtig.

Hier werden einige Beispiele gezeigt, an denen man sieht, wie (leicht) es ist, solche Abfragen zu spezifizieren.

```
QUERY_PAT = { enriches SORTEN + DATENBANK;

        --liefert alle Patienten auf einer Station
        query_Pat_Stat : Db × Station → Set Patient;

        axioms ∀ DB : Db, S : Station, Pat : Patient in
                Pat ∈ query_Pat_Stat ( DB, S ) ⇔
                        Pat ∈ entPatient( DB ) ∧
                        Station(Pat) = attr S;
        endaxioms;
}


QUERY_TEMP = { enriches SORTEN + DATENBANK;

        --liefert alle Temperaturwerte zu einem Patienten mit Uhrzeit
        query_TEMP : Db × PatId → Set (DateTime × Temperatur);

        axioms ∀ DB : Db, P : PatId, PfW : Pflegewert, Temp : Mess_Wert, Dat : DateTime in
                ( Dat, Temp ) ∈ query_TEMP( DB, P ) ⇔
                        ∃ An : Anordnung . An ∈ entAnordnung( DB ) ∧
                        Patid( An ) = attr P ∧
                        messung DB ( An, PfW ) ∧
                        is_mk_Temperatur( ↓ Wert( PfW ) ) ∧
                        Temp = Temperatur( ↓ Wert( PfW ) ) ∧
                        Dat = ↓ Erstellung( PfW );
        endaxioms;
}
```

Beispiel für die Anfrage eines Pflegers welchen Patienten noch Blut abgenommen werden muß.

```
QUERY_OPFER = { Enriches HILF;

        --liefert die Patienten, denen Blut abgenommen werden soll
        query_Opfer : Db × PflegerId → Set PatId;

        axioms ∀ DB : Db, PflegId : PflegerId, An : Anordnung, PI : PatId in
                PI ∈ query_Opfer( DB, PflegId ) ⇔
                        ∃ An : Anordnung . An ∈ entAnordnung( DB ) ∧
                        PatId( An ) = attr PI ∧
                        messung DB ( An, PfW ) ∧
                        Bb_erwünscht( An ) = attr true ∧
                        pflegbar( Db, ↓ PatId( An ), PflegId ) ∧
                        ¬ ∃ LA : LabAuftrag . erstellung DB( An, LA );
        endaxioms;
}
```

Die Funktionen zur graphischen, übersichtlichen Darstellung der Ergebnisse sind hier nicht näher spezifiziert. Solche Funktionen wurden in einem Entwurf für die Benutzerschnittstelle exemplarisch spezifiziert (s. [Shi94]).

5 Labor-Ablauf

5.1 Teildatenmodell

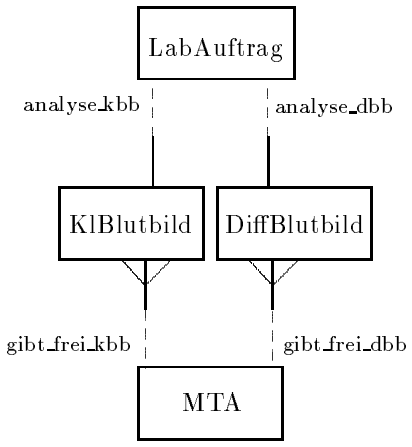

Abbildung 10: Der für die Laborabläufe relevante Teil des E/R- Modells

Wie beim Teildatenmodell der Herzkatheteruntersuchung besitzt auch dieses Datenmodell zusätzlich zu den aus dem Diagramm ersichtlichen Integritätsbedingungen noch statische Integritätsbedingungen, die sich auf die Attributwerte der an den Relationen beteiligten Entitäten beziehen. Wie aus der folgenden Beschreibung der Entitäten ersichtlich ist, besitzen sowohl 'LabAuftrag', als auch 'KlBlutbild' und 'DiffBlutbild' das Schlüsselattribut ANr : AuftragNr. Es wird nun gefordert, daß eine Entität vom Typ KlBlutbild genau dann in Beziehung steht zu einer Entität vom Typ 'LabAuftrag', wenn das Attribut 'ANr' in beiden Entitäten denselben Wert hat. Entsprechendes gilt auch für die Beziehung zwischen Entitäten vom Typ 'LabAuftrag' und 'DiffBlutbild'. Dies läßt sich mit Prädikaten `ACanalyse_kbb` und `ACanalyse_dbb` (wobei AC für 'additional condition' steht) folgendermaßen ausdrücken:

```
ACanalyse_kbb,ACanalyse_dbb : Db → Bool;
ACanalyse_kbb,ACanalyse_dbb strict total;

axioms ∀ db: Db in

ACanalyse_kbb db =
( ∀ bb: KlBlutbild, a: LabAuftrag. (analyse_kbb db)(a,bb) ⇔
  bb ∈ entKlBlutbild(db) ∧ a ∈ entLabAuftrag(db) ∧ ANr(bb)=ANr(a));


ACanalyse_dbb db =
( ∀ diff : DiffBlutbild, a: LabAuftrag. (analyse_dbb db)(a,diff) ⇔
  diff ∈ entDiffBlutbild(db) ∧ a ∈ entLabAuftrag(db) ∧ ANr(diff)=ANr(a)
);
endaxioms;
```

