HDMS-A und OBSCURE in KORSO

Die Funktionale Essenz von HDMS–A aus Sicht
der algorithmischen Spezifikationsmethode

Teil 3: Die Spezifikation der atomaren Funktionen

Christoph Benzmüller
Technischer Bericht **A/06/93**

Dezember 1993

Christoph Benzmüller (Autor)

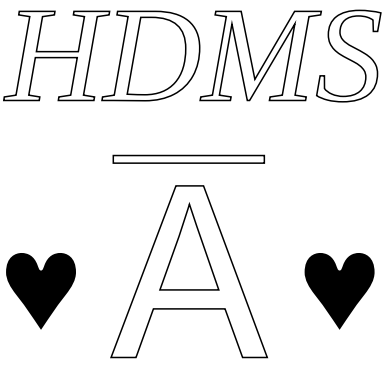
christoph.benzmueller@cs.uni–sb.de

unter Mitarbeit von

Serge Autexier und Ramses A. Heckler

und unter Beratung durch

Stefan Conrad (Uni Braunschweig)
Rudi Hettler (TU München)



Christoph Benzmüller

christoph.benzmueller@cs.uni–sb.de

Die Spezifikation der atomaren Funktionen

Christoph Benzmüller

Dezember 1993

Inhaltsverzeichnis

1 Einleitung

1.1 Das allgemeine Vorwort

The following report is part of the central case study HDMS-A[1] within the german national project KORSO[2]. This study is dedicated to the development of a complex information system for the support of the patient data administration in the specialized heart disease clinic DHZB[3]. While the developpers group PMI[4] develops the real system for the clinic called HDMS, the project KORSO's aim was the rigorous development of an abstracted version of HDMS by exclusive use of pure formal methods. The abstraction refers both to number of modelled documents and depth of treatment, while still considering the relevant aspects in a partly parameterized way.

The investigation of HDMS-A has been done by 11 partners within KORSO. An extended overview provides [CHL94a, CHL94b]. The main topics are: an actual state analysis of the selected documents in the patient record as well as of the existing and motivating problems concerning safety and security, distribution and effectiveness (see [CKL93]); the requirements analysis and specification, beginning with a description of the chosen policy ([Huß93]) and two technical formalisms providing means for the translation of entity-relationship diagrams as well as data flow diagrams into SPECTRUM (see [Het93, Nic93]), finally the requirements specification itself ([SNM+93]); the main field of safety and security treated and in general ([GH93]) and concretely: [Ren94, Ste93]; finally the investigation of the integration of existing software components into a formal development: [Con93, Dam93, Sch94, Shi94, MZ94]. Apart from those there were few specialized contributions to selected topics: [Hec93, Aut93, Ben93, Fuc94, SH94].

Any of the cited reports can be obtained directly from the authors. Details about that can be found in [CHL94a, CHL94b].

1.2 Die Fallstudie HDMS-A

Ziel der Fallstudie HDMS-A innerhalb des Forschungsprojekts KORSO ist die praxisnahe Untersuchung und Weiterentwicklung in der Theorie entwickelter Methoden des formalen Softwareengineerings. Einen allgemeinen Überblick zur HDMS-A Fallstudie vermittelt insbesondere [CHL94a, CHL94b]. Als

[1] Heterogeneous Distributed Information Management System

[2] Korrekte (=correct) Software, sponsored by the german ministry for research and technology

[3] Deutsches Herzzentrum Berlin

[4] Projektgruppe Medizin Informatik am DHZB und der TU Berlin

konkret zu untersuchendes Beispiel wurde ein in der Entwicklung befindliches Softwaresystem für das Berliner Herzzentrum gewählt. Nachdem eine Analyse der Anforderungen an das System durchgeführt wurde (siehe hierzu [CKL93]), bestand die weitere Aufgabe von HDMS-A darin, diese Anforderungssicht mit formalen Mitteln zu beschreiben. Neben einer Spezifikation in der Spezifikationssprache SPECTRUM, die in München erstellt wurde, sollte in Saarbrücken parallel eine Spezifikation mit dem OBSCURE-System angefertigt werden. Im Gegensatz zur Münchner Spezifikation sollte sich diese nur auf einen Ausschnitt des insgesamt zu modellierenden Systems konzentrieren. Näheres hierzu ist in [Hec93] zu finden.

1.3 Zu diesem Bericht

Dieser Bericht ist der 3.Teil des Abschlußberichts des Lehrstuhls von Prof. Loeckx zum Fallbeispiel HDMS-A. Er enthält die Spezifikation des Ablaufs 'Herzkatheter-Untersuchung' in der Spezifikationssprache OBSCURE. Nur in Verbindung mit den Berichten [Hec93] und [Aut93] vermittelt er einen allgemein verständlichen Eindruck der geleisteten Arbeit. Die folgende Auflistung vermittelt einen Überblick zum Inhalt der 3 Teile des Gesamtberichts:

1. Teil
   In [Hec93] wird die Aufgabe von HDMS-A, der zu modellierende Ausschnitt und die Methodik erläutert. Einige Unterschiede der OBSCURE-Spezifikation zur SPECTRUM-Version - wie zum Beispiel die unterschiedliche Handhabung der Schlüsselgenerierung - werden dort diskutiert. Spezifische Begriffe, die in dem folgenden Bericht (3.Teil) oftmals ohne weitere Erläuterung verwendet werden, werden - sofern keine explizite Referenz angegeben wird - in diesem 1.Teil näher erklärt.

2. Teil
   [Aut93] beschreibt ein Schema zur Transformation einer Datenbankbeschreibung auf der Grundlage einer Entity/Relationship-Modellierung in eine Spezifikation in OBSCURE. Mithilfe dieses Schemas kann aus den E/R-Diagrammen zum HDMS-A Beispiel eine vollständige Spezifikation der Datenmodellebene abgeleitet werden, auf der die in diesem Bericht vorgestellte Spezifikation der atomaren Ebene aufsetzt. (Anmerkung: Begriffe wie 'Datenmodellebene' und 'atomare Ebene' werden - wie bereits erwähnt - in [Hec93] eingeführt.)

3. Teil
   Der Inhalt des folgenden Berichts konzentriert sich, aufbauend auf den beiden vorgenannten, auf die Modellierung des Ausschnitts HK-Untersuchung der atomaren Ebene von HDMS-A. Sämtliche in dieser Spezifikation importierten Sorten und Operationen müssen bereitgestellt werden von einer, bisher nicht existierenden, konkreten Spezifikation der Datenmodellebene, die man durch Anwendung des Schemas in [Aut93] auf das E/R-Diagramm des Gesamtdatenmodells zu HDMS-A gewinnen kann. Diesen Zusammenhang erläutert auch die folgende Darstellung.

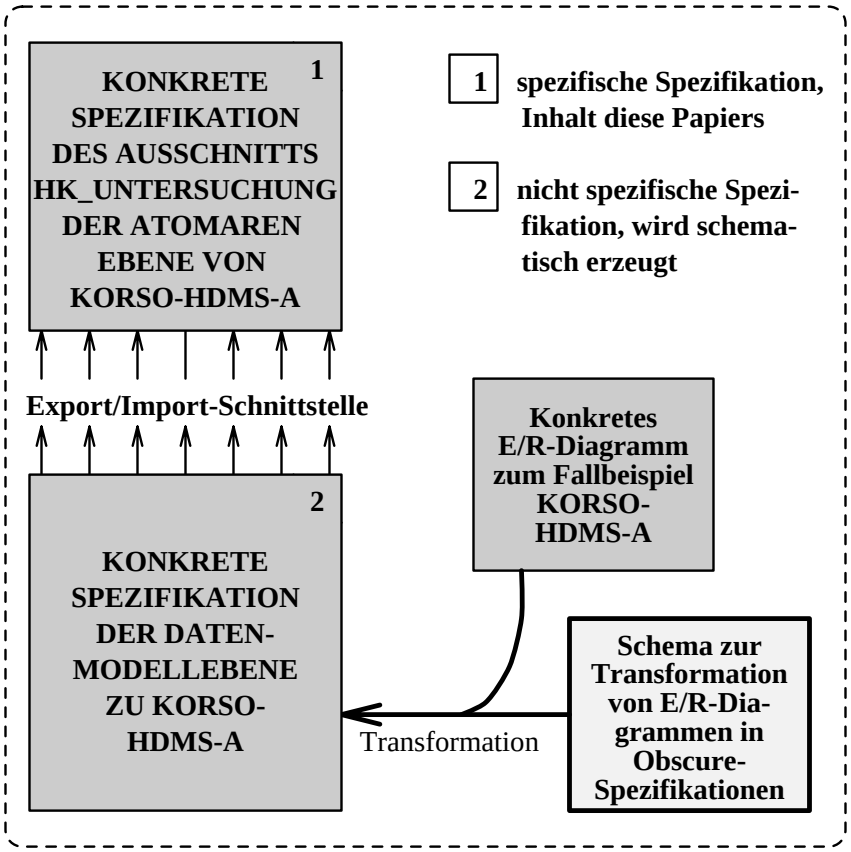


Abbildung1: Einordnung der vorliegenden Spezifikation

1.4 Anmerkungen zur Form

Dieses Papier wurde mithilfe des OPTICAL-Systems erstellt. Dieses System ermöglicht es, Spezifikationsquelltexte geeignet zu zerlegen und mit ausführlichem Kommentar in LaTeX-Format anzureichern. Aus dem so erstellten File kann man mithilfe zweier Programme dann einerseits zu Dokumentationszwecken ein komplettes LaTeX-File generieren, andererseits die isolierte OBSCURE-Spezifikation extrahieren. Damit ist sichergestellt, daß die in diesem Papier vorgestellten Spezifikationstexte auch tatsächlich vom Parser des OBSCURE-Systems auf syntaktische Korrektheit überprüft wurden. Näheres hierzu findet man in Kapitel 6, bzw. in [Hec92].

1.5 Anmerkungen zu konkreten Schlüsseln

In [Aut93] wird die konkrete Schlüsselsorte eines Entitätstyps spezifiziert, als ein n-Tupel über dessen n Schlüsselattributsorten. Für Entitätstypen mit nur einer Schlüsselattributsorte - im Ausschnitt HK_Untersuchung treten vornehmlich solche auf - weiche ich von diesem Vorschlag ab, und identifiziere die konkrete Schlüsselsorte mit dieser einen Schlüsselattributsorte.

1.6 Besonderheiten im Vergleich zur Münchner Spezifikation

Im wesentlichen ist die vorliegende Spezifikation sehr ähnlich zur Münchner Version (siehe [SNM+93]). Die Unterschiede gründen größtenteils auf dem Übergang von axiomatischer zu algorithmischer Spezifikationsmethode, der Beschränkung von OBSCURE auf PL1 (mit Gleichheit) und Besonderheiten der Spezifikationssprachen (z.B. Definiertheitsprädikat in SPECTRUM).
Im folgenden sollen die wesentlichen Änderungen, zu denen wir uns entschlossen haben, kurz beschrieben werden.

- Existenzquantoren
  Die Existenzquantoren in der SPECTRUM-Spezifikation müssen in der OBSCURE-Spezifikation durch eine konstruktive Operation ersetzt werden.
  Betrachten wir ein Beispiel einer Verwendung des Existenzquantors. In der SPECTRUM-Spezifikation zur Operation *akt_uw* (siehe Ablauf HK-Untersuchung in [SNM+93], Axiome zu Funktion "init_HKP"') taucht folgende Formulierung auf:
  $\neg(\exists\ uw\ untersuchung\ db\ (uw, hkd))$

Diese Formulierung kann in OBSCURE ersetzt werden durch (siehe hierzu: Spezifikationstext der Operation *all_not_in_untersuchung* in 4.4):
not is_inuntersuchung$(auwi, db)$
Das Prädikat *is_in_untersuchung*$(auwi, db)$ muß dabei in dem Träger db überprüfen, ob es einen Träger hkd (der Sorte HK_Daten) in db gibt, der zu dem, durch den abstrakten Schlüssel[5] $auwi$ (der Sorte $AKeyHK\_UW$) referenzierten, Träger uw (der Sorte HK_UW) in der Relationship *untersuchung* steht. Diese Operation muß also die Einträge der Relationship *untersuchung* aus db extrahieren, und dann nachschauen, ob es einen entsprechenden Eintrag zu der durch $auwi$ referenzierten HK_Überweisung gibt. Solche Operationen sind schematischer Natur und können deshalb durch die schematisch generierte Datenmodellebene bereitgestellt werden. Von der Spezifikation der atomaren Ebene werden sie importiert.

- Definiertheitsprädikat
  Die Definiertheit einer elementaren Transaktion wird in der SPECTRUM-Spezifikation durch Verwendung des SPECTRUM-Definiertheitsprädikats δ beschrieben. In OBSCURE führen wir anstelle des SPECTRUM-Definiertheitsprädikats je eine boolsche Operation $atom\_func\_def$: ... $\rightarrow$ *bool* zu den elementaren Transaktionen ein (auch wir nennen diese Operationen im folgenden Definiertheitsprädikate). Die Idee ist, mithilfe dieser Operationen festzulegen, wann eine elementare Transaktion fehlerfrei abläuft. Deshalb erfolgt die Spezifikation der elementaren Transaktionen immer nach folgendem Schema: Mittels des OBSCURE-Sprachkonstrukts IF-THEN-ELSE-FI wird zunächst im IF-Teil mit dem entsprechenden Definiertheitsprädikat überprüft, ob die Transaktion auf den aktuellen Argumenten fehlerfrei ablaufen kann. Ist dies der Fall, so spezifiziert der THEN-Teil, wie die elementare Transaktion auf die Eingabe reagiert. Liefert die Überprüfung jedoch "false", so legt der ELSE-Teil fest, daß die elementare Transaktion die ERROR-Konstante der entspechenden Zielsorte zurückliefert. Näheres zu dieser Thematik findet man in [Hec93], Abschnitt 4.3.

- Kreuzprodukt zweier Sorten als Zielsorte
  Die elementaren Transaktionen bilden zum Teil in Kreuzprodukte zweier Sorten ab. Dies ist in OBSCURE nicht möglich. Deshalb führen

[5] Die Begriffe abstrakter und konkreter Schlüssel werden in [Hec93] näher beschrieben.

wir an diesen Stellen jeweils eine Hilfssorte ein, deren Trägermenge aus dem jeweiligen Kreuzprodukt besteht. Mithilfe eines parametrisierten Aufrufs des Standardmoduls MK_PAIR[6] angewendet auf die beiden Teilsorten des Kreuzprodukts können wir diese Hilfssorten spezifizieren.

