Medizinische Fakultät
der
Universität Duisburg-Essen

Institut für Medizinische Informatik, Biometrie und Epidemiologie

Ein integriertes System zur computerunterstützten Entwicklung medizinischer Dokumentationssysteme unter besonderer Berücksichtigung von Integritätsbedingungen (*InCoMe*)

Inaugural-Dissertation
zur
Erlangung des Doktorgrades
der Naturwissenschaften in der Medizin
(Dr. rer. medic.)
durch die Medizinische Fakultät
der Universität Duisburg-Essen

vorgelegt von
Dipl.-Wirt.-Inf. Ralf Goertzen
aus Oberhausen
2005

Dekan:	Univ.-Prof. Dr. Karl-Heinz Jöckel
1. Gutachter:	Priv.-Doz. Dr. Jürgen Stausberg
2. Gutachter:	Univ.-Prof. Dr. Frank-Dieter Dorloff
3. Gutachter:	Prof. Dr. Hans-Ulrich Prokosch

Tag der mündlichen Prüfung: 27. Oktober 2005

Publikationen zu Ergebnissen dieser Arbeit

Goertzen, R., Stausberg, J. (2004): Ein Modell zur computerunterstützten Generierung von medizinischen Dokumentationssystemen. Ger Med Sci GM04gmds305. Online-Publikation; http://www.egms.de/en/meetings/gmds2004/04gmds305.shtml [zuletzt geprüft am 10.11.2005]

Goertzen, R., Stausberg, J. (2005): Ein integriertes System zur computerunterstützten Entwicklung von medizinischen Dokumentationssystemen. Ger Med Sci GM05gmds187. Online-Publikation; http://www.egms.de/en/meetings/gmds2005/05gmds347.shtml [zuletzt geprüft am 10.11.2005]

Goertzen, R., Stausberg, J. (2005): CDMSDesigner – Ein Werkzeug zur Modellierung medizinischer Dokumentationssysteme. Ger Med Sci GM05gmds193. Online-Publikation; http://www.egms.de/en/meetings/gmds2005/05gmds400.shtml [zuletzt geprüft am 10.11.2005]

Inhaltsverzeichnis

1 Einleitung

Bei der Erhebung medizinischer Daten gibt es viele Faktoren, welche die Qualität und damit die Verwertbarkeit der erfassten Informationen beeinflussen. Um das Qualitätsniveau zu sichern, werden auf höchster legislativer Ebene Richtlinien geschaffen, die ein einheitliches Vorgehen und bestimmte Maßnahmen zur Gewährleistung von Vollständigkeit und Validität der Daten definieren (u. a. WHO 1992, EU 1997). Das gesamte Vorgehen von der Konzeption der Erhebung (Anforderungsanalyse) bis zur Schulung des Personals ist hierbei involviert (Campbell, Sweatman 2000). Ziel ist die Sicherstellung einer qualitativ hochwertigen Patientenversorgung (Ammwenwerth, Haux 2000) durch computerunterstützte Dokumentation. Hierfür entscheidend ist unter anderem „die geeignete Verfügbarkeit klinischer Informationen“ (Knaup, Garde 2004).
Laut Patel ist ein wichtiger Bestandteil des Prozesses die Validierung des Datenmaterials und insbesondere, „dass die Daten eine vernünftige Repräsentation dessen bilden, was tatsächlich passierte.“ (Patel 2000)
In einer *Guidance for Industry* der *Food and Drug Administration* (FDA) wird gefordert, dass jedes Erfassungssystem die Erhebung qualitativ hochwertiger Daten durch den Einsatz geeigneter Funktionen unterstützen soll:

> “Prompts, flags, or other help features within the computerized system should be used to encourage consistent use of clinical terminology and to alert the user to data that are out of acceptable range.” (FDA 1999)