An die Relationen 'gibt_frei_kbb' und 'gibt_frei_dbb' wird ebenfalls eine Zusatzforderung gestellt: Eine Entität vom Typ KlBlutbild (bzw. DiffBlutbild) ist genau dann in der Relation 'gibt_frei_kbb' (bzw. 'gibt_frei_dbb') mit einer Entität vom Typ MTA, wenn das Attribut MTAId in beiden Entitäten übereinstimmt. Um dies zu spezifizieren, verwenden wir Prädikate

```
ACgibt_frei_kbb und ACgibt_frei_dbb:

ACgibt_frei_kbb, ACgibt_frei_dbb:Db → Bool;
ACgibt_frei_kbb, ACgibt_frei_dbb strict total;

axioms ∀ db: Db in

ACgibt_frei_kbb db =
( ∀ bb: KlBlutbild, mta: MTA. (gibt_frei_kbb db)(mta,bb) ⇔
bb ∈ entKlBlutbild(db) ∧ mta ∈ entMTA(db) ∧ MTAId(bb) = MTAId(mta));

ACgibt_frei_dbb db =
( ∀ diff: DiffBlutbild, mta: MTA. (gibt_frei_dbb db)(mta,diff) ⇔
  diff ∈  entDiffBlutbild(db) ∧ mta ∈ entMTA(db) ∧ MTAId(diff) = MTAId(mta));
endaxioms;
```

Zusammen mit den aus dem obigen Diagramm direkt generierbaren statischen Integritätsbedingungen `Canalyse_kbb`, `Canalyse_dbb`, `Cgibt_frei_kbb`, `Cgibt_frei_dbb` (vgl. [Het93]) ergibt sich durch Konjunktion eine statische Integritätsbedingung

```
OKLab : Db → Bool;
```

für den für die Laborabläufe relevanten Teil des Datenmodells.

Zu den einzelnen Entitätstypen

- Bemerkungen zu LabAuftrag und Blutbild befinden sich in 4.9 (vor der Spezifikation `PFLEGER_LAB_ABL`). Die Spezifikation der Attribute dieser Entitäten ist in 2.1 gegeben. Blutbild ist der Typ einer externen Entität (siehe 2.2).
- MTA ist in 2.1 spezifiziert.
- KlBlutbild und DiffBlutbild sind ebenfalls in 2.1 spezifiziert. Die Sorte Blutbild aus der Spezifikation BLUTBILD in der IST-Analyse für HDMS-A [CKL93] wird hier differenziert in zwei Sorten KlBlutbildWert (für das kleine Blutbild) und DiffBlutbildWert (für das Differentialblutbild) mit jeweiligen Konstruktoren `klBlutbild` und `diffBlutbild`, die als Parameter die Werte haben, die im kleinen Blutbild, bzw. Differentialblutbild bestimmt werden. Das Gesamtblutbild läßt sich dann mit Hilfe einer Funktion

  ```
  combine : KlBlutbildWert × DiffBlutbildWert → BlutbildWert
  ```

erzeugen. Wir verzichten hier auf die genaue Spezifikation. Zusätzlich zu den bei der Blutbildanalyse ermittelten Werten (vom Typ KlBlutbildWert bzw. DiffBlutbildWert) werden in den Entitäten KlBlutbild und DiffBlutbild noch folgende Informationen festgehalten:

- die Auftragsnummer des der Blutprobe zugeordneten Laborauftrages
- ein Statusattribut, welches Zusatzinformation zu den Laborwerten liefert. Die Sorte Status ist spezifiziert durch

  ```
  sort Status = normal | kontrolle | nichtverwertbar;
  ```

  wobei mit den Elementen der Sorte Status folgende Information verbunden ist:

`normal`:	Werte sind im normalen Rahmen.
`kontrolle`:	Werte wurden noch einmal kontrolliert, da sie nicht im erwarteten Rahmen waren.
`nichtverwertbar`:	Die Blutprobe war unbrauchbar, und es konnte kein Wert ermittelt werden.

- Die Kennung der MTA, die den Wert freigibt.
- Der Zeitpunkt, zu dem die Werte im Labor eingetragen werden.

5.2 Datenflußdiagramm

Dieses Datenflußdiagramm beschreibt den Ablauf der Blutuntersuchung im Labor. Dabei werden zu einer ankommenden Blutprobe das kleine Blutbild und, wenn erwünscht, auch das Differentialblutbild ermittelt. Die entsprechenden Analysen (klBbAnalyse und diffBbAnalyse) erfolgen außerhalb der Systemgrenzen. Die Analyseergebnisse sind Eingaben der Systemtransaktionen `enterklBlutbild` und `enterDiffBlutbild`, bei denen der Eintrag in die Datenbank erfolgt.