- Schlüsselgenerierung
  Schlüssel haben die Aufgabe, Entities eindeutig zu identifizieren. Für das zu modellierende System stellt sich das Problem der Generierung neuer eindeutiger Schlüssel. Die SPECTRUM-Spezifikation stellt zur Schlüsselgenerierung Operationen $gen\_key\_E_i$ (für alle Entititätstypen E_i) höherer Ordnung zur Verfügung. Ihnen kann ein Prädikat übergeben werden, mit dem eine spezielle Anforderung an den zu bestimmenden Schlüssel formuliert werden kann. In der SPECTRUM-Spezifikation werden somit die schematischen Anteile der Schlüsselgenerierung von den spezifischen getrennt (hierzu siehe insbesondere [Hec93], Abschnitte 3.3.2, 5.1.6 und 6.2.1). Die Modellierung einer entsprechenden Operation in OBSCURE ist nicht möglich. Wir sind vielmehr gezwungen, die Funktionalität höherer Ordnung aufzulösen und uns bereits jetzt zu jedem Entitytyp für eine spezifische Anforderung (neben der Schlüsseleigenschaft) an den zu bestimmenden Schlüssel zu entscheiden und ferner einen konkreten Algorithmus zur Schlüsselgenerierung anzugeben. Dies kann man als einen Nachteil dieser Spezifikation ansehen. Wir müssen uns an dieser Stelle bereits auf konkrete Anforderungen an einen zu bestimmenden Schlüssel festlegen, obwohl dies auf Anforderungsebene nicht erforderlich scheint. Die Schlüsselgenerierungsoperationen sind also in unserer Spezifikation spezifisch und werden deshalb auch erst auf der atomaren Ebene eingeführt, während diese in der Münchner Spezifikation von der schematisch erzeugbaren Datenmodellebene bereitgestellt werden.
  Die spezifischen Schlüsselgenerierungsoperationen in der OBSCURE-Spezifikation greifen allerdings zurück auf schematische Operationen zur Generierung eines neuen Trägers einer Attributsorte, ausgehend von einer Menge von Trägern dieser Sorte. Diese Operationen (*next*-Operationen) werden in unserer Spezifikation von der schematischen Datenmodelleben bereitgestellt. Weitere Informationen hierzu findet man in [Hec93] und in 4.3.

[6] Das OBSCURE-System stellt dem Spezifizierer eine Datenbank von oft benötigten Standardspezifikationen für Listen, Mengen, Tupel, usw. zur Verfügung.

2 Beschreibung der Struktur der Spezifikation

Die folgende Gesamtspezifikation der atomaren Ebene (siehe [Hec93]) setzt sich zusammen aus einem Modul BASIS und einem Modul HK_UNTERSUCHUNG (in einer kompletten Spezifikation zur atomaren Ebene würden hier die Spezifikationen sämtlicher Abläufe kombiniert) und ferner aus der Anwendung des OK-Prädikats und einer FORGET-Konstruktion. Diese vier Stichpunkte werden im folgenden näher erläutert:

1. Der Modul BASIS
   Wie bereits in der Einleitung (siehe Abschnitt 1) beschrieben, ergibt sich im Vergleich zur SPECTRUM-Spezifikation (siehe [SNM+93]) für uns das Problem, daß wir nicht-konstruktive Aspekte der Spezifikation in eine konstruktive Beschreibung überführen müssen. Im Hinblick auf den Ablauf HK-Untersuchung müssen wir (zusätzlich zu den durch die Datenmodellebene bereitgestellten konstruktiven Operationen) zwei weitere Unformungen durchführen. Dies betrifft die Operationen akt_uw und gen_KonKey_Befund. Die Spezifikation dieser Operationen führen wir auf der Basisebene durch, um diese von der Spezifikation der elementaren Transaktionen zu trennen. Im folgenden geben wir einige kurze Erläuterungen zu den genannten Operationen. Man beachte, daß auch ein Definiertheitsprädikat zur Operation akt_uw spezifiziert wird.

   - $gen\_KonKey\_Befund : Db -> BefundId$
     Diese Operation hat die Aufgabe, einen neuen konkreten Schlüssel[7] auf der Grundlage eines Argumentes der Sorte Db zu berechnen. Diese Operation wird nicht von der schematischen Spezifikation der Datenmodellebene (siehe Abschnitt [Aut93]) bereitgestellt, weil sie im allgemeinen spezifischer Natur ist. Neben der schematischen Bedingung, daß ein mit dieser Operation berechneter Schlüssel in dem Argument der Sorte Db nicht vorkommt, besteht der spezifische Anteil in weiteren möglichen Anforderungen an den zu berechnenden Schlüssel, die auf der schematischen Datenmodellebene nicht formuliert werden können. In der SPECTRUM-Spezifikation versucht man dieses Problem zu lösen, indem man

[7] In der vorliegenden Spezifikation wird zwischen konkreten und abstrakten Schlüsseln unterschieden. Näheres dazu findet man in [Hec93]

Operationen höherer Ordnung zur Schlüsselgenerierung heranzieht. Als weiteres Argument, neben einem Träger der Sorte *Db*, wird diesen Operationen ein Prädikat übergeben, indem eine spezifische Anforderung an den zu bestimmenden Schlüssel formuliert werden kann. Eine solche Lösung ist in OBSCURE nicht möglich. Deshalb wird die Schlüsselgenerierung hier als ein spezifisches Problem aufgefaßt. Spezifiziert wird diese im Modul SHLUESSELGENERIERUNG (siehe Abschnitt 4.3).

- $akt\_uw\ :\ Db\ PatId\ ->\ HK\_UW$
  Diese Operation bestimmt eine noch unberabeitete HK-Überweisung im aktuellen Trägern der Sorte *Db*, die zu dem Patienten gehört, der durch das zweite Argument eindeutig identifiziert wird. In der SPECTRUM-Spezifikation wird diese Operation (die dort nicht konstruktiv ist) gemeinsam mit den atomaren Funktionen spezifiziert. Wir führen sie dagegen auf der Basisebene im Modul SONDERFUNKTIONEN (siehe Abschnitt 4.4) ein.

- $akt\_uw\_def\ :\ Db\ PatId\ ->\ bool$
  Diese Operation ist das Definiertheitsprädikat (siehe 1.6) zur Funktion akt_uw. Sie überprüft lediglich, ob für einen konkreten Patienten und eine Datenbank (Träger der Sorte *Db*) eine noch unbearbeitete HK-Überweisung existiert. Auch diese Operation wird im Modul SONDERFUNKTIONEN (siehe Abschnitt 4.4) spezifiziert.

Neben diesen drei Operationen wird auf der Basisebene (siehe Abschnitt 4.5) ein OK-Prädikat *OK* spezifiziert, mithilfe dessen sich die Erfülltheit der statischen Integritätsbedingungen für Träger der Sorte *Db* überprüfen läßt.

- *OK*
  Im Modul OK_PRAEDIKAT werden die statischen Integritätsbedingungen, die schon auf der schematischen Datenmodellebene spezifiziert wurden, angereichert um die spezifischen statischen Integritätsbedingungen im Zusammenhang mit den einzelnen Abläufe. Da wir uns hier auf den Ablauf HK-Untersuchung

konzentrieren, sind dies nur zwei zusätzliche Integritätsbedingungen. Die Konjunktion sämtlicher statischer Integritätsbedingungen bildet das OK-Prädikat OK.

Der Modul BASIS_SCHNITTSTELLE vereinigt die Importsignaturen sämtlicher Module dieser Basisebene zu einer zentralen Beschreibung der Anforderungen an die Datenmodellebene. Dieser gibt also an, welche Sorten und Operationen von der Datenmodellebene exportiert werden müssen, damit ein späteres Zusammenfügen beider Spezifikationen möglich ist. Natürlich sorgen FORGET-Konstruktionen dafür, daß die Basisebene nur die gewünschten Operationen und Sorten exportiert. Hilfsoperationen und Hilfssorten können vergessen werden.

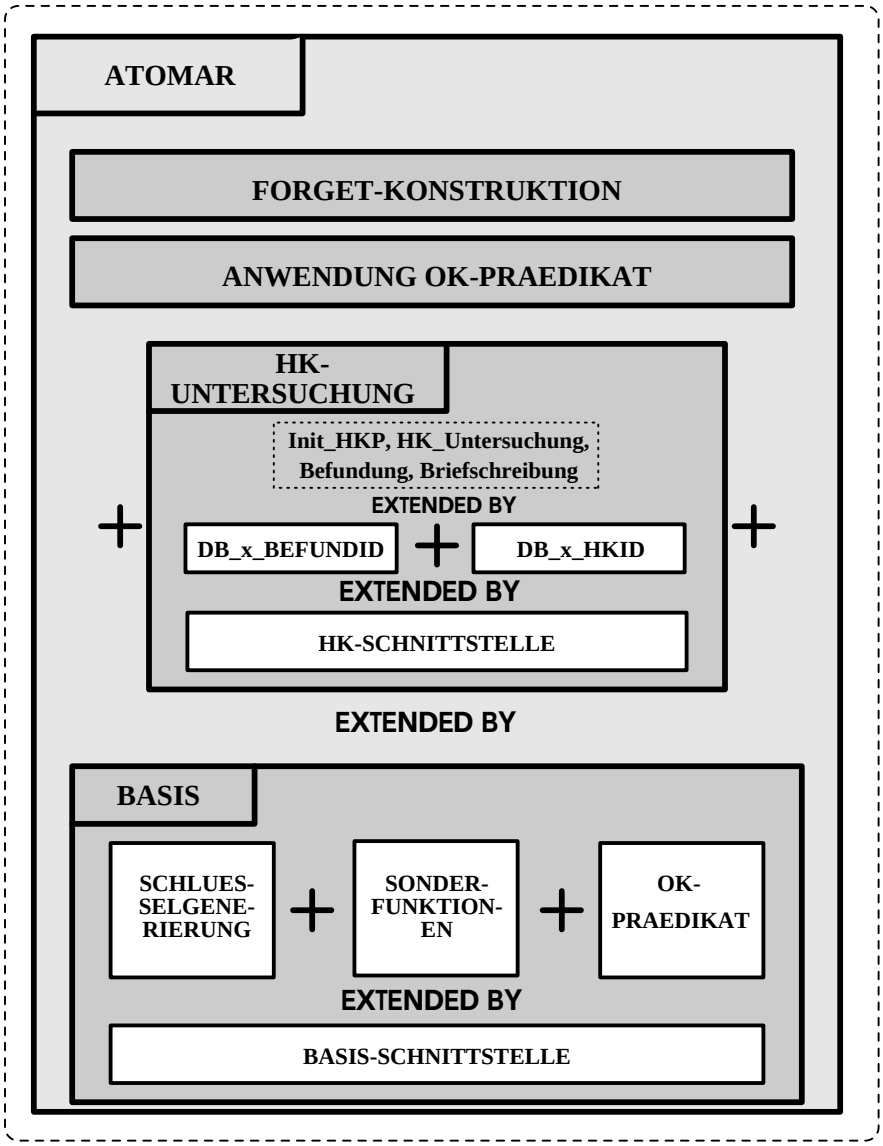
2. Der Modul HK_UNTERSUCHUNG
   Im Modul HK_UNTERSUCHUNG werden im wesentlichen die atomaren Funktionen des Ablaufs HK-Untersuchung spezifiziert. Die eigentliche Spezifikation dieser atomaren Funktionen ähnelt sehr denen der SPECTRUM-Spezifikation. Im Unterschied zu dieser enthält der Modul HK_UNTERSUCHUNG noch weitere Bestandteile, die kurz erläutern werden:
   Weil drei der vier atomaren Funktionen in ein Kreuzprodukt zweier Sorten abbilden, ist es in OBSCURE notwendig, eine Hilfssorte für jede dieser Kreuzproduktkonstellationen zu spezifizieren. Dies geschieht in den Moduln DB_x_BEFUNDID und DB_x_HKID (siehe 3.3.1).
   Außerdem beschreibt der Modul HK_SCHNITTSTELLE - analog zum Modul BASIS_SCHNITTSTELLE der Basisebene - die vereinigte Importsignatur sämtlicher Module der Spezifikation HK_UNTERSUCHUNG.

3. Anwendung des OK-Prädikats
   Die atomaren Funktionen sollen den Integritätsbedingungen genügen. Dieser Anforderung an die Spezifikation verleihen wir Nachdruck durch eine explizite Zusicherung der Gültigkeit der Integritätsbedingungen (für jede einzelne atomare Funktion) in einem Exportaxiom. Falls die Spezifikationen der atomaren Funktionen diesen Zusicherungen nicht genügen, so hat die OBSCURE-Spezifikation kein Modell. Damit ergibt

sich der Nachweis der Gültigkeit dieser Exportaxiome als wichtige Verifikationsaufgabe.

4. FORGET-Konstruktion
   Außer den atomaren Funktionen (und den damit verbundenen Sorten) werden sämtliche Hilfssorten und Hilfsoperationen vergessen.

Die folgende Darstellung erläutert die Struktur der Gesamtspezifikation.



3 Spezifikation des Ablaufs HK-Untersuchung

Ausgangspunkt von HDMS-A war eine Anforderungsanalyse des zu spezifizierenden Systems (siehe hierzu [CKL93]). Die Ergebnisse dieser Anforderungsanalyse wurden mithilfe geeigneter Beschreibungsmittel festgehalten. Die eingesetzten Mittel bestehen aus der E/R-Modellierungstechnik in Kombination mit Datenflußdiagrammen. Weil diese in unserem Zusammenhang auf einer Stufe zwischen informaler Analysetätigkeit und formaler Spezifikation als Hilfsmittel eingesetzt werden, werden sie im folgenden als semiformale Beschreibungsmittel bezeichnet, ohne dabei von dem formalen Aspekt der E/R-Diagramme ablenken zu wollen.
Der erste Unterabschnitt stellt nun kurz die semiformale Beschreibungen zum Auschnitt HK-Untersuchung des Gesamtsystems vor, während die weiteren Unterabschnitte dann auf die konkrete Spezifikation dieses Ausschnitts in OBSCURE eingehen.