Nach Kittlaus ist Software das komplexeste heute bekannte Erzeugnis menschlichen Handelns (Kittlaus 2003). Der Prozess von der Anforderungsanalyse bis zur Fertigstellung und Nutzung der Anwendung ist unter dem Fokus eines durchgängigen Qualitätsmanagements zu sehen, welches Erhebungsfehler weitestgehend unterbindet. Dies lässt sich gewährleisten durch eine frühzeitige, automatisierte Kontrolle der Plausibilität der Daten durch Prüfung von semantischen und logischen Zusammenhängen der Merkmale in Form von Integritätsbedingungen (Mludek 2001). Integritätsbedingungen, das heißt Anforderungen an bestimmte Eigenschaften von Daten, welche direkt oder indirekt die Vollständigkeit und Korrektheit der erhobenen Daten (positiv) beeinflussen, können dabei Auswirkungen auf unterschiedliche Ebenen der Anwendungsarchitektur (Datenschicht, Anwendungslogik oder Präsentationsschicht) haben. Auch hier ist eine übergreifende Betrachtung der Prozesse und der Anforderungen an das medizinische Dokumentationssystem essenziell.

Ansätze für eine derartig stringente Betrachtungsweise wurden durch verschiedene Autoren erarbeitet. Leiner und Haux (Leiner, Haux 1996) propagieren die Verwendung eines Dokumentationsprotokolls, um die systematische Planung von Dokumentationen durch ein definiertes Vorgehensmodell zu unterstützen. Bestandteil des Protokolls ist das konzeptuelle Design des Dokumentationssystems, insbesondere eines Schemas und eines Merkmalskatalogs.
Die Notwendigkeit einer strukturierten Planung wird auch durch Knaup, Garde und Haux (Knaup, Garde 2004) festgestellt. Die Autoren bemängeln, dass „Ansätze zur systematischen Planung klinischer Dokumentation noch immer nicht weit verbreitet und kaum publiziert“ sind und schlagen ein Meta-Modell zur formalen Definition von Dokumentationsumgebungen und deren Beziehungen untereinander vor.
Mludek bezieht in seiner Dissertation den Qualitätsaspekt mit ein und präsentiert ein mit Mitteln der Aussagenlogik formal dargestelltes Modell für Integritätsbedingungen eines Dokumentationssystems zur Studiendokumentation in einem Verbund multizentrischer Therapiestudien der pädiatrischen Onkologie (Mludek 2001).
Im Rahmen des Computer Aided Software Engineering (CASE) werden Anforderungen an Software-Entwicklungsumgebungen gestellt, um das zu Grunde liegende Vorgehensmodell adäquat zu unterstützen (Kelter 1991). Neben Vollständigkeit und Benutzerfreundlichkeit kommt der Integration eine besondere Bedeutung zu, da diese sich auf multiple, heterogene Facetten erstreckt (z. B. Verteilung, Daten, Benutzungsschnittstelle, Überwachung des Prozesses etc.).

Den Aspekt der Integration greift diese Arbeit auf und stellt als Erweiterung der oben vorgestellten Ansätze Konzepte vor, die eine durchgängige und übergreifende Sicherung der Qualität in medizinischen Dokumentationssystemen auf Modell-, Prozess- und Software-Engineering-Ebene unterstützen. In diesem Zusammenhang steht nicht die Datenqualität allein im Fokus der Betrachtung, sondern vielmehr auch eine Qualität des Systems als Ganzes, die durch geeignete Vorgehensmodelle und Techniken bei Entwicklung und Implementierung sicherzustellen ist. Die Ziele sind dabei im Einzelnen:

1. Definition eines Modells zur Beschreibung der Anforderungen an ein medizinisches Dokumentationssystem
2. Entwurf und Implementierung eines *in*tegrierten Systems zur *co*mputerunterstützten Entwicklung *me*dizinischer Dokumentationssysteme (*InCoMe*)
3. Entwurf und Implementierung eines Algorithmus zur Auswertung und Überprüfung komplexer logischer Aussagen

Da auf dem Stand heutiger Software-Technik eine vollständig automatisierte Generierung bei gleichzeitig größtmöglicher Flexibilität der Semantik ohne die Notwendigkeit einer manuellen Kontrolle bzw. Modifikation nicht möglich erscheint, ergibt sich als Limitation für diese Arbeit, dass eine *weitestgehend* maschinelle Unterstützung des Generierungsprozesses angestrebt wird.