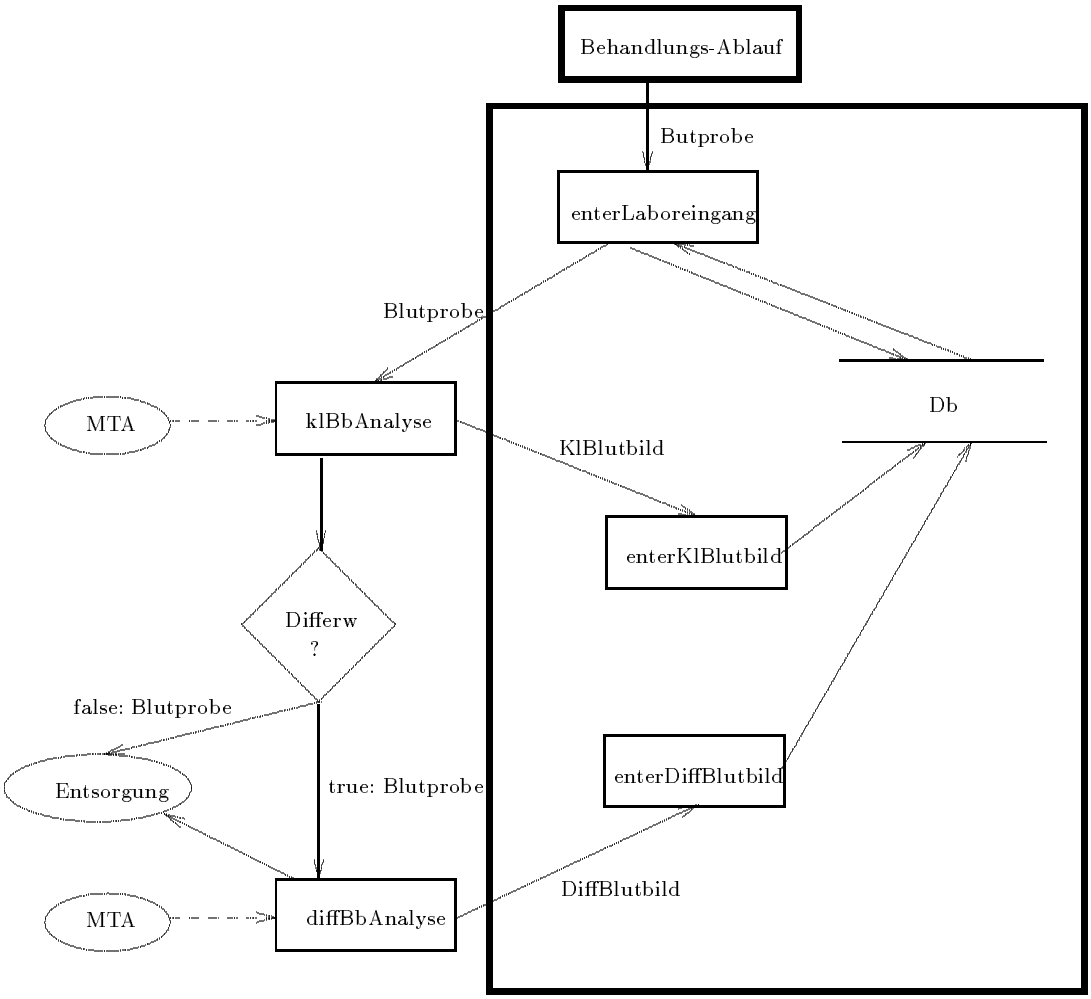

Abbildung 11: Datenflußdiagramm für den Labor-Ablauf

5.3 Zusammenhang zur Ist-Analyse

Der Ablauf 'Blutuntersuchung' mit den Unterabläufen 'Kleines Blutbild' und 'Differentialblutbild' aus der Ist-Analyse wurde hier stark vereinfacht. Es wurde von allen Details der Blutbildbestimmung durch Coultermaschinen und MTA's abstrahiert. Aus dieser Abstraktion resultieren die Umweltfunktionen `klBbAnalyse` und `diffBbAnalyse`.

Das in der Ist-Analyse mit einer Blutprobe ans Labor gesandte NAB-Formular (Notfallanforderungsbogen) wird ersetzt durch die Entität 'LabAuftrag'. Die Verbindung zwischen einer Blutprobe und dem zugehörigen Auftrag ist jetzt durch das Attribut `ANr` (Auftragsnummer) auf der Blutprobe hergestellt. In der Ist-Analyse ist das Ergebnis der Blutbilduntersuchungen die mit dem Blutbild beschriftete Coulterkarte. Diese wird jetzt modelliert durch die Entitäten 'KlBlutbild' und 'DiffBlutbild'. Das Kopieren des Blutbilds auf der Coulterkarte auf das NAB-Formular wird ersetzt durch die Transaktionen `enterklBlutbild` und `enterdiffBlutbild`, in denen die Analyseergebnisse in die Datenbank eingetragen werden und dabei dem zugehörigen Auftrag zugeordnet werden. Die in der Ist-Analyse auftretenden Mehrfachkopierungen der Coulter-Karte entfallen auf Grund der zentralen Verfügbarkeit der Analyseergebnisse in der Datenbank.

5.4 Elementare Transaktionen

Die folgenden elementaren Transaktionen stellen die Schnittstelle des Labors zur Datenbank dar. "Elementar" bedeutet dabei, daß diese Transaktionen entweder als Ganzes oder gar nicht ausgeführt werden dürfen. Sie sind so spezifiziert, daß, wenn sie auf einen Datenbankzustand db angewandt werden, der die statische Integritätsbedingung `OKLab` erfüllt, für den Nachfolgezustand `db'` der Datenbank gilt:

$$\delta(\texttt{db'}) \Rightarrow \texttt{OKLab(db')}.$$

In anderen Worten: `OKLab` ist eine Invariante bzgl. dieser Transaktionen. Da die Transaktionen nur auf dem für das Labor relevanten Teilausschnitt der Datenbank operieren, ist somit auch die globale Integritätsbedingung `OK` eine Invariante dieser Transaktionen.