3.1 Semiformale Beschreibung

Das folgende E/R-Diagramm beschreibt den zum Ablauf HK-Untersuchung relevanten Ausschnitt des gesamten HDMS-A Datenmodells.

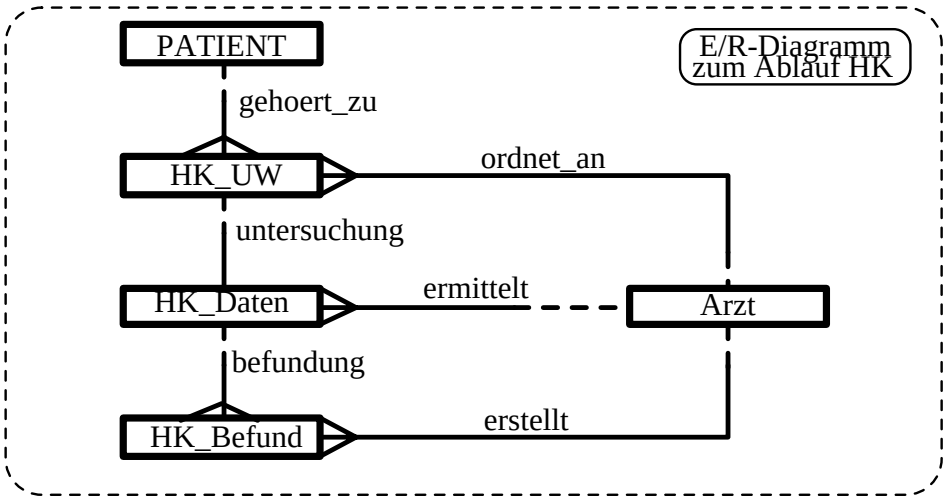


Abbildung2: E/R-Diagramm zum Ablauf HK

Die Attribute zu den einzelnen Entitytypen sind der folgenden Übersicht zu entnehmen.

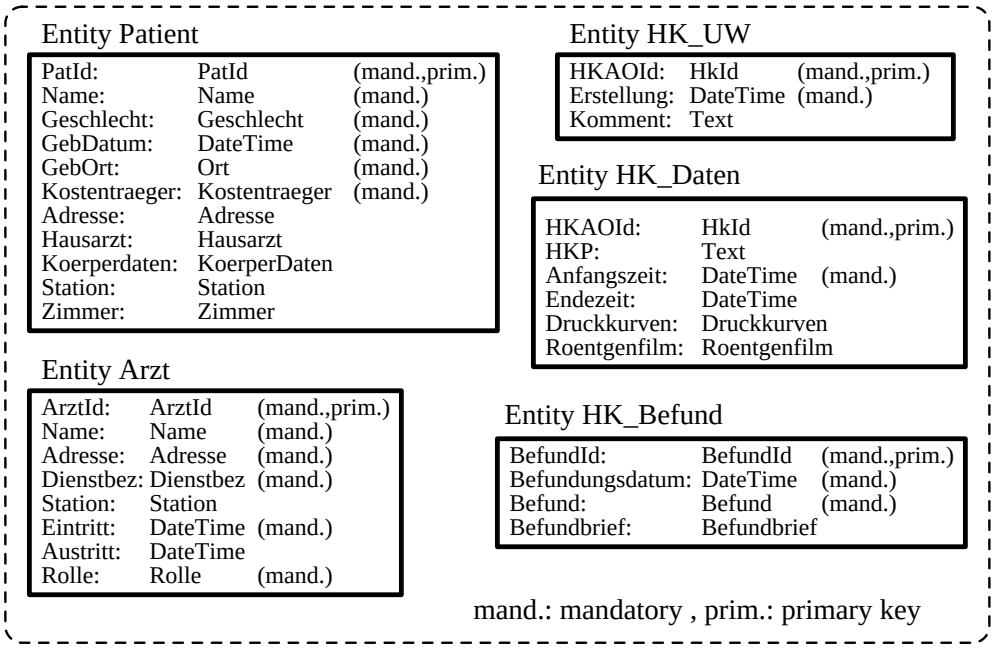


Abbildung3: Die Attribute der Entities zum Ablauf HK

Diese beiden Diagramme beschreiben das Datenmodell, das der folgenden Spezifikation zugrunde liegt. In Abbildung 3 sind insbesondere die Zusätze "mandatory" und "primary key" zu beachten. Hat ein Attribut einen Zusatz "mandatory" so wird gefordert, daß ein solches Attribut niemals den leeren Eintrag haben kann (siehe Stichwort "leere Attribute" in [Hec93]). Durch den Zusatz "primary key" wird gefordert, daß ein Attributeintrag die Schlüsseleigenschaft erfüllt. Diese beiden Forderungen sind zu den statischen Integritätsbedingungen zur Spezifikation zu zählen und werden später bei der Formulierung des OK-Prädikats (siehe Abschnitt 4.5) berücksichtigt. Das folgende Datenflußdiagramm beschreibt nun den zur Spezifikation ausgewählten Ablauf HK-Untersuchung.

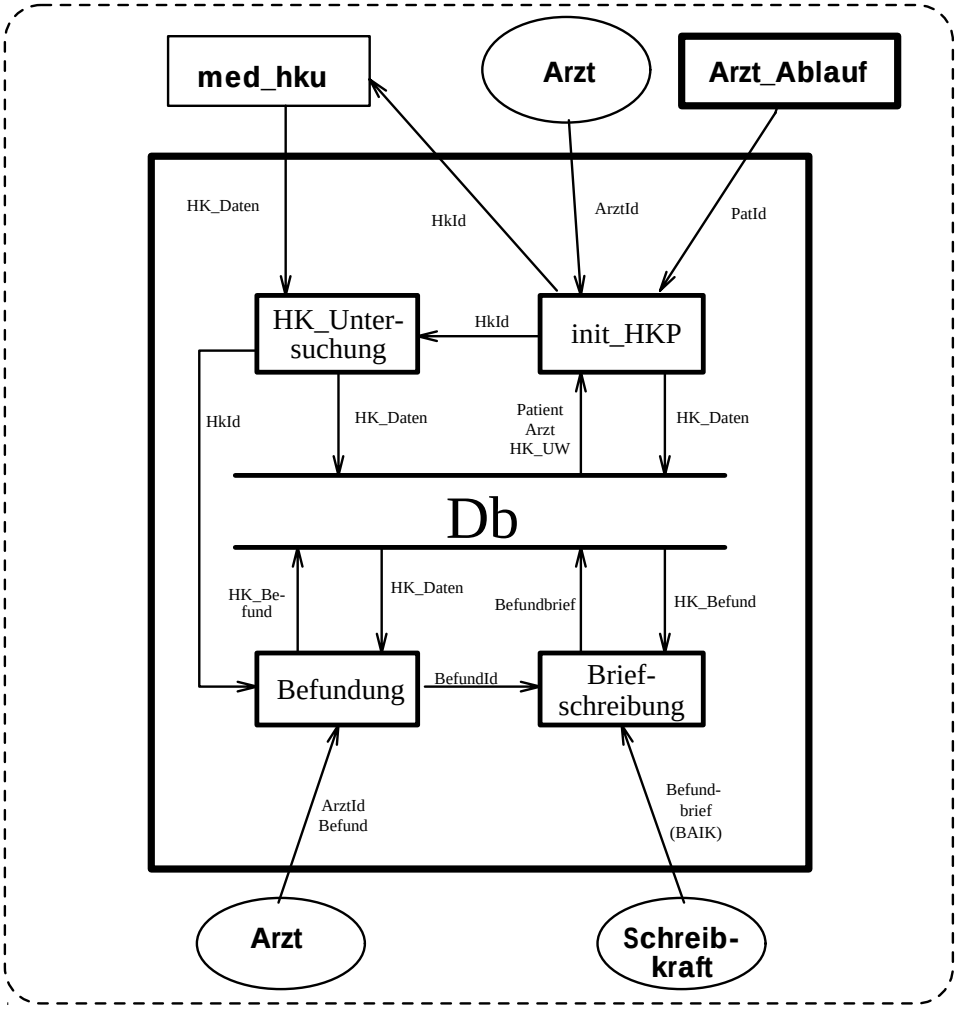


Abbildung4: DFD-Diagramm zum Ablauf HK

Anmerkung: Die Diagramme sind identisch zu denen der Münchner Spezifikationsvorlage (siehe [SNM+93]).

3.2 Der Schnittstellenmodul HK_SCHNITTSTELLE

Die in diesem Modul genannten Sorten und Operationen beschreiben die Schnittstelle der Spezifikation des Ablaufs HK-Untersuchung zur Spezifikation der Datenbank. Diese Information wollen wir durch eine große Importliste explizit in unsere Spezifikation aufnehmen. Generiert wird in diesem Modul nichts.
Zuerst werden die benötigten Importsorten aufgeführt.

(HK_SCHNITTSTELLE.obs 1) ≡
IMPORTS
SORTS
Db
Arzt Patient HK_Befund HK_UW
ArztId DateTime HkId HK_Daten
PatId BefundId Text Befundbrief
AttrArztId AttrDateTime
AttrDruckkurven AttrRoentgenfilm
AttrPatId AttrBefundId AttrText
AttrBefund AttrBefundbrief AttrHkId
AKeyHK_Daten AKeyHK_Befund
AKeyPatient AKeyArzt AKeyHK_UW
Druckkurven Roentgenfilm

Siehe auch owf–moduln 2, 3, 4, 5, 6, 7, 8, 9, und 10.

Die folgenden *is_in*...-Operationen überprüfen, ob Entities (diese werden durch abstrakte Schlüssel identifiziert) in einem Träger der Sorte *Db* vorhanden sind. Der Zugriff auf Entities in einem Träger der Sorte *Db* erfolgt mit den *get*...-Operationen.

(HK_SCHNITTSTELLE.obs 1)+ ≡
OPNS
is_inPatient: AKeyPatient Db –> bool
is_inArzt: AKeyArzt Db –> bool
is_inHK_Daten: AKeyHK_Daten Db –> bool
is_inHK_Befund: AKeyHK_Befund Db –> bool
getPatient: AKeyPatient Db –> Patient
getArzt: AKeyArzt Db –> Arzt
getHK_Daten: AKeyHK_Daten Db –> HK_Daten
getHK_Befund: AKeyHK_Befund Db –> HK_Befund

Mithilfe der *codeKonKey_...*-Operationen werden aus konkreten Schlüsseln abstrakte berechnet.

```
(HK_SCHNITTSTELLE.obs 1)+ ≡
        codeKonKey_HK_Daten: HkId -> AKeyHK_Daten
        codeKonKey_HK_Befund: BefundId -> AKeyHK_Befund
        codeKonKey_Patient: PatId -> AKeyPatient
        codeKonKey_Arzt: ArztId -> AKeyArzt
        codeKonKey_HK_UW: HkId -> AKeyHK_UW
```

Das Kreieren neuer Objekte eines bestimmten Entitytyps erfolgt mittels der *create_...*-Operationen, während die *update*- und *put...*-Operationen den schreibenden Zugriff auf einen Träger der Sorte *Db* realisieren.

```
(HK_SCHNITTSTELLE.obs 1)+ ≡
        createHK_Daten: AttrHkId AttrText AttrDateTime AttrDateTime
                AttrDruckkurven AttrRoentgenfilm -> HK_Daten
        createHK_Befund: AttrBefundId AttrDateTime AttrBefund
                AttrBefundbrief -> HK_Befund
        updateHK_Daten: HK_Daten Db -> Db
        updateHK_Befund: HK_Befund Db -> Db
        putHK_Daten: HK_Daten Db -> Db
        putHK_Befund: HK_Befund Db -> Db
```

Die *est...*-Operationen realisieren den schreibenden Zugriff auf Relationships.

```
(HK_SCHNITTSTELLE.obs 1)+ ≡
        estermittelt: Db Arzt HK_Daten -> Db
        estuntersuchung: Db HK_UW HK_Daten -> Db
        esterstellt: Db Arzt HK_Befund -> Db
        estbefundung: Db HK_Daten HK_Befund -> Db
```

Zu jeder Attributsorte existiert eine *UNDEF_...*-Konstante, die den leeren Attributeintrag repräsentiert.

```
(HK_SCHNITTSTELLE.obs 1)+ ≡
        UNDEF_Text: -> AttrText
        UNDEF_Druckkurven: -> AttrDruckkurven
        UNDEF_Roentgenfilm: -> AttrRoentgenfilm
        UNDEF_DateTime: -> AttrDateTime
        UNDEF_Befundbrief: -> AttrBefundbrief
```

Mittels der *attr*-Operationen erfolgt die Erweiterung der *Domainsorten* um die $UNDEF\_...$-Konstanten zur Einführung der Attributsorten, während die *rtta*-Operationen den umgekehrten Schritt beschreiben. Außerdem werden Gleichheitsoperationen für die Sorten $AttrDateTime$ und $AttrBefundbrief$ benötigt.

```
(HK_SCHNITTSTELLE.obs 1)+ ≡
        attr: DateTime –> AttrDateTime
        attr: Text –> AttrText
        attr: Druckkurven –> AttrDruckkurven
        attr: Roentgenfilm –> AttrRoentgenfilm
        attr: BefundId –> AttrBefundId
        attr: HK_Befund –> AttrBefund
        attr: Befundbrief –> AttrBefundbrief
        attr: HkId –> AttrHkId
        rtta: AttrHkId –> HkId
        _ = _: AttrDateTime AttrDateTime –> bool
        _ = _: AttrBefundbrief AttrBefundbrief –> bool
```

Die folgenden Operationen realisieren den Zugriff auf Attributeinträge von Entities.

```
(HK_SCHNITTSTELLE.obs 1)+ ≡
        Endezeit: HK_Daten –> AttrDateTime
        HKAOId: HK_Daten –> AttrHkId
        HKAOId: HK_UW –> AttrHkId
        Untersuchungsdatum: HK_Daten –> AttrDateTime
        Anfangszeit: HK_Daten –> AttrDateTime
        Befundbrief: HK_Befund –> AttrBefundbrief
        Befundungsdatum: HK_Befund –> AttrDateTime
        Befund: HK_Befund –> AttrBefund
```

Die folgende Operation *med_hku* wird nicht von der Datenmodellebene bereitgestellt. Sie ist eine Umweltfunktion, die eine Schnittstelle zu externen Einheiten beschreibt. Innerhalb der vorliegenden Spezifikation wird diese nicht modelliert und muß deshalb an dieser Stelle importiert werden. Als Argument erhält *med_hku* einen Träger der Schlüsselattributsorte $HkId$ und liefert eine Entity des Typs HK_Daten zurück. Diese Entity enthält unter anderem Röntgen- und Druckdaten, die von externen Einheiten bestimmt wurden.

```
(HK_SCHNITTSTELLE.obs 1)+ ≡
        med_hku : HkId –> HK_Daten
```

Die folgenden, mit "Basisebene" gekennzeichneten, Operationen beschreiben die Schnittstelle zur Basisebene, daß heißt, diese Operationen werden nicht von der Datenmodelleben bereitgestellt, sondern in der Basisebene spezifiziert (siehe Abschnitt 4.1). Aus technischen Gründen müssen diese hier mit aufgelistet werden, damit der Modul HK_UNTERSUCHUNG syntaktisch korrekt ist.

```
(HK_SCHNITTSTELLE.obs 1)+ ≡
        gen_KonKey_Befund : Db –> BefundId     ## Basisebene
        akt_uw : Db PatId –> HK_UW             ## Basisebene
        akt_uw_def : Db PatId –> bool          ## Basisebene
        OK: Db –> bool                         ## Basisebene
```

Anmerkung: Nähere Angaben zu den hier aus der Datenmodellebene importierten Operationen findet man in [Aut93].