Die Dissertation ist wie folgt gegliedert: In Kapitel 2 werden die grundlegenden Konzepte dieser Arbeit eingeführt und die in ihr verwendeten Methoden und Technologien erläutert. Die folgenden drei Kapitel stellen jeweils die erzielten Ergebnisse zu den oben genannten Zielen vor. Zunächst präsentiert Kapitel 3 ein Modell, mit dem medizinische Dokumentationssysteme zur späteren automatisierten Weiterverarbeitung abstrakt formuliert werden können. Es werden die Bausteine vorgestellt, aus denen sich das Abbild eines Dokumentationssystems zusammensetzt, wobei besonders auf die Formulierung von Integritätsbedingungen eingegangen wird. Eine Grammatik definiert die formale Darstellung der Elemente und bildet damit gleichzeitig die Grundlage für das in Kapitel 4 dargestellte integrierte System zur Verarbeitung des Modells *InCoMe*. Nach der Vorstellung der Architektur des entwickelten Systems werden dessen Bestandteile im Einzelnen untersucht und detailliert beschrieben. Auswirkungen und Realisierung der Integritätsanforderungen und -überprüfung behandelt Kapitel 5. Hier bilden die Einflüsse der deklarierten Integritätsbedingungen auf die jeweiligen Schichten der Anwendungsarchitektur einen Schwerpunkt.
Eine Diskussion der in dieser Dissertation erarbeiteten Ergebnisse erfolgt im abschließenden Kapitel 6.

2 Grundlagen und Methoden

In diesem Kapitel werden die relevanten Grundlagen der Arbeit sowie verwendete Methoden und Techniken erläutert. Zuerst wird in Abschnitt 2.1 das Gebiet medizinischer Dokumentation behandelt. Im Anschluss daran folgt eine Betrachtung grundlegender Aspekte der Qualität und Integrität in Abschnitt 2.2. Während sich Abschnitt 2.3 mit den in dieser Arbeit verwendeten Technologien befasst, präsentiert der abschließende Abschnitt 2.4 kurz das zur Evaluation der Ergebnisse herangezogene Praxisbeispiel aus dem Kompetenznetz HIV/AIDS.

2.1 Medizinische Dokumentation

Zunächst werden an dieser Stelle die Begriffe *Dokumentation* und *medizinische Dokumentation* definiert (vgl. Institut für Medizininformatik 2004):

Definition 1: **Dokumentation** bezeichnet die Methoden und Tätigkeiten des Sammelns, Erschließens, Ordnens, Aufbewahrens und gezielten Wiederfindens von Informationen zu spezifischen Frage- oder Aufgabenstellungen.

Definition 2: Unter **medizinischer Dokumentation** versteht man die Dokumentation medizinischer Information oder medizinischen Wissens.

Die medizinische Dokumentation entwickelte sich von ihren Ursprüngen im alten Ägypten über die erste Einrichtung eines Medical Record Department im London des 16. Jahrhunderts bis zur ersten Beschreibung der Aufgaben und Organisation medizinischer Dokumentation in strukturierter Form durch Otto Nacke im Jahr 1960 (Köhler, Meyer zu Bexten 2002, Köhler 2003). Zunehmende Bedeutung bei der Durchführung der Dokumentation bekamen maschinelle Hilfsmittel zu deren Unterstützung, so genannte *medizinische Dokumentationssysteme*.