Laboreingang: `enterLaboreingang`

Für eine eingehende Blutprobe liefert enterLaboreingang genau dann ein definiertes Ergebnis, wenn es zu der Auftragsnummer auf der Blutprobe einen Auftrag in der Datenbank gibt, zu dem noch kein Blutbild oder Differentialblutbild ermittelt wurde und wenn der 'Differw'- Marker auf der Blutprobe mit dem entsprechenden Eintrag im Auftrag übereinstimmt. Ist diese Bedingung erfüllt, so wird das Attribut 'Laboreingang' im Auftrag gesetzt.

Eintragen des kleinen Blutbildes: `enterKlBlutbild`

Hier wird der Wert des durch die Analyse ermittelten kleinen Blutbilds (ergänzt um Statusinformation) zusammen mit der Auftragsnummer der Blutprobe (als Schlüssel) und der Kennung der für den Wert verantwortlichen MTA in die Datenbank übernommen.

Eintragen des Differentialblutbildes: `enterDiffBlutbild`

Hier wird der Wert des durch die Analyse ermittelten Differentialblutbilds (ergänzt um Statusinformation) zusammen mit der Auftragsnummer der Blutprobe (als Schlüssel) und der Kennung der für den Wert verantwortlichen MTA in die Datenbank übernommen.

5.5 System-externe Aktionen in den Laborabläufen

Bestimmung des kleinen Blutbilds: `klBbAnalyse`

Dies ist der Vorgang der Ermittlung der zu der jeweiligen Blutprobe gehörigen Werte des kleinen Blutbildes. Zusätzlich zu den reinen Blutbildwerten und der Auftragsnummer wird ein Status geliefert ("normal" oder "kontrolle" oder "unverwertbar"), sowie ein Identifikator für die MTA, die für diesen Wert verantwortlich zeichnet. Der Blutbildwert, den die Analyse-Funktion liefert, ist der vom Labor freigegebene Wert. Von dieser Funktion der Außenwelt wird erwartet, daß sie total ist. In dem Fall, daß keine sinnvollen Werte ermittelt werden können, muß dies im Status vermerkt werden.

Weiterhin verlangen wir von dieser Funktion, daß sie die Nummern und Marker auf der Blutprobe unverändert läßt, und daß die zu dem ermittelten Wert gehörige Auftragsnummer übereinstimmt mit der Auftragsnummer der eingehenden Blutprobe.

Bestimmung des Differentialblutbilds: `diffBbAnalyse`

Dies ist der Vorgang der Ermittlung der zu der jeweiligen Blutprobe gehörigen Werte des Differentialblutbildes. Zusätzlich zu den reinen Blutbildwerten und der Auftragsnummer wird ein Status geliefert ("normal" oder "kontrolle" oder "unverwertbar"), sowie ein Identifikator für die MTA, die für diesen Wert verantwortlich zeichnet. Der Blutbildwert, den die Analyse-Funktion liefert, ist der vom Labor freigegebene Wert. Von dieser Funktion der Außenwelt wird erwartet, daß sie total ist. In dem Fall, daß keine sinnvollen Werte ermittelt werden können, muß dies im Status vermerkt werden.

Weiterhin verlangen wir von dieser Funktion, daß die zu dem ermittelten Wert gehörige Auftragsnummer übereinstimmt mit der Auftragsnummer der eingehenden Blutprobe.