3.3 Spezifikation der elementaren Transaktionen

Der Ablauf HK-Untersuchung setzt sich zusammen aus den vier elementaren Transaktionen *init_HKP*, *HK_Untersuchung*, *Befundung* und *Briefschreibung*. Jede elementare Transaktion wird - analog zur SPECTRUM-Spezifikation - spezifiziert durch eine OBSCURE-Operation gleichen Namens. Die in der Einleitung (siehe Abschnitt 1) aufgezeigten Probleme, die aus der Verwendung der algorithmischen Spezifikationsmethode resultieren, führen zu den prinzipiellen Änderungen im Vergleich zur SPECTRUM-Spezifikation. Wie in der Einleitung beschrieben, ist es zunächst notwendig, für die Wertebereiche der Operationen *init_HKP*, *HK_Untersuchung* und *Befundung* - nämlich die Kreuzprodukte $Db\_ \times \_HkId$ und $Db\_ \times \_BefundId$ - entsprechende Hilfssorten einzuführen. Diese Generierung der Hilfssorten wird in Abschnitt 3.3.1 beschrieben. In den restlichen Unterabschnitten werden dann die einzelnen elementaren Transaktionen spezifiziert.

3.3.1 Spezifikation der Kreuzprodukt-Hilfssorten

Mittels eines parametrisierten Aufrufs des OBSCURE-Standardmoduls MK_PAIR angewendet auf die Sorten *Db* und *HkId*, bzw. *Db* und *BefundId* werden die Kreuzproduktsorten $Db\_ \times \_HkId$ und $Db\_ \times \_BefundId$ eingeführt. Zusätzlich werden entsprechende Zugriffsoperationen spezifiziert.

```
(DB_x_BEFUNDID.obs 11) ≡
```

INCLUDE MK_PAIR(**SORTS** Db BefundId)
[**SORTS** Db_x_BefundId
OPNS {_/_}, get_fst, get_snd, set_fst, set_snd]
(DB_x_HKID.obs 12) ≡
INCLUDE MK_PAIR(**SORTS** Db HkId)
[**SORTS** Db_x_HkId
OPNS {_/_}, get_fst, get_snd, set_fst, set_snd]

3.3.2 Spezifikation der Operationen *init_HKP*, *HK_Untersuchung*, *Befundung* und *Briefschreibung*

In der Einleitung wurde bereits darauf hingewiesen, daß wir zu jeder elementaren Transaktion ein Definiertheitsprädikat einführen. Die Definiertheitsprädikate werden gemeinsam mit den atomaren Funktionen auf dieser Ebene spezifiziert:

⟨ zu_kreierende_Operationen 13 ⟩ ≡
init_HKP_def : Db PatId ArztId –> bool,
init_HKP : Db PatId ArztId DateTime –> Db_x_HkId,
HK_Untersuchung_def : Db HkId –> bool,
HK_Untersuchung : Db HkId HK_Daten –> Db_x_HkId,
Befundung_def : Db HkId ArztId –> bool,
Befundung : Db HkId ArztId HK_Befund DateTime –> Db_x_BefundId,
Briefschreibung_def: Db BefundId –> bool,
Briefschreibung: Db BefundId Befundbrief –> Db

Dieser Code wird verwendet in owf–modul 23.

Nach der Angabe der Stelligkeiten wollen wir nun den konkreten Spezifikationstext dieser Operationen angeben. Zur besseren Lesbarkeit geben wir zuvor noch eine Übersicht der verwendeten Variablensymbole:

⟨ Variablendefinition 14 ⟩ ≡
db : Db
uw : HK_UW
pi : PatId
ai : ArztId
arzt : Arzt
hkd, hkd': HK_Daten
x : Db_x_HkId
y : Db_x_BefundId
hkid: HkId

```
bef, bef', b: HK_Befund
bi : BefundId
brief: Befundbrief
dt : DateTime
```

Dieser Code wird verwendet in owf–modul 23.

3.3.3 Die elementare Transaktion *init_HKP*

Zunächst spezifizieren wir die zugehörige boolsche Operation $init\_HKP\_def$. Die elementare Transaktion $init\_HKP$ läuft genau dann fehlerfrei ab, falls es in der Datenbank den Schlüsseln $codeKonKey\_Patient(pi)$ und $codeKonKey\_Arzt(ai)$ zugeordnete Objekte der Sorte $Patient$ bzw. $Arzt$ gibt und die HK_Überweisung uw zu dem durch pi beschriebenen Patienten gehört und ferner uw in der Datenbank noch zu keinen Untersuchungsdaten in Verbindung steht. Die beiden letztgenannten Bedingungen werden durch die Operation akt_uw_def überprüft, welche auf der Basisebene eingeführt wird (siehe Abschnitt 4.4) .

⟨ Definiertheit_zu_init_HKP 15 ⟩ ≡

```
init_HKP_def(db,pi,ai)
    <- is_inPatient(codeKonKey_Patient(pi),db)
       and
       is_inArzt(codeKonKey_Arzt(ai),db)
       and
       akt_uw_def(db,pi);
```

Dieser Code wird verwendet in owf–modul 23.

Die elementare Transaktion $init\_HKP$ kreiert, falls sie fehlerfrei abläuft, ein Objekt der Sorte HK_Daten (dabei werden die notwendigen Attribute entsprechend belegt), legt dieses in der Datenbank ab und stellt die Verbindungen zu der zugeordneten HK_Überweisung und dem ermittelnden Arzt her.

⟨ Atomare_Funktion_init_HKP 16 ⟩ ≡

```
init_HKP(db,pi,ai,dt)
    <- IF init_HKP_def(db,pi,ai)
       THEN LET uw : akt_uw(db,pi),
                 arzt : getArzt(codeKonKey_Arzt(ai),db)
            IN
            LET hkd : createHK_Daten(
                          attr(rtta(HKAOId(uw))),
                          UNDEF_Text,
                          attr(dt),
                          UNDEF_DateTime,
                          UNDEF_Druckkurven,
                          UNDEF_Roentgenfilm)
            IN {estermittelt(
                   estuntersuchung(
                      putHK_Daten(hkd,db),
                      uw,hkd),
                   arzt,hkd)
                /(rtta(HKAOId(uw)))}
            TEL
            TEL
       ELSE ERROR(Db_x_HkId)
       FI;
```

Dieser Code wird verwendet in owf–modul 23.

3.3.4 Die elementare Transaktion *HK_Untersuchung*

Das Definiertheitsprädikat zur atomaren Funktion *HK_Untersuchung* überprüft, ob es zum abstrakten Schlüssel *codeKonKey_Hk_Daten(hkid)* auch ein zugeordnetes Objekt der Sorte *HK_Daten* in der Datenbank gibt und ob dieses Objekt unter dem Attribut Endezeit den leeren Attributeintrag (*UNDEF_Datetime*) aufweist.

⟨ Definiertheit_zu_HK_Untersuchung 17 ⟩ ≡

```
HK_Untersuchung_def(db,hkid)
    <- IF is_inHK_Daten(codeKonKey_HK_Daten(hkid),db)
       THEN Endezeit(getHK_Daten(
                codeKonKey_HK_Daten(hkid),db))
            =
            UNDEF_DateTime
       ELSE false
       FI;
```

Dieser Code wird verwendet in owf–modul 23.

Im Fall der Fehlerfreiheit überschreibt die elementare Transaktion *HK_Untersuchung* die zu *codeKonKey_Hk_Daten(hkid)* gehörende Entity der Sorte *HK_Daten*, durch eine, mittels der importierten Operation *med_hku* ermittelte, Entity der Sorte *HK_Daten* (diese enthält insbesondere nichtleere Einträge zu den Attributen *Roentgenfilm*, *Druckkurven* und *HKP*).

```
⟨ Atomare_Funktion_HK_Untersuchung 18 ⟩ ≡
        HK_Untersuchung(db,hkid,hkd)
              <– IF HK_Untersuchung_def(db,hkid)
                 THEN {updateHK_Daten(hkd,db)/hkid}
                 ELSE ERROR(Db_x_HkId)
                 FI;
```

Dieser Code wird verwendet in owf–modul 23.

3.3.5 Die elementare Transaktion *Befundung*

Genau dann, wenn es zu den Schlüsseln *codeKonKey_Hk_Daten(hkid)* und *codeKonKey_Arzt(ai)* zugeordnete Objekte der Sorten *HK_Daten* und *Arzt* in der Datenbank gibt und ferner das über *codeKonKey_Hk_Daten(hkid)* referenzierte Objekt der Sorte *HK_Daten* den leeren Eintrag (*UNDEF_Zeit*) unter dem Attribut *Endezeit* hat , läuft die elementare Transaktion *Befundung* fehlerfrei ab.

```
⟨ Definiertheit_zu_Befundung 19 ⟩ ≡
        Befundung_def(db,hkid,ai)
              <– IF is_inHK_Daten(codeKonKey_HK_Daten(hkid),db)
                    and
                    is_inArzt(codeKonKey_Arzt(ai),db)
                 THEN not (Endezeit(getHK_Daten(
                                       codeKonKey_HK_Daten(hkid),db))
                                =
                                UNDEF_DateTime)
                 ELSE false
                 FI;
```

Dieser Code wird verwendet in owf–modul 23.

Die Operation *Befundung* kreiert, falls sie fehlerfrei abläuft, ein Objekt der Sorte *HK_Befund* und schreibt dieses in die Datenbank und stellt die Verbindung zu dem durch *codeKonKey_Hk_Daten(hkid)* beschriebenen Objekt

der Sorte HK_Daten her. Man beachte, daß beim Kreieren eines Objektes der Sorte HK_Befund, ein neuer eindeutiger Schlüssel durch die Operation $gen\_KonKey\_HK\_Befund$ ermittelt wird.

⟨ Atomare_Funktion_Befundung 20 ⟩ ≡

```
Befundung(db,hkid,ai,b,dt)
        <– IF Befundung_def(db,hkid,ai)
           THEN LET hkd : getHK_Daten(
                                codeKonKey_HK_Daten(hkid),db) ,
                       arzt: getArzt(codeKonKey_Arzt(ai),db) ,
                       bi  : gen_KonKey_Befund(db)
                IN LET bef : createHK_Befund(
                                attr(bi),
                                attr(dt),
                                attr(b),
                                UNDEF_Befundbrief)
                   IN {esterstellt(
                            estbefundung(
                               putHK_Befund(bef,db),
                               hkd,bef),
                            arzt,bef)
                       /bi}
                   TEL
                TEL
           ELSE ERROR(Db_x_BefundId)
           FI;
```

Dieser Code wird verwendet in owf–modul 23.

3.3.6 Die elementare Transaktion *Briefschreibung*

Die elementare Transaktion $Briefschreibung$ ist genau dann fehlerfrei ab, wenn es zum Schlüssel $codeKonKey\_Hk\_Befund(bi)$ ein zugehöriges Objekt der Sorte HK_Befund in der Datenbank gibt und ferner dieses Objekt HK_Befund noch keinen Eintrag (außer $UNDEF\_Befundbrief$) unter dem Attribut $Befundbrief$ hat.

⟨ Definiertheit_zu_Briefschreibung 21 ⟩ ≡

```
Briefschreibung_def(db,bi)
        <– IF is_inHK_Befund(codeKonKey_HK_Befund(bi),db)
           THEN Befundbrief(getHK_Befund(
                              codeKonKey_HK_Befund(bi),db))
                =
                UNDEF_Befundbrief
           ELSE false
           FI;
```

Dieser Code wird verwendet in owf–modul 23.

Läuft die Operation *Briefschreibung* fehlerfrei ab, so wird das über $codeKonKey\_Befund(bi)$ referenzierte Objekt der Sorte HK_Befund überschrieben, indem das Attribut der Sorte $Befundbrief$ durch das übergebene Argument der Sorte $Brief$ belegt wird.

⟨ Atomare_Funktion_Briefschreibung 22 ⟩ ≡

```
Briefschreibung(db,bi,brief)
        <– IF Briefschreibung_def(db,bi)
           THEN LET bef: getHK_Befund(
                                codeKonKey_HK_Befund(bi),db)
                IN LET bef': createHK_Befund(
                                        attr(bi),
                                        Befundungsdatum(bef),
                                        Befund(bef),
                                        attr(brief))
                     IN updateHK_Befund(bef',db)
                     TEL
                TEL
           ELSE ERROR(Db)
           FI;
```

Dieser Code wird verwendet in owf–modul 23.

Der HK_Befund und der Befundbrief werden durch den Arzt erstellt. Damit dies möglich ist, benötigt der Arzt (oder eine Schreibkraft) lesenden Zugriff auf die HK_Daten und den HK_Befund. Dieser Zugriff kann mittels der hier (aus der Datenbankebene) importierten Zugriffsfunktionen $getHK\_Daten$ und $getHK\_Befund$ erfolgen. Deshalb werden keine speziellen Funktionen eingeführt.