Definition 3: Ein **medizinisches Dokumentationssystem** ist ein System zur technischen und organisatorischen Unterstützung medizinischer Dokumentation.

Computerbasierte Dokumentationssysteme werden eingesetzt, um den mit der Dokumentation verbundenen (Zeit-)Aufwand zu reduzieren (Ammenwerth, Haux 2000) und die Qualität der dokumentierten Daten zu sichern.

2.2 Qualität und Integrität

Bei der Nutzung medizinischer Dokumentationssysteme hat die *Qualität* der erhobenen Daten einen entscheidenden Einfluss auf deren weitere Verwendbarkeit, so dass diese einen großen Stellenwert bei der Konzeption eines Generierungssystems einnimmt.

Definition 4: „**Qualität**: Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt." (Sens, Fischer 2003)

Der Begriff der Qualität lässt sich dabei sowohl auf das System als Ganzes, als auch auf die in ihm dokumentierten Daten beziehen. Eine adäquate Systemqualität kann durch geeignete Vorgehensmodelle, Methoden und Techniken (siehe Abschnitt 2.3.2) gewährleistet werden.
Um eine determinierte Qualität der Daten in einem System sicherstellen zu können, benötigt der Benutzer adäquate Unterstützung, wie Gaus in seiner Dokumentations- und Ordnungslehre (Gaus 1995) darlegt: „Der Benutzer einer Datendokumentation hat höchst selten die Möglichkeit, ein Datum als fehlerhaft zu erkennen. Ein falscher Wert, ein falscher Befund, kurz ein falsches Datum kann erheblichen Schaden verursachen und ist viel schlechter als gar keine Information. In der Datendokumentation muss die […] Richtigkeitsrate […] sehr nahe bei eins liegen. […] Deshalb ist eine bestmögliche Kontrolle aller Daten vor der Eingabe zwingend notwendig."

Hierzu ist eine eindeutige und vollständige Spezifikation der *Anforderungen* an die Eigenschaften der Daten essenziell:

Definition 5: „**Qualitäts(an)forderung**: Formulierung der Erfordernisse oder deren Umsetzung in eine Serie von quantitativ oder qualitativ festgelegten Forderungen an die Merkmale einer Einheit zur Ermöglichung ihrer Realisierung und Prüfung." (Bundesärztekammer 2004)

Ein wichtiger Begriff der Datenqualität ist die *Integrität* der verwalteten Daten. Eine Definition gibt Zehnder:

Definition 6: „**Datenintegrität** umfasst alle Aspekte, welche das korrekte und zuverlässige Arbeiten mit Datenbanken sicherstellen und unterstützen." (Zehnder 1998)

Die Definition beschränkt ihren Fokus auf das in einer Datenbank abgelegte *Abbild* eines Realitätsausschnittes. Ein technisch korrektes und zuverlässiges Arbeiten aus Sicht einer Datenbank ist aber auch dann möglich, wenn das in ihr geschaffene Abbild nicht mit dem modellierten Realitätsausschnitt übereinstimmt, da die Datenbank selbst eine solche Abweichung zwischen Realität und Datenmodell nicht feststellen kann. Es ist daher sinnvoll, weitere Überlegungen mit einzubeziehen, wie etwa die des Grundschutzhandbuchs des Bundesamtes für Sicherheit in der Informationstechnik (BSI):

Definition 7: „**Integritätskontrolle**: Hierunter fällt die Vermeidung semantischer Fehler bzw. semantisch unsinniger Zustände der Datenbank durch Einhaltung und Überwachung der geforderten Integritätsbedingungen.“ (BSI 2004)

Hier kommt als weitere Ebene die Bedeutung der erfassten Informationen zur Betrachtung. Dabei ist zwischen *Syntax* und *Semantik* zu differenzieren:

Definition 8: Mit **Syntax** wird die Lehre vom Bau größerer sprachlicher Einheiten aus kleineren Einheiten bezeichnet. (vgl. Seelos 1990)

Definition 9: **Semantik** ist das Teilgebiet der Linguistik, das sich mit der Bedeutung sprachlicher Einheiten beschäftigt. (vgl. Seelos 1990)

Die Semantik lässt sich nur sinnvoll auf syntaktisch korrekte Gebilde anwenden, daher ist die syntaktische Korrektheit Voraussetzung für die Überprüfung der Semantik eines Konstrukts und somit implizit in Definition 7 enthalten.