5.6 Formale Spezifikation der Aktionen (= elementare Transaktionen und Umweltaktionen) in den Laborabläufen

```
LAB_AKTIONEN =

{ enriches DB + BLUTPROBE;
strict;

enterLaboreingang: Db × Blutprobe → Db × Blutprobe;
-- Eingangsprüfung und Eintragen des Eingangszeitpunkts in den
-- zur Blutprobe geh¨orenden Auftrag in der Datenbank.
-- enterLaboreingang ist eine elementare Transaktion.

axioms ∀ db: Db,bp: Blutprobe in
  δ(enterLaboreingang( db, bp) ) ⇔
  let nr = ANr(bp) and mark = Differw(bp)
     in ∃ a: LabAuftrag. a ∈ entLabAuftrag(db) ∧
        ANr(a)= nr ∧ Differw(a) = mark ∧ Laboreingang(a)=UNDEF ∧
        ¬( ∃ bb: KlBlutbild.(analyse_kbb db)(a, bb)) ∧
        ( mark = attr(true) ⇒
                ¬(∃ diff: DiffBlutbild. (analyse_dbb db)(a, diff)) )
  endlet;

  δ(enterLaboreingang(db, bp) ) ⇒
  letrec nr = ANr(bp) and
         a' = setLaboreingang(getLabAuftrag(↓nr,db),attr(datetime(db))) and
         db'= updateLabAuftrag(↓nr,a',db)
     in enterLaboreingang(db, bp) = (db', bp)
  endlet;
endaxioms;

-----------------------------------------------------------------
klBbAnalyse: Blutprobe → Blutprobe × KlBlutbild;
-- Bestimmung des kleinen Blutbilds
-- klBbAnalyse ist Umweltfunktion.
klBbAnalyse total;

axioms ∀ bp: Blutprobe, ∀⊥ bp': Blutprobe, bb: KlBlutbild in
  klBbAnalyse( bp) = (bp', bb ) ⇒
        (ANr(bp')= ANr(bp) ∧ ANr(bb) = ANr(bp) ∧
        Differw(bp) = Differw(bp') ∧
        (Status(bb)≠attr(nichtverwertbar) ⇒ attr(Wert(bb))≠UNDEF) );
endaxioms;

-----------------------------------------------------------------
enterKlBlutbild: Db × KlBlutbild → Db;
-- Eintragen des kleinen Blutbilds in die Datenbank
-- enterKlBlutbild ist eine elementare Transaktion.

axioms ∀ db: Db, bb: KlBlutbild in

  δ(enterKlBlutbild( db, bb ) ) ⇔
  let nr = ↓ANr(bb) and mtaId = ↓MTAId(bb) in
```

```
        δ(↓Laboreingang(getLabAuftrag(nr, db))) ∧ δ(getMTA(mtaId, db)) ∧
        ¬(∃ klbb: KlBlutbild. (analyse_kbb db)(getLabAuftrag(nr, db), klbb))
  endlet;

  δ(enterKlBlutbild( db, bb) ) ⇒
   letrec a   = getLabAuftrag(↓aNr(bb), db) and
          mta = getMTA(↓MTAId(bb), db) and
          bb' = setZeitpunkt(bb, attr(datetime(db)) and
          db' = putKlBlutbild(bb', db) in
               enterKlBlutbild( db, bb) =
               estanalyse_kbb(estgibt_frei_kbb(db', mta, bb'),a, bb')
   endlet;
endaxioms;

-----------------------------------------------------------------
diffBbAnalyse: Blutprobe → Blutprobe × DiffBlutbild;
-- Bestimmung des Differentialblutbilds
-- diffBbAnalyse ist Umweltfunktion.

diffBbAnalyse total;

axioms ∀ bp, Blutprobe, ∀⊥ bp': Blutprobe, diff: DiffBlutbild in
  diffBbAnalyse(bp) = (bp', diff) ⇒
    ANr(bp')= ANr(bp) ∧ ANr(diff) = ANr(bp) ∧
    (Status(diff)≠attr(nichtverwertbar) ⇒ attr(Wert(diff))≠UNDEF);
endaxioms;

-----------------------------------------------------------------
enterDiffBlutbild: Db × DiffBlutbild → Db;
-- Eintragen des Differentialblutbilds in die Datenbank
-- enterDiffBlutbild ist elementare Transaktion.

axioms ∀ db: Db, diff: DiffBlutbild in
  δ(enterDiffBlutbild( db, diff) ⇔
  let nr = ↓ANr(diff)and mtaId = ↓MTAId(diff) in
        δ(↓Laboreingang(getLabAuftrag(nr, db))) ∧ δ(getMTA(mtaId, db)) ∧
        ¬(∃ diffbb: DiffBlutbild.
          (analyse_dbb db)(getLabAuftrag(nr, db), diffbb));
  endlet;

  δ(enterDiffBlutbild( db, diff ) ⇒
  letrec a     = getLabAuftrag(↓aNr(diff), db) and
         mta   = getMTA(↓MTAId(diff), db) and
         diff' = setZeitpunkt(diff, attr(datetime(db)) and
         db'   = putKlBlutbild(diff', db) in
              enterDiffBlutbild( db, diff) =
              estanalyse_dbb(estgibt_frei_dbb(db',mta, diff'),a, diff')
  endlet
endaxioms;
}
```

6 Herzkatheter-Untersuchung

6.1 Teildatenmodell

Folgender Teil des HDMS-A Datenmodells ist für die mit der Herzkatheter-Untersuchung im Zusammenhang stehenden Abläufe relevant:

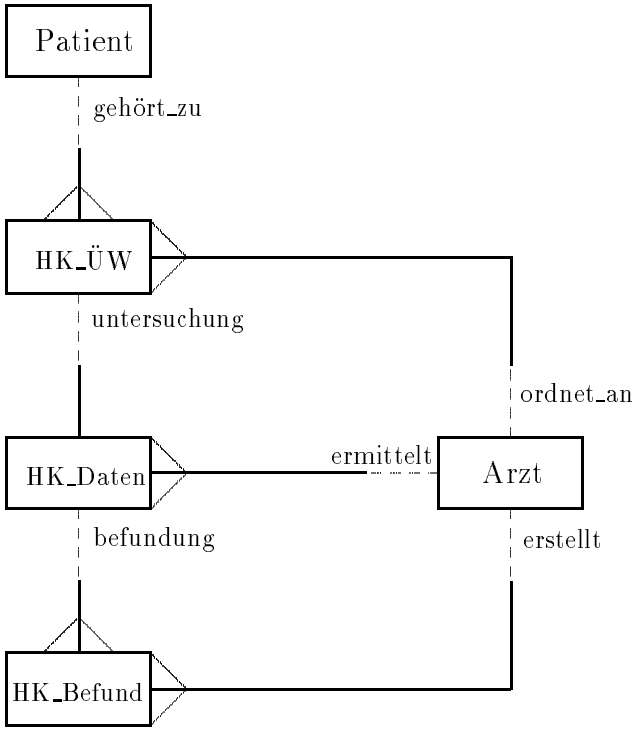

Abbildung 12: Teildatenmodell HK

Dieses Datenmodell besitzt zwei nicht in obigem Diagramm dargestellte Integritätsbedingungen:

- Wenn eine Entity des Typs HK_ÜW mit einer Entity des Typs HK_Daten in der Relationship untersuchung steht, so hat das Attribut HKAOId bei beiden Entities den gleichen Wert. In SPECTRUM läßt sich diese Bedingung durch folgendes Prädikat formulieren:

```
C1 : Db → Bool;
C1 strict total;
axioms ∀ db in
  C1 db = (∀ao,hkd. untersuchung db (ao,hkd) ⇔ ao ∈ entHK_AO db ∧
                  hkd ∈ entHK_Daten db ∧ HKAOId ao = HKAOId hkd);
endaxioms;
```

- Zu jedem Patienten existiert höchstens eine noch nicht bearbeitete HK-Überweisung:

```
C2 : Db → Bool;
C2 strict total;
axioms ∀ db in
  C2 db = ∀p, h1, h2 . gehört_zu db (p,h1) ∧ gehört_zu db (p,h2) ∧
          (¬∃d. untersuchung db (h1,d)) ∧ (¬∃d. untersuchung db (h2,d))
          ⇒ h1=h2
endaxioms;
```

Die Beschreibung der hier verwendeten Entitytypen befindet sich zusammen mit dem Gesamtdatenmodell in Kapitel 2.

6.2 Zusammenhang zur IST-Analyse

Die beiden in dem Teildatenmodell von Abbildung 12 neu dazugekommenen Entitytypen HK_Daten und HK_Befund sind durch Aufteilung aus der in der IST-Analyse eingeführten Sorte Hk_befund hervorgegangen. Die Sorte Hk_befund umfaßt sowohl die bei der Herzkatheter-Operation anfallenden Daten als auch den daraus entstandenen Befund. Da die Herzkatheter-Operation und die Befundung der Herzkatheter-Daten zwei (räumlich wie fachlich) völlig getrennte Vorgänge darstellen, wurde die Sorte Hk_befund in der funktionalen Essenz durch die beiden Entitytypen HK_Daten und HK_Befund dargestellt. Im Gegensatz zur IST-Analyse, die versucht, die genaue Struktur der sehr komplexen Sorte Hk_befund zu spezifizieren, läßt die Funktionale Essenz die Struktur der zu erstellenden medizinischen Daten (vgl. die Sorten Druckkurven, Befund, ...) offen. Es wird davon ausgegangen, daß diese Art von Sorten im Lauf der weiteren Entwicklung nur in enger Zusammenarbeit mit Medizinern adäquat spezifiziert werden kann.

6.3 Datenflußdiagramm

Das Datenflußdiagramm zeigt die HK-Untersuchung im Zusammenspiel mit der Umwelt und dem Arzt-Ablauf.

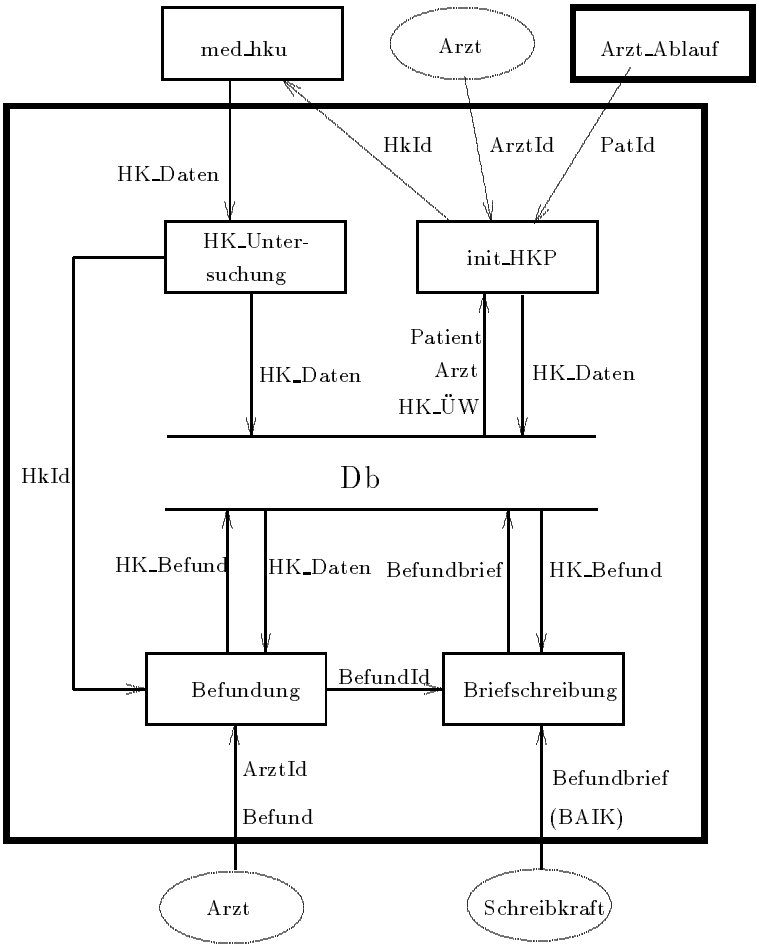

Abbildung 13: DFD Herzkatheter-Untersuchung

6.4 Elementare Transaktionen

Der Herzkatheter-Ablauf umfaßt folgende elementare Transaktionen:

- **init_HKP**: Operationsvorbereitung, Prüfen der Voraussetzungen für Eingriff.
- **HK_Untersuchung**: Chirurgischer Eingriff, Ermitteln der HK_Daten und Eintragen der Daten in die Datenbank.
- **Befundung**: Erstellen und Speichern eines HK-Befundes.
- **Briefschreibung**: HK-Befund in Briefform bringen, unterstützt durch das BAIK-System.

```
HK_Aktionen = { enriches DB;

strict;

akt_ao: Db × PatId → HK_ÜW;

axioms ∀ db, pi, ao in
δ(akt_ao(db,pi)) ⇔
  letrec pat = getPatient(pi,db)
    in δpat ∧ ∃ao. gehört_zu db (pat,ao) ∧ ¬∃hkd. untersuchung db (ao,hkd)
  endlet;

akt_ao(db,pi) = ao ⇒
  letrec pat = getPatient(pi,db)
    in gehört_zu db (pat,ao) ∧ ¬∃hkd. untersuchung db (ao,hkd)
  endlet;
endaxioms;

init_HKP : Db × PatId × ArztId → Db × HkId;

axioms ∀ db, db', pi, ai, hkid in
    δ(init_HKP(db,pi,ai)) ⇔
      letrec patient = getPatient(pi,db) and
          arzt    = getArzt(ai,db)
      in δpatient ∧ δarzt ∧ ∃ao. gehört_zu db (patient,ao) ∧
         ¬∃hkd.  untersuchung db (ao,hkd)
  endlet;

init_HKP(db,pi,ai) = (db',hkid) ⇔
  letrec arzt = getArzt(ai,db) and
      patient = getPatient(pi,db) and
      ao      = akt_ao(db,pi) and
      hkd     = createHK_Daten(attr hkid,UNDEF,attr(datetime db),UNDEF,UNDEF,UNDEF)
  in hkid = ↓(HKAOId ao) ∧
     db' = estermittelt(estuntersuchung(putHK_Daten(db,hkd),ao,hkd),arzt,hkd)
  endlet;
endaxioms;

-- Funktion med_hku beschreibt den medizinischen Ablauf, der die HK_Daten
-- liefert.
med_hku: HkId → HK_Daten;
med_hku total;

axioms ∀ i in
  ↓HKAOId(med_hku i) = i;
endaxioms;

HK_Untersuchung: Db × HK_Daten → Db × HkId;

axioms ∀ db, hkd, db', in
δ(HK_Untersuchung(db,hkd)) =
        letrec hkid = ↓(HKAOId(hkd)) and
               hkd' = getHK_Daten(hkid,db)
```

```
        in δ hkd' ∧ Endezeit hkd' = UNDEF endlet;

letrec hkid = ↓(HKAOId(hkd))
in HK_Untersuchung(db,hkd) = (db', hkid) ⇒
     (db' = updateHK_Daten(hkid, hkd, db) endlet;
endaxioms;

Befundung : Db × HkId × ArztId × Befund → Db × BefundId;

axioms ∀ db, hkid, ai, b, db' bi in
δ(Befundung(db,hkid,ai,b)) =
        letrec  hkd = getHK_Daten(hkid,db) and
               arzt = getArzt(ai,db)
        in δ hkd ∧ δ arzt ∧ Endezeit hkd ≠ UNDEF endlet;

Befundung(db,hkid,ai,b) = (db',bi) ⇒
        (bi = genkeyBefund(db, λx. true)) ∧
        (db' = letrec hkd = getHK_Daten(hkid,db) and
                      arzt = getArzt(ai,db) and
                       bef = createHK_Befund(attr bi, attr(datetime db),
                                             attr b, UNDEF)
               in esterstellt(estbefundung(putHK_Befund(db,bef),hkd,bef),
                              arzt,bef)
               endlet);
endaxioms;

Briefschreibung : Db × BefundId × Befundbrief → Db;

axioms ∀ db, bi, brief, db' in
δ(Briefschreibung(db,bi,brief)) =
        letrec bef = getHK_Befund(bi,db)
        in δ bef ∧ Befundbrief bef = UNDEF endlet;

Briefschreibung(db,bi,brief) = db' ⇔
        (db' = letrec bef = getHK_Befund(bi,db) and
                       bef' = createHK_Befund(attr bi, Befundungsdatum bef,
                                              Befund bef, attr brief)
               in updateHK_Befund(bi, bef', db) endlet);
endaxioms;
}
```

Query-Funktionen

Um den HK_Befund bzw. den Befundbrief zu erstellen, benötigt der Arzt (die Schreibkraft) lesenden Zugriff auf die HK_Daten (den HK_Befund). Diese Abfragen sind jedoch so einfach, daß es dafür keiner eigenen Funktionen bedarf. Die Zugriffsfunktionen `getHK_Daten` und `getHK_Befund` reichen aus.