3.4 Der zusammengesetzte Modul HK_UNTERSUCHUNG

Die Gesamtspezifikation des Ablaufs HK-Untersuchung setzt sich aus den zuvor beschriebenen Bestandteilen zusammen, indem sie den Schnittstellenmodul HK_SCHNITTSTELLE erweitert um die Spezifikation der Kreuzprodukt-Hilfssorten und den so erhaltenen Teil durch die Spezifikation der atomaren Funktionen samt ihrer Definiertheitsüberprüfungen erweitert.

(HK_UNTERSUCHUNG.obs 23) ≡
(**INCLUDE** HK_SCHNITTSTELLE
EXTENDED BY
INCLUDE DB_x_BEFUNDID **PLUS INCLUDE** DB_x_HKID
ENDEXTENDED)
EXTENDED BY
CREATE
OPNS
⟨ zu_kreierende_Operationen 13 ⟩
SEMANTICS
VARS
⟨ Variablendefinition 14 ⟩
PROGRAMS
⟨ Definiertheit_zu_init_HKP 15 ⟩
⟨ Atomare_Funktion_init_HKP 16 ⟩
⟨ Definiertheit_zu_HK_Untersuchung 17 ⟩
⟨ Atomare_Funktion_HK_Untersuchung 18 ⟩
⟨ Definiertheit_zu_Befundung 19 ⟩
⟨ Atomare_Funktion_Befundung 20 ⟩
⟨ Definiertheit_zu_Briefschreibung 21 ⟩
⟨ Atomare_Funktion_Briefschreibung 22 ⟩
ENDCREATE
ENDEXTENDED

3.5 Die Signatur zum Modul HK_UNTERSUCHUNG

Die folgende Auflistung der Signatur zum Modul HK_UNTERSUCHUNG wurde mit dem OBSCURE-System automatisch erzeugt.

```
File name: "HK_UNTERSUCHUNG.T"
```

```
list of sorts and operations:

imported and exported (inherited).
 SORTS
  Db BefundId AttrBefundbrief AttrDateTime AKeyHK_Befund
  AttrHkId AttrBefund AttrBefundId AttrPatId AttrArztId
  ArztId AKeyHK_Daten AttrDruckkurven AttrRoentgenfilm
  DateTime HK_Daten AKeyHK_UW Patient Roentgenfilm Text
  Druckkurven HkId HK_Befund AKeyArzt AKeyPatient AttrText
  Arzt Befundbrief PatId HK_UW
 OPNS
  esterstellt : Db Arzt HK_Befund -> Db
  estermittelt : Db Arzt HK_Daten -> Db
  estbefundung : Db HK_Daten HK_Befund -> Db
  estuntersuchung : Db HK_UW HK_Daten -> Db
  putHK_Daten : HK_Daten Db -> Db
  updateHK_Befund : HK_Befund Db -> Db
  putHK_Befund : HK_Befund Db -> Db
  updateHK_Daten : HK_Daten Db -> Db
  gen_KonKey_Befund : Db -> BefundId
  attr : Befundbrief -> AttrBefundbrief
  UNDEF_Befundbrief : -> AttrBefundbrief
  Befundbrief : HK_Befund -> AttrBefundbrief
  UNDEF_DateTime : -> AttrDateTime
  Untersuchungsdatum : HK_Daten -> AttrDateTime
  attr : DateTime -> AttrDateTime
  Anfangszeit : HK_Daten -> AttrDateTime
  Endezeit : HK_Daten -> AttrDateTime
  Befundungsdatum : HK_Befund -> AttrDateTime
  codeKonKey_HK_Befund : BefundId -> AKeyHK_Befund
  HKAOId : HK_Daten -> AttrHkId
  HKAOId : HK_UW -> AttrHkId
  attr : HkId -> AttrHkId
  Befund : HK_Befund -> AttrBefund
  attr : HK_Befund -> AttrBefund
  attr : BefundId -> AttrBefundId
  codeKonKey_HK_Daten : HkId -> AKeyHK_Daten
  attr : Druckkurven -> AttrDruckkurven
  UNDEF_Druckkurven : -> AttrDruckkurven
  UNDEF_Roentgenfilm : -> AttrRoentgenfilm
```

```
  attr : Roentgenfilm -> AttrRoentgenfilm
  getHK_Daten : AKeyHK_Daten Db -> HK_Daten
  med_hku : HkId -> HK_Daten
  createHK_Daten : AttrHkId AttrText AttrDateTime
  AttrDateTime AttrDruckkurven AttrRoentgenfilm -> HK_Daten
  codeKonKey_HK_UW : HkId -> AKeyHK_UW
  is_inArzt : AKeyArzt Db -> bool
  is_inPatient : AKeyPatient Db -> bool
  is_inHK_Befund : AKeyHK_Befund Db -> bool
  is_inHK_Daten : AKeyHK_Daten Db -> bool
  akt_uw_def : Db PatId -> bool
  OK : Db -> bool
  _ = _ : AttrBefundbrief AttrBefundbrief -> bool
  _ = _ : AttrDateTime AttrDateTime -> bool
  getPatient : AKeyPatient Db -> Patient
  rtta : AttrHkId -> HkId
  createHK_Befund : AttrBefundId AttrDateTime AttrBefund
  AttrBefundbrief -> HK_Befund
  getHK_Befund : AKeyHK_Befund Db -> HK_Befund
  codeKonKey_Arzt : ArztId -> AKeyArzt
  codeKonKey_Patient : PatId -> AKeyPatient
  attr : Text -> AttrText
  UNDEF_Text : -> AttrText
  getArzt : AKeyArzt Db -> Arzt
  akt_uw : Db PatId -> HK_UW

not imported and exported (created).
 SORTS
  Db_x_HkId Db_x_BefundId
 OPNS
  get_fst : Db_x_HkId -> Db
  get_fst : Db_x_BefundId -> Db
  Briefschreibung : Db BefundId Befundbrief -> Db
  get_snd : Db_x_BefundId -> BefundId
  HK_Untersuchung_def : Db HkId -> bool
  init_HKP_def : Db PatId ArztId -> bool
  Briefschreibung_def : Db BefundId -> bool
  Befundung_def : Db HkId ArztId -> bool
  init_HKP : Db PatId ArztId DateTime -> Db_x_HkId
  set_snd : HkId Db_x_HkId -> Db_x_HkId
```

```
  set_fst : Db Db_x_HkId -> Db_x_HkId
  HK_Untersuchung : Db HkId HK_Daten -> Db_x_HkId
  { _ / _ } : Db HkId -> Db_x_HkId
  set_snd : BefundId Db_x_BefundId -> Db_x_BefundId
  set_fst : Db Db_x_BefundId -> Db_x_BefundId
  Befundung : Db HkId ArztId HK_Befund DateTime ->
  Db_x_BefundId
  { _ / _ } : Db BefundId -> Db_x_BefundId
  get_snd : Db_x_HkId -> HkId

imported and not exported (hidden).
  ## no sorts
  ## no operations

list of constructors:

imported

exported
  { _ / _ } : Db BefundId -> Db_x_BefundId
  { _ / _ } : Db HkId -> Db_x_HkId

list of parameters:

imported.
 ## no sorts
 ## no operations

exported.
 ## no sorts
 ## no operations
## The End
```

4 Spezifikation der Basisebene

4.1 Die zusammengesetzte Basisebene

Die Basisebene spaltet sich auf in die drei Einzelmodule SCHLUESSELGENERIERUNG, SONDERFUNKTIONEN und OK_PRAEDIKAT, sowie einen zusätzlichen Modul BASIS_SCHNITTSTELLE, der die Schnittstelle dieser drei Module zur Datenmodellebene beschreibt. Damit ergibt sich die Spezifikation der Basisebene wie folgt:

(BASIS.obs 24) ≡
(**INCLUDE** SCHLUESSELGENERIERUNG
PLUS
INCLUDE SONDERFUNKTIONEN
PLUS
INCLUDE OK_PRAEDIKAT)
COMPOSE
INCLUDE BASIS_SCHNITTSTELLE

4.2 Der Modul BASIS_SCHNITTSTELLE

Der Modul BASIS_SCHNITTSTELLE beschreibt, analog zum Modul HK_SCHNITTSTELLE, die Anforderungen der Basisebene an die Datenmodellebene (siehe [Aut93]). Das heißt, daß hier die Importe der drei Einzelmodule der Basisebene zu einer gemeinsamen Importliste zusammengefaßt werden. Die importierten Sorten werden zuerst aufgelistet.

(BASIS_SCHNITTSTELLE.obs 25) ≡
IMPORTS
SORTS Db SetOfHK_Befund SetOfBefundId
HK_Befund DB_Entities
BefundId AttrBefundId
PatId HkId HK_UW SetOfHK_UW
AKeyPatientAKeyHK_UW
AttrHkId AttrPatId
DB_Relationships Untersuchung UntersuchungsPaar
SetOfPatient AKeyHK_Daten HK_Daten
GehoertZu SetOfAKeyHK_UW Patient Arzt

Siehe auch owf–moduln 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, und 40.

Die beiden folgenden Operationen extrahieren aus einem Träger der Datenbanksorte *Db* die Entitieskomponente bzw. die Relationshipskomponente.

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
OPNS
        getDB_Relationships: Db -> DB_Relationships
        getDB_Entities : Db -> DB_Entities
```

Die folgenden *get*...-Operationen realisieren den Zugriff auf die aktuelle Menge von Objekten eines bestimmten Entitys- oder Relationshiptyp in der entsprechenden Komponente der Datenbank.

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
        getSetOfHK_Befund : DB_Entities -> SetOfHK_Befund
        getSetOfHK_UW: DB_Entities -> SetOfHK_UW
        getSetOfPatient: DB_Entities -> SetOfPatient
        getuntersuchung: DB_Relationships -> Untersuchung
        getgehoert_zu: DB_Relationships -> GehoertZu
```

Weiter benötigt werden die folgenden *empty*...-Konstanten einzelner Mengensorten[8].

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
        emptySetOfPatient: -> SetOfPatient
        emptySetOfHK_Befund : -> SetOfHK_Befund
        emptySetOfBefundId : -> SetOfBefundId
        emptySetOfHK_UW: -> SetOfHK_UW
        emptySetOfAuwi: -> SetOfAKeyHK_UW
        emptyuntersuchung: -> Untersuchung
```

Für verschiedene Gleichheitsabfragen müssen wir die entsprechenden Gleicheitsoperationen importieren.

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
        _ = _ : SetOfHK_Befund SetOfHK_Befund -> bool
```

[8]Unter einer Mengensorte verstehen wir normalerweise eine durch Instantiierung des OBSCURE-Standardmoduls MK_SET mit einer konkreten Elementsorte *El* spezifizierte Trägersorte *SetOfEl*. Innerhalb dieses Fallbeispiels haben wir uns auf Datenmodellebene anstelle des OBSCURE-Standardmoduls MK_SET zur Verwendung eines Standardmoduls MK_MONOLIST (zur Spezifikation von Monolisten) entschlossen. Erklärungen zu Monolisten und die Gründe für deren Verwendung sind nachzulesen in [Hec93]. Trotzdem möchte ich im folgenden Mengenbegriffe verwenden, weil sie anschaulicher sind und auf dieser Spezifikationsebene die Problematik der Verwendung von MK_MONOLIST anstelle von MK_SET auf Datenmodellebene nicht von Bedeutung ist.

```
_ = _ : SetOfHK_UW SetOfHK_UW -> bool
_ = _ : SetOfPatient SetOfPatient -> bool
_ = _ : SetOfAKeyHK_UW SetOfAKeyHK_UW -> bool
_ = _ : Untersuchung Untersuchung -> bool
_ = _ : AttrHkId AttrHkId -> bool
```

Wir benötigen ferner verschiedene Mengenoperationen auf unterschiedlichen Mengensorten.

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
        choose : SetOfHK_Befund -> HK_Befund
        choose: SetOfPatient -> Patient
        choose : SetOfHK_UW -> HK_UW
        choose : SetOfAKeyHK_UW -> AKeyHK_UW
        choose: Untersuchung -> UntersuchungsPaar
        del : HK_Befund SetOfHK_Befund -> SetOfHK_Befund
        del : AKeyHK_UW SetOfAKeyHK_UW -> SetOfAKeyHK_UW
        del : HK_UW SetOfHK_UW -> SetOfHK_UW
        del: Patient SetOfPatient -> SetOfPatient
        del: UntersuchungsPaar Untersuchung -> Untersuchung
        ins : BefundId SetOfBefundId -> SetOfBefundId
        ins : AKeyHK_UW SetOfAKeyHK_UW -> SetOfAKeyHK_UW
        ins : HK_UW SetOfHK_UW -> SetOfHK_UW
        #: SetOfAKeyHK_UW -> integer
        _ - _: SetOfAKeyHK_UW SetOfAKeyHK_UW -> SetOfAKeyHK_UW
```

Die folgenden Operationen realisieren den Zugriff auf Attributeinträge zu bestimmten Entitytypen.

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
        BefundId : HK_Befund -> AttrBefundId
        HKAOId: HK_UW -> AttrHkId
        HKAOId: HK_Daten -> AttrHkId
        PatId : Patient -> AttrPatId
```

Die *rtta*...-Operationen bilden Träger von Attributsorten ab auf die zugeordneten Träger der Domainsorten.

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
        rtta : AttrBefundId -> BefundId
        rtta: AttrPatId -> PatId
        rtta: AttrHkId -> HkId
```

Ferner werden drei unterschiedliche *is_in*...-Operationen benötigt. Die erste überprüft für einen Patienten (der durch einen abstrakten Schlüssel identifiziert wird), ob dieser in einem aktuellen Träger der Datenbanksorte enthalten ist. Die zweite überprüft, ob eine Relationship *gehoert_zu* zwischem einem bestimmten Patienten und einer HK-Überweisung existiert, und die dritte testet für eine HK-Überweisung, ob diese zu irgendeinem Objekt (der Sorte *HK_Daten*) in der Relationship *untersuchung* steht.

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
        is_inPatient: AKeyPatient Db –> bool
        is_ingehoert_zu: AKeyPatient AKeyHK_UW Db –> bool
        is_inuntersuchung: AKeyHK_UW Db –> bool
```

Relationships werden beschrieben durch Mengen (oder Listen) von Tupeleinträgen. Um bei einen konkreten Tupeleintrag die einzelnen Komponenten zu selektieren, benötigen wir *get_fst*- und *get_snd*-Operationen.