Es wird deutlich, dass Integrität unter vielen unterschiedlichen Aspekten betrachtet werden kann. Die folgende Liste stellt verschiedene Ausprägungen von Integrität dar, die bei der Entwicklung eines medizinischen Dokumentationssystems von Bedeutung sind:

- **physische Integrität**: Existenz sowie Vermeidung von Verlust oder Zerstörung der Daten
- **syntaktische Integrität**: Einhaltung formaler syntaktischer Regeln
- **Konsistenz**: Widerspruchsfreiheit der gespeicherten Daten
- **semantische Integrität**: Vermeidung semantisch unplausibler Zustände der Daten durch Einbeziehung von Bedeutungsrelationen des Realitätsausschnittes
- **Validität**: Grad der Übereinstimmung mit der Wirklichkeit

- **Vollständigkeit**: vollständige Abbildung des relevanten Realitätsausschnittes

Physikalische Integrität lässt sich durch technische oder organisatorische Maßnahmen (z. B. Datensicherung) unabhängig vom jeweils zu Grunde liegenden Dokumentationssystem sicherstellen. Validität und Vollständigkeit eines Systems lassen sich kaum durch eine Datenbank oder durch eine Anwendung überprüfen, sondern sind im Wesentlichen durch den Modellierungsprozess determiniert.
Für die Betrachtungen in dieser Arbeit sind daher nur die Begriffe Konsistenz sowie syntaktische und semantische Integrität relevant, die im Folgenden unter Integrität verstanden werden sollen.

2.3 Verwendete Technologien

In den folgenden Abschnitten werden die jeweils verwendeten Technologien der einzelnen Ergebnisteile (A: *Dokumentationsmodell*, B: *InCoMe - Integriertes System* sowie C: *Integritätsprüfung*) vorgestellt.

2.3.1 Dokumentationsmodell

Für die plattformunabhängige Beschreibung des Modells des zu generierenden medizinischen Dokumentationssystems wurde auf die Sprache *XML* zurückgegriffen. Die Definition der verbindlichen Grammatik für eine solche Modellbeschreibung erfolgte über ein *XML-Schema*.

XML

Die computerunterstützte Generierung eines Dokumentationssystems erfordert den Einsatz eines Beschreibungsmittels, mit dem die Erfordernisse des Ablaufs exakt und weitestgehend systemunabhängig dargestellt werden können. Als ein solches Beschreibungsmittel bietet sich die *Extensible Markup Language* (XML) an, welche als Untermenge der *Standard Generalized Markup Language* (SGML) Methoden und Mittel zur Strukturierung und Beschreibung von Daten deklariert.
Die XML-Spezifikation stellt eine Grammatik zur Verfügung, die wohlgeformte XML-Dokumente definiert. Wohlgeformte XML-Dokumente entsprechen hinsichtlich ihres Aufbaus bestimmten, festgelegten Kriterien.
Die logischen Elemente eines XML-Dokuments werden durch so genannte Tags umschlossen, wobei XML, anders als zum Beispiel HTML, keine vordefinierte Menge von Tags bereitstellt. Die Menge der für ein konkretes Dokument gültigen Tags wird individuell in einer Document Type Definition (DTD) oder in einem XML-Schema beschrieben.