7 Entlassung

7.1 Schnittstelle zum Datenmodell

Relevanter Ausschnitt aus dem E/R-Modell:

Abbildung 14: Teildatenmodell Entlassung

7.2 Allgemeine Bemerkungen

Wir können davon ausgehen, daß der Patient unter einer PatId erfaßt ist und daß wir die PatId kennen. Also ist die Aktivität "Aufnahmekarte suchen" nicht mehr essentiell; es handelt sich um einen einfachen Speicherungsvorgang. Der Diagnosen-Statistik-Vordruck sowie evtl. Anlagen zur Abgangsmeldung werden unverändert weitergereicht an das Finanz- und Rechnungswesen; sie befinden sich außerhalb der Systemgrenzen. Die Abgangsmeldung scheint nur der Kommunikation zwischen der Station und der Aufnahme (die die Aufnahmekarten-Kartei führt) zu dienen. Nachdem die Aufnahmekarten-Kartei als Aufnahme-Geschichte Bestandteil des Patienten-Archivs geworden ist, können alle Entlassungs-Aktivitäten vollständig von der Station aus erledigt werden. Auch bisher konnte der Patient offensichtlich direkt von der Station aus heimgehen. Grundsätzlich ist die Entlassung nicht die triviale Umkehroperation zur Aufnahme. Die Informationen zu Station und Zimmer sind bei der Entlassung offensichtlich aussagelos geworden und werden wieder gelöscht. Bei einer evtl. Neuaufnahme können diese Werte (ebenso wie Adresse, Hausarzt, Kostenträger) überschrieben werden.

7.3 Datenflußdiagramm

Das Datenflußdiagramm zeigt den Zusammenhang des Entlassungs-Ablaufs mit dem Arzt-Ablauf und die Kommunikation über die Systemgrenze hinaus (Eingaben vom Arzt und Ausgaben an das Rechnungswesen).

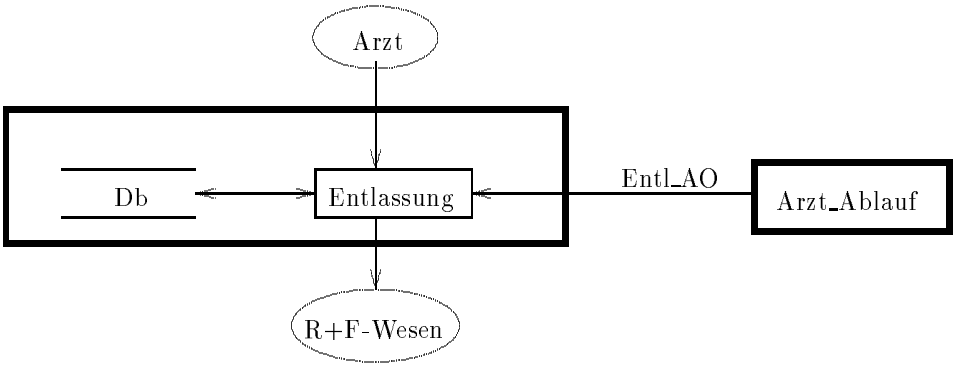

Abbildung 15: DFD Entlassung

7.4 Formale Spezifikation der elementaren Transaktionen

```
ENTLASSUNG_AKTIONEN = {
enriches DB;

entlassung: Db × Entl_AO → Db × EntlNachricht;
entlassung strict total;
axioms ∀ db, eao in
  entlassung(db, eao) = (db', en)
  where
        pi  = PatId(eao) and
        db' = updatePatient(pi,
                setStation(setZimmer(getPatient(pi, db), UNDEF), UNDEF), db) and
        en  = createEntlNachricht(attr pi, Name(getPatient(pi,db)),
                                  attr (datetime db), EntlassungsZiel(eao));
endaxioms;
```

Für zahlreiche, fruchtbare Anregungen bedanken sich die Autoren bei "Ramses" (A. Heckler) und bei Malte Grosse.

Für die mühevolle redaktionelle Arbeit bedanken sich die Autoren bei David von Oheimb.

Literatur

[BFG+93a] Manfred Broy, Christian Facchi, Radu Grosu, Rudi Hettler, Heinrich Hussmann annd Dieter Nazareth, Franz Regensburger, Oscar Slotosch, and Ketil Stølen. The Requirement and Design Specification Language SPECTRUM An Informal Introduction Part I. Technical Report TUM-I9311, TU München, 1993.

[BFG+93b] Manfred Broy, Christian Facchi, Radu Grosu, Rudi Hettler, Heinrich Hussmann, Dieter Nazareth, Franz Regensburger, Oscar Slotosch, and Ketil Stølen. The Requirement and Design Specification Language SPECTRUM An Informal Introduction Part II. Technical Report TUM-I9312, TU München, 1993.

[CKL93] Felix Cornelius, Marcus Klar, and Michael Löwe. Ein Fallbeispiel für KorSo: IST-Analyse für HDMS-A. Technical Report 93-28, Technische Universität Berlin, 1993.

[Het93] R. Hettler. Zur Übersetzung von E/R-Schemata nach SPECTRUM. Technical Report TUM-I9333, TU München, 1993.

[Huß93] H. Hußmann. Zur formalen Beschreibung der funktionalen Anforderungen an ein Informationssystem. Technical Report TUM-I9332, Technische Universität München, 1993.

[LCFW92] M. Löwe, F. Cornelius, J. Faulhaber, and R. Wessäly. Ein Fallbeispiel für KORSO: Das heterogene verteilte Managementsystem HDMS der Projektgruppe Medizin-Informatik (PMI) am Deutschen Herzzentrum Berlin und an der TU Berlin — Ein Vorschlag. Technical Report 92-45, TU Berlin, December 1992.

[Nic93] Friederike Nickl. Ablaufspezifikation durch Datenflußmodellierung und stromverarbeitende Funktionen. Technical Report TUM-I9334, TU München, 1993.

[Shi94] H. Shi. Benutzerschnittstelle und -Interaktion für die HK-Untersuchung. Technical Report at the Universität Bremen, to appear, February 1994.