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
        get_fst: UntersuchungsPaar –> AKeyHK_UW
        get_snd: UntersuchungsPaar –> AKeyHK_Daten
```

Die folgenden *get*...-Operationen realisieren den Zugriff per abstraktem Schlüssel auf einzelne Entitäten, und die *codeKonKey_*...-Operationen berechnen aus einem konkreten Schlüssel einen abstrakten.

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
        getHK_UW: AKeyHK_UW Db –> HK_UW
        getHK_Daten: AKeyHK_Daten Db –> HK_Daten
        codeKonKey_Patient: PatId –> AKeyPatient
        codeKonKey_HK_UW: HkId –> AKeyHK_UW
```

Auf Datenmodellebene wird ein Modul BINARY-RELATION verwendet zur Spezifikation der Relationshiptypen. Dieser Modul stellt eine Operation *reachable_from* zur Verfügung, mithilfe derer es möglich ist, für einen Träger der Urbildsorte des Relationshiptyps die Menge aller Träger der Bildsorte zu bestimmen, zu denen ein Eintrag gesetzt ist. Die *next*...-Operation ist der schematische Anteil der Schlüsselgenenierung (siehe Abschnitt 4.3) und berechnet aus einer Menge von Trägern einer Schlüsselattributsorte einen neuen Träger, der nicht in dieser Menge enthalten ist.

```
(BASIS_SCHNITTSTELLE.obs 25)+ ≡
        reachable_from: AKeyPatient GehoertZu –> SetOfAKeyHK_UW
```

nextBefund : SetOfBefundId –> BefundId

Sämtliche nun folgenden Operationen spezifizieren statische Integritätsbedingungen[9].
Die ...*_key*-Operationen überprüfen für einen bestimmten Entitytyp der Datenbank, ob die Schlüsseleigenschaft von den Einträgen der mit "primary key" (siehe 3.1)ausgezeichneten Attributsorten eingehalten wird.

(BASIS_SCHNITTSTELLE.obs 25)+ ≡
Patient_keys: Db –> bool
HK_UW_keys: Db –> bool
HK_Daten_keys: Db –> bool
Arzt_keys: Db –> bool
HK_Befund_keys: Db –> bool

Die folgenden Operationen überprüfen die Einhaltung der Grade von Relationshiptypen.

(BASIS_SCHNITTSTELLE.obs 25)+ ≡
gehoert_zu_1_N: Db –> bool
untersuchung_1_1: Db –> bool
ordnet_an_N_1: Db –> bool
ermittelt_N_1: Db –> bool
befundung_1_N: Db –> bool
erstellt_N_1: Db –> bool

Die ...*_mandatories*-Operationen überprüfen die Einhaltung der "mandatory"-Bedingungen (siehe 3.1) für die Attributeinträge der Träger eines bestimmten Entitytyps.

(BASIS_SCHNITTSTELLE.obs 25)+ ≡
Patient_mandatories: Db –> bool
HK_UW_mandatories: Db –> bool
HK_Daten_mandatories: Db –> bool
Arzt_mandatories: Db –> bool
HK_Befund_mandatories: Db –> bool

Die folgenden Operationen überprüfen die Bedingungen der zwingenden Partizipation zu einzelnen Relationstypen.

[9]Zur näheren Erläuterung des Begriffs "statische Integritätsbedingungen" siehe auch [Hec93]

(BASIS_SCHNITTSTELLE.obs 25)+ ≡

```
        gehoert_zu_zw : Db -> bool
        untersuchung_zw : Db -> bool
        ordnet_an_zw : Db -> bool
        ermittelt_zw : Db -> bool
        befundung_zw : Db -> bool
        erstellt_zw : Db -> bool
```

4.3 Der Modul SCHLUESSELGENERIERUNG

In diesem Modul wird eine Operation *gen_KonKey_HK_Befund* spezifiziert, die als Argument einen Träger der Sorte *Db* erhält und darauf aufbauend einen neuen konkreten Schlüssel zum Entitytyp[10] *HK_Befund* berechnet. Dazu extrahiert sie zunächst die Menge (oder Liste) der aktuellen Träger der Sorte *HK_Befund*. Anschließend berechnet die Hilfsoperation *befundidset* angewendet auf diese Menge die Menge der zugeordneten Schlüsselattributeinträge der Sorte *BefundId*. Die importierte Operation *nextBefund* kann nun aus dieser Menge von Trägern der Sorte *BefundId* einen weiteren Träger berechnen, der noch nicht in dieser Menge enthalten ist. Somit wird, als Resultat der Operation *gen_KonKey_Befund*, ein neuer konkreter Schlüssel zum Entitytyp *HK_Befund* generiert.

(SCHLUESSELGENERIERUNG.obs 42) ≡

```
INCLUDE BASIS_SCHNITTSTELLE
EXTENDED BY
CREATE
 OPNS
        gen_KonKey_Befund : Db -> BefundId
        befundidset : SetOfHK_Befund -> SetOfBefundId
SEMANTICS
 VARS
        db: Db,
        bs: SetOfHK_Befund
        b : HK_Befund
 PROGRAMS
        gen_KonKey_Befund(db) <-
           LET bs : getSetOfHK_Befund(getDB_Entities(db))
           IN nextBefund(befundidset(bs))
```

[10] Der Begriff Entitytyp word hier analog zu dem Begriff Entitysorte verwendet.

```
        TEL;

    befundidset(bs) <-              ## Hilfsoperation
        IF bs = emptySetOfHK_Befund
        THEN emptySetOfBefundId
        ELSE LET b : choose(bs)
                IN ins(rtta(BefundId(b)),befundidset(del(b,bs)))
                TEL
        FI;
ENDCREATE
ENDEXTENDED
FORGET_ALL_BUT
  OPNS gen_KonKey_Befund **
```

Außer der Operation *gen_KonKey_Befund* und den mit dieser verbundenen Sorten werden am Ende alle sonstigen Operationen und Sorten vergessen.

4.4 Der Modul SONDERFUNKTIONEN

In diesem Modul wird die Sonderfunktion *akt_uw* spezifiziert. Diese Operation erhält als Argumente einen Träger *db* der Sorte *Db* und einen Träger *pi* der Sorte *PatId* und soll auf dieser Grundlage einen Träger *uw* der Sorte *HK_UW* berechnen, für den gefordert wird, daß dieser zu dem durch *pi* beschriebenen Träger des Entitytyps *Patient* in der Relationship *gehoert_zu* steht und ferner kein Träger der Sorte *HK_Daten* in *db* existiert, der zum berechneten Träger *uw* in der Relationship *untersuchung* steht. Einfacher ausgedrückt, berechnet diese Sonderfunktion eine noch nicht bearbeitete HK-Überweisung zu einem bestimmten Patienten. Dazu verwendet sie Hilfsoperationen *all_in_gehoert_zu* und *all_not_in_untersuchung*.
Neben der Operation *akt_uw* wird zunächst noch das entsprechende Definiertheitsprädikat *akt_uw_def* spezifiziert, welches überprüft, ob eine noch nicht bearbeitete HK-Überweisung zu einem bestimmten Patienten existiert.

```
   (SONDERFUNKTIONEN.obs 44) ≡
INCLUDE BASIS_SCHNITTSTELLE
EXTENDED BY
CREATE
  OPNS
        akt_uw : Db PatId -> HK_UW
        akt_uw_def : Db PatId -> bool
        all_in_gehoert_zu : SetOfHK_UW PatId Db -> SetOfHK_UW
```

```
        all_not_in_untersuchung : SetOfHK_UW Db -> SetOfHK_UW
SEMANTICS
 VARS
        db : Db
        pi : PatId
        uwset,uwpi,uwo,rest : SetOfHK_UW
        uw: HK_UW
        auwi: AKeyHK_UW
        api: AKeyPatient
 PROGRAMS
        akt_uw_def(db,pi) <-
           IF is_inPatient(codeKonKey_Patient(pi),db)
           THEN LET uwset : getSetOfHK_UW(getDB_Entities(db))
                    IN LET uwpi : all_in_gehoert_zu(uwset,pi,db)
                       IN IF not(uwpi = emptySetOfHK_UW)
                          THEN LET uwo : all_not_in_untersuchung(uwpi,db)
                                   IN not(uwo = emptySetOfHK_UW)
                                   TEL
                          ELSE false
                          FI
                       TEL
                    TEL
           ELSE false
           FI;
```

Siehe auch owf–moduln 45 und 46.

(SONDERFUNKTIONEN.obs 44)+ ≡

```
        akt_uw(db,pi) <-
           IF akt_uw_def(db,pi)
           THEN LET uwset : getSetOfHK_UW(getDB_Entities(db))
                    IN LET uwpi : all_in_gehoert_zu(uwset,pi,db)
                       IN LET uwo : all_not_in_untersuchung(uwpi,db)
                          IN choose(uwo)
                          TEL
                       TEL
                    TEL
           ELSE ERROR(HK_UW)
           FI;

        all_in_gehoert_zu(uwset,pi,db) <-          ## Hilfsoperation
           IF uwset = emptySetOfHK_UW
           THEN emptySetOfHK_UW
           ELSE
```

```
            LET uw: choose(uwset)
            IN LET
                  rest: del(uw,uwset),
                  api: codeKonKey_Patient(pi),
                  auwi: codeKonKey_HK_UW(rtta(HKAOId(uw)))
               IN IF is_ingehoert_zu(api,auwi,db)
                     THEN ins(uw,all_in_gehoert_zu(rest,pi,db))
                     ELSE all_in_gehoert_zu(rest,pi,db)
                     FI
               TEL
            TEL
         FI;

(SONDERFUNKTIONEN.obs 44)+ ≡
      all_not_in_untersuchung(uwset,db) <-        ## Hilfsoperation
         IF uwset = emptySetOfHK_UW
         THEN emptySetOfHK_UW
         ELSE
          LET uw: choose(uwset)
          IN LET
                rest: del(uw,uwset),
                auwi: codeKonKey_HK_UW(rtta(HKAOId(uw)))
             IN IF not(is_inuntersuchung(auwi,db))
                   THEN ins(uw,all_not_in_untersuchung(rest,db))
                   ELSE all_not_in_untersuchung(rest,db)
                   FI
             TEL
          TEL
           FI
ENDCREATE
ENDEXTENDED
FORGET_ALL_BUT
  OPNS akt_uw **, akt_uw_def**
```

Hilfsoperationen und Hilfssorten werden am Ende vergessen.

4.5 Der Modul OK_PRAEDIKAT

Sämtliche statische Integritätsbedingungen werden in diesem Modul zu einem OK-Prädikat *OK* zusammengefaßt. Das OK-Prädikat umfaßt dabei folgende Aspekte(siehe auch [Hec93]):

- Grade von Relationshiptypen: Zu jeder Relationship wird im E/R-Modell der Grad benannt. Es wird dabei unterschieden zwischen den Graden 1:1, 1:N (bzw. N:1) und N:M.

- Zwingende Partizipation: Für Entities kann die Teilnahme an einer Relationship im E/R-Modell explizit gefordert werden.
  Beispiel: Eine Entity der Sorte HK_UW muß immer in der Relationship *gehört-zu* zu einer Entity der Sorte *Patient* stehen. Allgemeiner ausgedrückt heißt das, daß eine HK-Überweisung immer einem Patienten zugeordnet sein muß.

- Schlüsseleigenschaft von Attributeinträgen: In der Beschreibung der einzelnen Entitytypen (siehe 3.1) werden Schlüsselattribute als solche gekennzeichnet. Die Schlüsseleigenschaft eines Attributes stellt sicher, daß in einem konkreten Träger der Sorte Db ein Schlüsselwert stets genau einen Träger des entsprechenden Entitytyp identifiziert.

- Zwingende Attributeinträge: Attribute von Entitytypen, die einen zwingenden Eintrag vorsehen, werden ebenfalls in der Beschreibung gekennzeichnet. In diesem Falle soll sichergestellt werden, daß ein solches Attribut niemals den leeren Eintrag enthält.

- Spezielle statische Integritätsbedingungen der einzelnen Abläufe: Für den Ablauf HK-Untersuchung wird speziell gefordert:

  - Wenn eine Entity des Typs HK_UW zu einer Entity des Typs HK_Daten in der Relationship *untersuchung* steht, so hat das Attribut $HKAOId$ bei beiden den gleichen Wert. Dieser Bedingung wird mittels der Operation $C\_spezial\_1\_HK$ spezifiziert.
  - Zu jedem Patienten existiert höchstens eine noch nicht bearbeitete HK-Überweisung. Diese Bedingung wird mittels der Operation $C\_spezial2\_HK$ spezifiziert.

Diese Bedingungen können für einen gegebenen Träger der Sorte Db mittels des OK-Prädikats überprüft werden.

```
   (OK_PRAEDIKAT.obs 48) ≡
INCLUDE BASIS_SCHNITTSTELLE
EXTENDED BY
CREATE
  OPNS
          OK: Db –> bool
```

```
            C_spezial_1_HK: Db -> bool
            C_spezial_2_HK: Db -> bool
            C_help1: Untersuchung Db -> bool
            C_help2_a: SetOfPatient Db -> bool
            C_help2_b: Patient Db -> bool
            help_offene_HK_UW: SetOfAKeyHK_UW Db -> SetOfAKeyHK_UW
SEMANTICS
 VARS
            db : Db
            r : UntersuchungsPaar
            gz, rest : Untersuchung
            pset, pset' : SetOfPatient
            p : Patient
            api : AKeyPatient
            auwiset, auwiset': SetOfAKeyHK_UW
            auwi : AKeyHK_UW
```

Siehe auch owf–moduln 49, 50, und 51.