XML-Schema

In dieser Arbeit wird die Grammatik des Dokumentationsmodells mit Hilfe eines XML-Schemas formuliert. XML-Schemata folgen der dreistufigen Spezifikation des World Wide Web Consortium (W3C), in der es heißt:

> „XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents." (W3C 2001)

Das erstellte Dokumentationsmodell lässt sich gegen das definierte XML-Schema validieren. Ein valides Dokumentationsmodell folgt allen im Schema definierten Regeln.

2.3.2 *InCoMe* - Entwurfsmodul

Das Entwurfsmodul unterstützt die Erstellung eines Dokumentationssystemmodells über eine grafische Benutzeroberfläche. Neben der bei der Implementierung verwendeten Programmiersprache *Java* werden in diesem Abschnitt die angewendeten Modellierungs- und Entwicklungstechniken *UML*, *Entwurfsmuster*, *Architekturframework* und *Namenskonventionen* vorgestellt.

Java

Als Programmiersprache für die Implementierung des Entwurfsmoduls diente Java (Sun Microsystems 2004a) in der Version SDK 1.4.2. Java wurde 1991 als objektorientierte Programmiersprache von Sun Microsystems entwickelt. Ihr Hauptmerkmal ist die Plattformunabhängigkeit des ausführbaren Codes. Zur Umsetzung der grafischen Benutzeroberfläche diente die JFC/Swing-Komponentenbibliothek.

UML

Die Unified Modeling Language (UML) (Oestereich 1998) ist eine von den Modellierungspionieren Grady Booch, James Rumbaugh und Ivar Jacobson entwickelte, einheitliche Modellierungsnotation und –semantik. Mit Hilfe der Diagrammtypen Anwendungsfalldiagramm, Klassendiagramm, Aktivitätsdiagramm, Kollaborationsdiagramm, Sequenzdiagramm, Zustandsdiagramm, Komponentendiagramm und Verteilungsdiagramm lassen sich Strukturen und Abläufe objektorientierter Softwaresysteme beschreiben. Im Folgenden werden einige wichtige Begriffe der UML definiert (vgl. Oesterreich 1998).

Definition 10: Ein **Businessobjekt** (Business Object) repräsentiert einen Gegenstand, ein Konzept, einen Ort oder eine Person aus dem realen Geschäftsleben in

einem fachlichen geringen Detaillierungsgrad, d. h. einen fachlich elementaren Begriff.

Definition 11: **Pakete** (Packages) sind Ansammlungen von Modellelementen beliebigen Typs, mit denen das Gesamtmodell in kleinere, überschaubare Einheiten gegliedert wird.

Definition 12: **Schnittstellen** (Interfaces) beschreiben einen ausgewählten Teil des extern sichtbaren Verhaltens von Modellelementen, d. h. eine Menge von Signaturen.

Definition 13: Ein **Rahmenwerk** (Framework) ist eine Menge kooperierender Klassen, die unter Vorgabe eines Ablaufes eine generische Lösung für eine Reihe ähnlicher Aufgabenstellungen bereitstellen.

Vorgehensmodell

Als Vorgehensmodell dienten die Objektorientierte Analyse und das Objektorientierte Design (OOA/OOD) nach Booch (Booch 1994). Der von ihm entwickelte Makroprozess, der dem klassischen Wasserfallmodell (Royce 1970) sehr ähnelt, beinhaltet die Phasen

- Konzeptualisierung,
- Analyse,
- Design,
- Evolution sowie
- Wartung.

Durch dieses Vorgehensmodell wird eine Fortschrittskontrolle durch Meilensteine ermöglicht und dadurch die Planung des Entwicklungsprozesses wesentlich erleichtert.