(OK_PRAEDIKAT.obs 48)+ ≡

```
 PROGRAMS
            C_spezial_1_HK(db) <-
               LET gz : getuntersuchung(getDB_Relationships(db))
               IN C_help1(gz,db)
               TEL;

            C_help1(gz,db) <-
               IF gz = emptyuntersuchung THEN true
               ELSE
                LET r : choose(gz)
                IN LET rest : del(r,gz)
                    IN HKAOId(getHK_UW(get_fst(r),db))
                       =
                       HKAOId(getHK_Daten(get_snd(r),db))
                       and
                       C_help1(rest,db)
                    TEL
                TEL
               FI;

            C_spezial_2_HK(db) <-
               LET pset : getSetOfPatient(getDB_Entities(db))
```

```
            IN C_help2_a(pset,db)
            TEL;

        C_help2_a(pset,db) <-
            IF pset = emptySetOfPatient THEN true
            ELSE
              LET p : choose(pset)
              IN LET pset' : del(p,pset)
                  IN C_help2_b(p,db)and
                      C_help2_a(pset',db)
                  TEL
              TEL
            FI;

(OK_PRAEDIKAT.obs 48)+ ≡
        C_help2_b(p,db) <-
            LET api : codeKonKey_Patient(rtta(PatId(p)))
            IN LET auwiset : reachable_from(api,
                                getgehoert_zu(getDB_Relationships(db)))
                IN LET auwiset' : help_offene_HK_UW(auwiset,db)
                    IN #(auwiset - auwiset') <=1
                    TEL
                TEL
            TEL;
        help_offene_HK_UW(auwiset,db) <-
            IF auwiset = emptySetOfAuwi
            THEN emptySetOfAuwi
            ELSE LET auwi : choose(auwiset)
                    IN LET auwiset': del(auwi,auwiset)
                        IN IF is_inuntersuchung(auwi,db)
                            THEN help_offene_HK_UW(auwiset',db)
                            ELSE ins(auwi,help_offene_HK_UW(auwiset',db))
                            FI
                        TEL
                    TEL
            FI;

(OK_PRAEDIKAT.obs 48)+ ≡
        OK(db) <-
            C_spezial_1_HK(db) and          ## spezielle Integrit"atsbed
            C_spezial_2_HK(db) and
            Patient_mandatories(db) and     ## zwingende Attributeintr"age
            HK_UW_mandatories(db) and
```

HK_Daten_mandatories(db) and
Arzt_mandatories(db) and
HK_Befund_mandatories(db) and
Patient_keys(db) and ## Schluesseleigenschaft
HK_UW_keys(db) and
HK_Daten_keys(db) and
Arzt_keys(db) and
HK_Befund_keys(db) and
gehoert_zu_1_N(db) and ## Grade von Beziehungstypen
untersuchung_1_1(db) and
ordnet_an_N_1(db) and
ermittelt_N_1(db) and
befundung_1_N(db) and
erstellt_N_1(db) and
gehoert_zu_zw(db) and ## zwingende Partizipation
untersuchung_zw(db) and
ordnet_an_zw(db) and
ermittelt_zw(db) and
befundung_zw(db) and
erstellt_zw(db)

ENDCREATE
ENDEXTENDED
FORGET_ALL_BUT
OPNS OK **

Bis auf das OK-Prädikat werden am Ende sämtliche Operationen und Sorten vergessen.

4.6 Die Exportsignatur der Basisebene

Folgender Signaturausdruck wurde nach dem erfolgreichen Parsen der hier beschriebenen Spezifikation BASIS automatisch erstellt.

```
File name: "BASIS.T"
list of sorts and operations:

imported and exported (inherited).
 SORTS
  Db BefundId HK_UW PatId
  ## no operations
```

```
not imported and exported (created).
  ## no sorts
 OPNS
  gen_KonKey_Befund : Db -> BefundId
  akt_uw_def : Db PatId -> bool
  OK : Db -> bool
  akt_uw : Db PatId -> HK_UW

imported and not exported (hidden).
 SORTS
  GehoertZu SetOfAKeyHK_UW SetOfHK_UW SetOfBefundId
  SetOfHK_Befund AttrPatId AttrHkId AttrBefundId
  AKeyHK_Daten SetOfPatient UntersuchungsPaar HK_Daten
  Untersuchung AKeyHK_UW Patient DB_Relationships HkId
  DB_Entities HK_Befund Arzt AKeyPatient
 OPNS
  getgehoert_zu : DB_Relationships -> GehoertZu
  reachable_from : AKeyPatient GehoertZu -> SetOfAKeyHK_UW
  ins : AKeyHK_UW SetOfAKeyHK_UW -> SetOfAKeyHK_UW
  del : AKeyHK_UW SetOfAKeyHK_UW -> SetOfAKeyHK_UW
  _ - _ : SetOfAKeyHK_UW SetOfAKeyHK_UW -> SetOfAKeyHK_UW
  emptySetOfAuwi : -> SetOfAKeyHK_UW
  ins : HK_UW SetOfHK_UW -> SetOfHK_UW
  emptySetOfHK_UW : -> SetOfHK_UW
  del : HK_UW SetOfHK_UW -> SetOfHK_UW
  getSetOfHK_UW : DB_Entities -> SetOfHK_UW
  nextBefund : SetOfBefundId -> BefundId
  rtta : AttrBefundId -> BefundId
  emptySetOfBefundId : -> SetOfBefundId
  ins : BefundId SetOfBefundId -> SetOfBefundId
  emptySetOfHK_Befund : -> SetOfHK_Befund
  del : HK_Befund SetOfHK_Befund -> SetOfHK_Befund
  getSetOfHK_Befund : DB_Entities -> SetOfHK_Befund
  PatId : Patient -> AttrPatId
  HKAOId : HK_Daten -> AttrHkId
  HKAOId : HK_UW -> AttrHkId
  BefundId : HK_Befund -> AttrBefundId
  get_snd : UntersuchungsPaar -> AKeyHK_Daten
  emptySetOfPatient : -> SetOfPatient
```

```
getSetOfPatient : DB_Entities -> SetOfPatient
del : Patient SetOfPatient -> SetOfPatient
choose : Untersuchung -> UntersuchungsPaar
getHK_Daten : AKeyHK_Daten Db -> HK_Daten
emptyuntersuchung : -> Untersuchung
del : UntersuchungsPaar Untersuchung -> Untersuchung
getuntersuchung : DB_Relationships -> Untersuchung
codeKonKey_HK_UW : HkId -> AKeyHK_UW
choose : SetOfAKeyHK_UW -> AKeyHK_UW
get_fst : UntersuchungsPaar -> AKeyHK_UW
untersuchung_1_1 : Db -> bool
is_inPatient : AKeyPatient Db -> bool
is_inuntersuchung : AKeyHK_UW Db -> bool
Patient_mandatories : Db -> bool
Patient_keys : Db -> bool
erstellt_zw : Db -> bool
ermittelt_zw : Db -> bool
erstellt_N_1 : Db -> bool
ermittelt_N_1 : Db -> bool
befundung_1_N : Db -> bool
ordnet_an_N_1 : Db -> bool
gehoert_zu_zw : Db -> bool
is_ingehoert_zu : AKeyPatient AKeyHK_UW Db -> bool
befundung_zw : Db -> bool
gehoert_zu_1_N : Db -> bool
untersuchung_zw : Db -> bool
Arzt_mandatories : Db -> bool
Arzt_keys : Db -> bool
ordnet_an_zw : Db -> bool
HK_Befund_mandatories : Db -> bool
HK_Daten_mandatories : Db -> bool
HK_UW_mandatories : Db -> bool
HK_Befund_keys : Db -> bool
HK_Daten_keys : Db -> bool
HK_UW_keys : Db -> bool
_ = _ : SetOfAKeyHK_UW SetOfAKeyHK_UW -> bool
_ = _ : SetOfHK_UW SetOfHK_UW -> bool
_ = _ : SetOfHK_Befund SetOfHK_Befund -> bool
_ = _ : AttrHkId AttrHkId -> bool
_ = _ : SetOfPatient SetOfPatient -> bool
```

```
  _ = _ : Untersuchung Untersuchung -> bool
  choose : SetOfPatient -> Patient
  getDB_Relationships : Db -> DB_Relationships
  rtta : AttrHkId -> HkId
  getDB_Entities : Db -> DB_Entities
  choose : SetOfHK_Befund -> HK_Befund
  codeKonKey_Patient : PatId -> AKeyPatient
  # : SetOfAKeyHK_UW -> integer
  choose : SetOfHK_UW -> HK_UW
  getHK_UW : AKeyHK_UW Db -> HK_UW
  rtta : AttrPatId -> PatId

list of constructors:

imported

exported

list of parameters:

imported.
 ## no sorts
 ## no operations

exported.
 ## no sorts
 ## no operations
## The End
```

5 Die Gesamtspezifikation

5.1 Der Modul ATOMAR

Nun können wir die beiden Module BASIS und HK_UNTERSUCHUNG mit dem OBSCURE-Konstrukt EXTENDED BY zusammensetzen. Übertragen auf den gesamten HDMS-A Kontext, würde an dieser Stelle ein wesentlich umfangreicherer Modul BASIS - der weitere spezielle Operationen zu den hier nicht behandelten Abläufen von HDMS-A enthielte - erweitert werden um eine Kombination sämtlicher Ablaufspezifikationen. Da wir uns lediglich auf den Ablauf HK-Untersuchung konzentriert haben, entfällt dies hier.
In einem weiteren Schritt wenden wir das OK-Prädikat auf unsere Spezifikation an. Für jede unserer vier elementaren Transaktionen wollen wir fordern, daß sämtliche Integritätsbedingungen, die vor der Ausführung einer elementaren Transaktion von einem Träger der Sorte *Db* erfüllt werden, auch nach der Ausführung gültig sind. In der Spezifikation formulieren wir dazu zu jeder elementaren Transaktion ein Exportaxiom.
Anschließend vergessen wir bis auf die elementaren und die mit diesen verbundenen Sorten alles weitere.

`(ATOMAR.obs` 52) ≡

INCLUDE BASIS
EXTENDED BY
(
INCLUDE AUFNAHME
PLUS
INCLUDE ARZT_ABLAUF
PLUS
INCLUDE BEHANDLUNGS_ABLAUF
PLUS
INCLUDE LABOR_ABLAUF
PLUS

INCLUDE HK_UNTERSUCHUNG

PLUS
INCLUDE ENTLASSUMG
)

E_AXIOMS

```
VARS db : Db,
     pi : PatId,
     ai : ArztId,
     hi : HkId,
     hkd: HK_Daten,
     dt : DateTime,
     b  : HK_Befund,
     bi : BefundId,
     bb : Befundbrief;

OK(db) == true => OK(get_fst(init_HKP(db,pi,ai,dt))) == true |
                  init_HKP(db,pi,ai,dt) == ERROR(Db_x_HkId);
OK(db) == true => OK(get_fst(HK_Untersuchung(db,hi,hkd))) == true |
                    HK_Untersuchung(db,hi,hkd) == ERROR(Db_x_HkId);
OK(db) == true => OK(get_fst(Befundung(db,hi,ai,b,dt))) == true |
                    Befundung(db,hi,ai,b,dt) == ERROR(Db_x_BefundId);
OK(db) == true => OK(Briefschreibung(db,bi,bb)) == true |
                    Briefschreibung(db,bi,bb) == ERROR(Db);

## In einer kompletten Spezifikation zur atomaren Ebene
## wuerde hier das OK-Praedikat auf alle elementaren Transaktionen
## angewendet

ENDAXIOMS
ENDEXTENDED

FORGET_ALL_BUT
OPNS
     init_HKP : **,
     HK_Untersuchung : **,
     Befundung : **,
     Briefschreibung: **
## und alle sonstigen elementaren Transaktionen
```

5.2 Die Signatur der Gesamtspezifikation

Die folgende Auflistung der Signatur zur Spezifikation der atomaren Ebene wurde mit dem OBSCURE-System automatisch erzeugt.

```
File name: "ATOMAR.T"
list of sorts and operations:

imported and exported (inherited).
 SORTS
  Db BefundId ArztId DateTime HK_Daten HkId HK_Befund
  Befundbrief PatId
  ## no operations

not imported and exported (created).
 SORTS
  Db_x_HkId Db_x_BefundId
 OPNS
  Briefschreibung : Db BefundId Befundbrief -> Db
  init_HKP : Db PatId ArztId DateTime -> Db_x_HkId
  HK_Untersuchung : Db HkId HK_Daten -> Db_x_HkId
  Befundung : Db HkId ArztId HK_Befund DateTime ->
  Db_x_BefundId

imported and not exported (hidden).
 SORTS
  GehoertZu SetOfAKeyHK_UW SetOfHK_UW AttrBefundbrief
  AttrDateTime SetOfBefundId SetOfHK_Befund AKeyHK_Befund
  AttrBefund AttrArztId AttrPatId AttrHkId AttrBefundId
  AttrDruckkurven AKeyHK_Daten SetOfPatient
  UntersuchungsPaar AttrRoentgenfilm Untersuchung AKeyHK_UW
  Patient Roentgenfilm Text Druckkurven DB_Relationships
  DB_Entities AKeyArzt AttrText Arzt AKeyPatient HK_UW
 OPNS
  getgehoert_zu : DB_Relationships -> GehoertZu
  esterstellt : Db Arzt HK_Befund -> Db
  estermittelt : Db Arzt HK_Daten -> Db
  estbefundung : Db HK_Daten HK_Befund -> Db
  estuntersuchung : Db HK_UW HK_Daten -> Db
  putHK_Daten : HK_Daten Db -> Db
  updateHK_Befund : HK_Befund Db -> Db
  putHK_Befund : HK_Befund Db -> Db
  updateHK_Daten : HK_Daten Db -> Db
  reachable_from : AKeyPatient GehoertZu -> SetOfAKeyHK_UW
  ins : AKeyHK_UW SetOfAKeyHK_UW -> SetOfAKeyHK_UW
```

```
del : AKeyHK_UW SetOfAKeyHK_UW -> SetOfAKeyHK_UW
_ - _ : SetOfAKeyHK_UW SetOfAKeyHK_UW -> SetOfAKeyHK_UW
emptySetOfAuwi : -> SetOfAKeyHK_UW
ins : HK_UW SetOfHK_UW -> SetOfHK_UW
emptySetOfHK_UW : -> SetOfHK_UW
del : HK_UW SetOfHK_UW -> SetOfHK_UW
getSetOfHK_UW : DB_Entities -> SetOfHK_UW
nextBefund : SetOfBefundId -> BefundId
rtta : AttrBefundId -> BefundId
attr : Befundbrief -> AttrBefundbrief
UNDEF_Befundbrief : -> AttrBefundbrief
Befundbrief : HK_Befund -> AttrBefundbrief
UNDEF_DateTime : -> AttrDateTime
Untersuchungsdatum : HK_Daten -> AttrDateTime
attr : DateTime -> AttrDateTime
Anfangszeit : HK_Daten -> AttrDateTime
Endezeit : HK_Daten -> AttrDateTime
Befundungsdatum : HK_Befund -> AttrDateTime
emptySetOfBefundId : -> SetOfBefundId
ins : BefundId SetOfBefundId -> SetOfBefundId
emptySetOfHK_Befund : -> SetOfHK_Befund
del : HK_Befund SetOfHK_Befund -> SetOfHK_Befund
getSetOfHK_Befund : DB_Entities -> SetOfHK_Befund
codeKonKey_HK_Befund : BefundId -> AKeyHK_Befund
Befund : HK_Befund -> AttrBefund
attr : HK_Befund -> AttrBefund
PatId : Patient -> AttrPatId
HKAOId : HK_Daten -> AttrHkId
HKAOId : HK_UW -> AttrHkId
attr : HkId -> AttrHkId
BefundId : HK_Befund -> AttrBefundId
attr : BefundId -> AttrBefundId
attr : Druckkurven -> AttrDruckkurven
UNDEF_Druckkurven : -> AttrDruckkurven
codeKonKey_HK_Daten : HkId -> AKeyHK_Daten
get_snd : UntersuchungsPaar -> AKeyHK_Daten
emptySetOfPatient : -> SetOfPatient
getSetOfPatient : DB_Entities -> SetOfPatient
del : Patient SetOfPatient -> SetOfPatient
choose : Untersuchung -> UntersuchungsPaar
```

```
UNDEF_Roentgenfilm : -> AttrRoentgenfilm
attr : Roentgenfilm -> AttrRoentgenfilm
getHK_Daten : AKeyHK_Daten Db -> HK_Daten
med_hku : HkId -> HK_Daten
createHK_Daten : AttrHkId AttrText AttrDateTime
AttrDateTime AttrDruckkurven AttrRoentgenfilm -> HK_Daten
emptyuntersuchung : -> Untersuchung
del : UntersuchungsPaar Untersuchung -> Untersuchung
getuntersuchung : DB_Relationships -> Untersuchung
codeKonKey_HK_UW : HkId -> AKeyHK_UW
choose : SetOfAKeyHK_UW -> AKeyHK_UW
get_fst : UntersuchungsPaar -> AKeyHK_UW
untersuchung_1_1 : Db -> bool
is_inArzt : AKeyArzt Db -> bool
is_inPatient : AKeyPatient Db -> bool
is_inuntersuchung : AKeyHK_UW Db -> bool
Patient_mandatories : Db -> bool
Patient_keys : Db -> bool
is_inHK_Befund : AKeyHK_Befund Db -> bool
erstellt_zw : Db -> bool
ermittelt_zw : Db -> bool
erstellt_N_1 : Db -> bool
ermittelt_N_1 : Db -> bool
is_inHK_Daten : AKeyHK_Daten Db -> bool
befundung_1_N : Db -> bool
ordnet_an_N_1 : Db -> bool
gehoert_zu_zw : Db -> bool
is_ingehoert_zu : AKeyPatient AKeyHK_UW Db -> bool
befundung_zw : Db -> bool
gehoert_zu_1_N : Db -> bool
untersuchung_zw : Db -> bool
Arzt_mandatories : Db -> bool
Arzt_keys : Db -> bool
ordnet_an_zw : Db -> bool
HK_Befund_mandatories : Db -> bool
HK_Daten_mandatories : Db -> bool
HK_UW_mandatories : Db -> bool
HK_Befund_keys : Db -> bool
HK_Daten_keys : Db -> bool
HK_UW_keys : Db -> bool
```

```
  _ = _ : SetOfAKeyHK_UW SetOfAKeyHK_UW -> bool
  _ = _ : SetOfHK_UW SetOfHK_UW -> bool
  _ = _ : AttrBefundbrief AttrBefundbrief -> bool
  _ = _ : AttrDateTime AttrDateTime -> bool
  _ = _ : SetOfHK_Befund SetOfHK_Befund -> bool
  _ = _ : AttrHkId AttrHkId -> bool
  _ = _ : SetOfPatient SetOfPatient -> bool
  _ = _ : Untersuchung Untersuchung -> bool
  choose : SetOfPatient -> Patient
  getPatient : AKeyPatient Db -> Patient
  getDB_Relationships : Db -> DB_Relationships
  rtta : AttrHkId -> HkId
  getDB_Entities : Db -> DB_Entities
  choose : SetOfHK_Befund -> HK_Befund
  createHK_Befund : AttrBefundId AttrDateTime AttrBefund
  AttrBefundbrief -> HK_Befund
  getHK_Befund : AKeyHK_Befund Db -> HK_Befund
  codeKonKey_Arzt : ArztId -> AKeyArzt
  attr : Text -> AttrText
  UNDEF_Text : -> AttrText
  getArzt : AKeyArzt Db -> Arzt
  codeKonKey_Patient : PatId -> AKeyPatient
  # : SetOfAKeyHK_UW -> integer
  choose : SetOfHK_UW -> HK_UW
  getHK_UW : AKeyHK_UW Db -> HK_UW
  rtta : AttrPatId -> PatId

list of constructors:

imported

exported

list of parameters:

imported.
 ## no sorts
 ## no operations

exported.
```

```
 ## no sorts
 ## no operations
## The End
```

6 Nähere Erläuterungen zur Form

Mittels OPTICAL erstellte *Dokumentationen* von OBSCURE–Spezifikationen zerlegen den OBSCURE–Quelltext in eine Menge sich gegenseitig referenzierender *Beschreibungseinheiten*[11] . Wie verhalten sich diese Beschreibungseinheiten zu OBSCURE–*Moduln*?

- Das Prinzip ist natürlich dasselbe: die Modularisierung einer Spezifikation zerlegt den OBSCURE–Quelltext in eine Menge sich gegenseitig referenzierender Moduln (**INCLUDE**–Sprachkonstrukt!). Man beachte, daß sowohl die Moduln als auch die Beschreibunhseinheiten benannt sind, wodurch die Referenzierung überhaupt erst ermöglicht wird.
- ABER: diese beiden Zerlegungen müssen a priori nicht viel miteinander zu tun haben. In der Regel (wie auch im vorliegenden Falle!) werden jedoch jedem *Modul* jeweils eine oder mehrere *Beschreibungseinheiten* entsprechen.

Die Definition einer Beschreibungseinheit kann über mehrere Stellen in der Dokumentation verteilt sein: ihr Name kann also mehrere definierende Vorkommen haben. Als Beispiel betrachten wir einen Modul mit dem Namen "BASIS_SCHNITTSTELLE", der durch genau eine Beschreibungseinheit beschrieben werden soll. Der Name der Beschreibungseinheit sei "BASIS_SCHNITTSTELLE.obs":

1. Irgendwo in der Dokumentation steht das *erste* definierende Vorkommen des Namens "BASIS_SCHNITTSTELLE.obs" – und zwar in der Form:

   (BASIS_SCHNITTSTELLE.obs 15) ≡
   ... <Quelltext> ...

   Dabei enthält <Quelltext> beispielsweise die ersten fünf Zeilen des Moduls BASIS_SCHNITTSTELLE. Jede Beschreibungseinheit eines Dokuments hat eine eindeutige Nummer – in unserem Falle die Nummer 15.

[11] Ein zu "Beschreibungseinheit" äquivalenter Begriff ist "OWF–Modul"

2. An einer (späteren) anderen Stelle wird nun ein weiteres definierendes Vorkommen auftauchen - und zwar in der Form:

   (BASIS_SCHNITTSTELLE.obs 15) + ≡
   ... <Quelltext> ...

   Diesmal könnte <Quelltext> z.B. die Zeilen sechs bis zwanzig des Moduls BASIS_SCHNITTSTELLE enthalten.

3. Die gesamte Beschreibungseinheit BASIS_SCHNITTSTELLE.obs enthält genau die in den definierenden Vorkommen ihres Namens aufgeführten Quelltext–Stücke. Die Quelltext–Stücke denke man sich einfach zusammengesetzt in der Reihenfolge ihres Auftretens.

4. Es spielt keine Rolle, ob definierende Vorkommen der Namen von Beschreibungseinheiten von runden Klammern umgeben sind oder von "eckigen" – wie z.B. in:

   ⟨BASIS_SCHNITTSTELLE.obs 15⟩ ≡
   ... <Quelltext> ...

5. Angaben der Form Siehe auch owf–moduln 12, 13, ... sind *völlig falsch* und sollten daher *ignoriert* werden.

Die Namen von Beschreibungseinheiten können in der Definition anderer Beschreibungseinheiten *angewandte Vorkommen* haben. Dies hat zwei Konsequenzen:

1. Zum einen könnte die Definition der Beschreibungseinheit BASIS_SCHNITTSTELLE zurückgreifen auf andere Beschreibungseinheiten. Beispiel:

   (BASIS_SCHNITTSTELLE.obs 15) ≡
   ... <Quelltext> ...
   ⟨Operationen der BASIS_SCHNITTSTELLE 9⟩
   ... <noch mehr Quelltext> ...

   Hier ist ⟨Operationen der BASIS_SCHNITTSTELLE 9⟩ein angewandtes Vorkommen des Namens "Operationen der BASIS_SCHNITTSTELLE". Die zugehörige Beschreibungseinheit hat die Nummer 9.

2. Zum andern wird bei definierenden Vorkommen der Namen von Beschreibungseinheiten stets angegeben, in welchen anderen Definitionen sie ein angewandtes Vorkommen haben, z.B:

(Operationen der BASIS_SCHNITTSTELLE.obs 9) ≡
... <Quelltext> ...
Dieser Code wird verwendet in owf–modul 15

Bemerkung: verwendet die Definition von BASIS_SCHNITTSTELLE.obs den Namen "Operationen der BASIS_SCHNITTSTELLE", so bedeutet dies insbesondere, daß der OBSCURE–Modul BASIS_SCHNITTSTELLE in mindestens *zwei* Beschreibungseinheiten zerlegt wurde.

Index

Literatur

[Aut93] S. Autexier. HDMS-A und OBSCURE in KORSO – Die funktionale Essenz von HDMS-A aus Sicht der algorithmischen Spezifikationsmethode. Teil 2: Schablonen zur Übersetzung eines E/R-Schemas in eine OBSCURE Spezifikation. Technical Report A/05/93, Universität des Saarlandes, Saarbrücken, December 1993.

[Ben93] C. Benzmüller. HDMS-A und OBSCURE in KORSO – Die funktionale Essenz von HDMS-A aus Sicht der algorithmischen Spezifikationsmethode. Teil 3: Die Spezifikation der atomaren Funktionen. Technical Report A/06/93, Universität des Saarlandes, Saarbrücken, December 1993.

[CHL94a] F. Cornelius, H. Hußmann, and M. Löwe. The KORSO Case Study for Software Engineering with Formal Methods: A Medical Information System. Technical Report 94-5, Technische Universität Berlin, February 1994.

[CHL94b] F. Cornelius, H. Hußmann, and M. Löwe. The KORSO Case Study for Software Engineering with Formal Methods: A Medical Information System. In Broy, M. and Jähnichen, S., editor, KORSO–*Abschlußband (noch offen)*. Springer LNCS, 1994. to appear, also published in an extended version as technical report no. 94-5, Technische Universität Berlin, February 1994.

[CKL93] F. Cornelius, M. Klar, and M. Löwe. Ein Fallbeispiel für KORSO: Ist–Analyse HDMS-A. Technical Report 93-28, Technische Universität Berlin, 1993.

[Con93] S. Conrad. Einbindung eines bestehenden Datenbanksystems in einen formalen Software–Entwicklungsprozeß — ein Beitrag zur HDMS-A–Fallstudie. In H.-D. Ehrich, editor, *Beiträge zu* KORSO–*und* TROLL *light–Fallstudien*, pages 1–14. Technische Universität Braunschweig, Informatik–Bericht 93–11, 1993.

[Dam93] W. Damm. KORSO–Applikationen — HDMS-A Teilprojekt Universität Oldenburg. Short description of ongoing work, February 1993.

[Fuc94] T. Fuchß. Translating E/R-diagrams into Consistent Database Specifications. Technical Report 2/94, Universität Karlsruhe, January 1994.

[GH93] M. Grosse and H. Hufschmidt. SOLL–Spezifikation aus Sicht der Sicherheit. Technical Report A/07/93, Universität des Saarlandes, Saarbrücken, December 1993.

[Hec92] Ramses A. Heckler. Das *OPTICAL*–System. Interner Bericht (WP 92/37), Juli 1992. Universität des Saarlandes.

[Hec93] R.A. Heckler. HDMS-A und OBSCURE in KORSO – Die funktionale Essenz von HDMS-A aus Sicht der algorithmischen Spezifikationsmethode. Teil 1: Einführung und Anmerkungen. Technical Report A/04/93, Universität des Saarlandes, Saarbrücken, December 1993.

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

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

[MZ94] M. Mehlich and W. Zhang. Specifying Interactive Components for Configuratiing Graphical User Interfaces. Technical Report 9401, Ludwig-Maximilians-Universität München, 1994.

[Nic93] F. Nickl. Ablaufspezifikation durch Datenflußmodellierung und stromverarbeitende Funktionen. Technical Report TUM-I9334, Technische Universität München, 1993.

[Ren94] K. Renzel. Formale Beschreibung von Sicherheitsaspekten für das Fallbeispiel HDMS-A. Technical Report 9402, Ludwig-Maximilians-Universität Munich, January 1994.

[Sch94] M. Schulte. Spezifikation und Verifikation von kommunizierenden Objekten in einem verteilten System. Master's thesis, University of Oldenburg, Computer Science Department, March 1994. (in German).

[SH94] M. Strecker and R. A. Heckler. Modifizierungsrahmen, Zwei–Ebenen–Konzept und Eintopf–Konzept — Erster Bericht der Rahmen–Gruppe — vormals "Erweiterbarkeitsgruppe". Technical Report at the Universities of Ulm and Saarbrücken, to appear, 1994.

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

[SNM+93] O. Slotosch, F. Nickl, S. Merz, H. Hußmann, and R. Hettler. Die funktionale Essenz von HDMS-A. Technical Report TUM-I9335, Technische Universität München, 1993.

[Ste93] K. Stenzel. A Verified Access Control Model. Technical Report 26/93, Fakultät für Informatik, Universität Karlsruhe, Germany, December 1993.