Entwurfsmuster (Design Patterns)

Bei Entwurf und Implementierung des Systems wurde intensiv mit Entwurfsmustern gearbeitet. Entwurfsmuster dienen der Konservierung bewährter Lösungsansätze und unterstützen damit Robustheit, Erweiterbarkeit und Wiederverwendbarkeit von Software:

> „A design pattern is a description of communicating objects and classes that are customized to solve a general design problem in a particular context. [..] Each design pattern lets some aspect of system structure vary independently of other aspects, thereby making a system more robust to a particular kind of change.” (Gamma, Helm 1995)

Der Wiederverwendungsansatz wird besonders deutlich in der folgenden Definition:

> „Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." (Alexander, Ishikawa 1977)

Exemplarisch werden im Folgenden die beiden verwendeten Entwurfsmuster *Model-View-Controller* und *Factory* vorgestellt.

Beim Model-View-Controller-Entwurfsmuster (Buschmann, Meunier 1996) handelt es sich um ein Architekturmuster, bei dem die Elemente eines Systems in die drei Bestandteile Datenhaltung (Model), Präsentation (View) und Anwendungs-/Kontrollschicht (Controller) zerlegt werden:

> „The Model-View-Controller architectural pattern (MVC) divides an interactive application into three components. The model contains the core functionality and data. Views display information to the user. Controllers handle user input. Views and controllers together comprise the user interface."

Hauptvorteile der Verwendung dieses Musters sind die erhöhte Wiederverwendbarkeit der entwickelten Komponenten sowie eine leichte Eingliederung in Architekturframeworks.
Das Factory-Entwurfsmuster liefert als Erzeugungsmuster die Instanz einer Klasse zurück und erhöht so in Verbindung mit dem Einsatz von Schnittstellen die Modularität. Dem Aufrufer der Factory muss nur die benötigte Schnittstelle bekannt sein, nicht die sie implementierende, konkrete Klasse. Auf diese Weise lassen sich Komponenten flexibel austauschen, ohne dass Anpassungen in allen aufrufenden Programmteilen notwendig werden.

Zur detaillierten Darstellung von Design Patterns, deren Nutzen und Gebrauch sei hier auf die zuvor genannte Literatur verwiesen.

Architekturframework

Im Rahmen der Implementierung dieser Arbeit wurde - speziell zur Realisierung des MVC-Entwurfsmusters - ein Architekturframework implementiert, das im Paket `gui` zusammengefasst ist. Da es sich bei den dort entwickelten Klassen um allgemein (wieder-) verwendbare Elemente handelt, die keinen ausschließlichen Bezug zu der hier betrachteten Anwendung haben, wird auf eine ausführliche Darstellung verzichtet. Die in diesem Paket enthaltenen Klassen und Interfaces sind im Anhang aufgelistet.

Design der Grafischen Benutzeroberfläche

Eine Grafische Benutzeroberfläche (Graphical User Interface – GUI) ist die visuelle Schnittstelle von Computerprogrammen, über welche der Benutzer mit diesen in Interaktion treten kann. Obwohl der Fokus bei der Implementierung nicht auf dem GUI-Design lag, da die Verwendung des Entwurfsmoduls nicht durch einen Endbenutzer erfolgt, wurden dennoch weitestgehend die in DIN EN ISO 9241-10 (DIN 1996) festgelegten Grundsätze der Dialoggestaltung (Aufgabenangemessenheit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlerrobustheit, Individualisierbarkeit und Lernförderlichkeit) befolgt sowie insbesondere die Portierbarkeit der Anwendung sichergestellt.

Namenskonventionen

Durch die Verwendung von Namenskonventionen erhöhen sich Wartbarkeit und Erweiterbarkeit eines Systems enorm. Zur Unterscheidung der Stellung verschiedener Klassen im System wurden Regeln verwendet, die im Folgenden anhand von Beispielen erläutert werden:

- `Entity`**`Home`**: Factory-Klasse eines Pakets (Factory-Pattern); enthält alle Instanziierungsmethoden für dieses Paket
- `Context`: Schnittstelle eines Businessobjekts (BO); deklariert alle Methoden, die durch andere Klassen referenzierbar sind
- `Context`**`Base`**: Basisklasse eines BO; enthält alle Zugriffsmethoden auf Attribute (Bean-Pattern)
- `Context`**`Impl`**: Implementierungsklasse eines BO; enthält alle fachlichen Methoden
- `ContextEditor`**`Ctrl`**: Controller-Klasse; enthält alle zur Steuerung von Aktionen auf BO notwendigen Elemente
- `ContextEditor`**`View`**: View-Klasse; enthält alle Elemente der grafischen Repräsentation eines BO
- `ConditionEditor`**`Dial`**: Dialog-Klasse; wie View, aber in eigenem Dialog-Fenster
- `CDMS`**`Servlet`**: von `HttpServlet` abgeleitete Klasse zur zentralen Steuerung des Request/Response-Paradigma

2.3.3 *InCoMe* - Generierungsmodul

Das Generierungsmodul muss Mittel zur Verfügung stellen, mit denen es möglich ist, ein Dokumentationsmodell in andere Darstellungen zu transformieren. Dies geschieht mittels *XSLT*.

XSLT

XSL Transformations (XSLT) (W3C 1999) definiert eine Sprache, mit der Regeln zur Transformation eines XML-Dokumentes formuliert werden können. Die Menge der Regeln wird als Stylesheet bezeichnet. Knoten des Ursprungsdokumentenbaumes werden durch Abbildungsregeln, welche Beziehungen von Mustern (patterns) zu Ersetzungsvorlagen (templates) deklarieren, in ein neues Dokument überführt.

2.3.4 *InCoMe* - Applikationsframework

Grundsätzlich ist es mit dem entwickelten Ansatz möglich, die Implementierung des medizinischen Dokumentationssystems in einer beliebigen Programmiersprache umzusetzen. Als Beispiel für eine flexible Implementierung wurde ein System aus *Applikationsserver*, *Servlets*, *Java Server Pages*, *Struts* sowie *relationalen Datenbanken* verwendet, auf dessen Basis ein integratives Framework bereitgestellt wurde. Dieses ist in der Lage, jeden Entwurf eines Dokumentationssystems ohne die Notwendigkeit einer Quellcode-Kompilierung direkt in eine lauffähige Applikation zu transformieren.

Applikationsserver

Grundlage des Applikationsframeworks ist der im Rahmen des Apache Jakarta Projektes entwickelte und als Open-Source-Software verfügbare Servlet-Container Tomcat (Jakarta 2004). Tomcat stellt die offizielle Referenzimplementierung der im Folgenden vorgestellten, von Sun Microsystems entwickelten Technologien der Servlets und Java Server Pages dar. Deren Zusammenwirken wird in Abbildung 1 präsentiert.

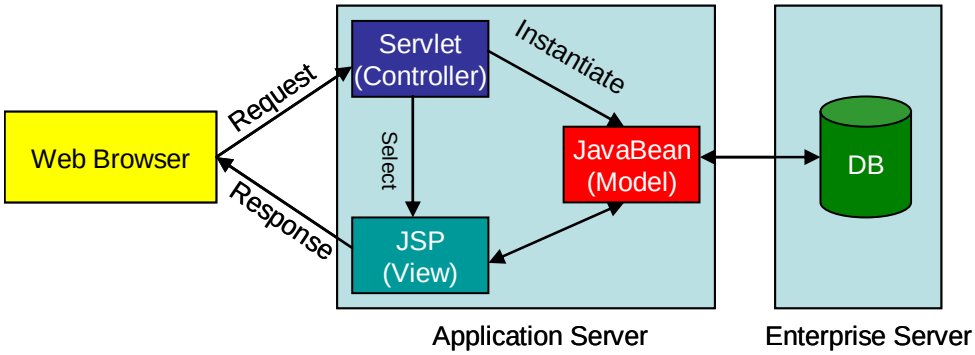


Abbildung 1: Servlets und JSP (aus Mahmoud 2003)

Servlets

Servlets (Sun 2004c) sind in Java implementierte, plattform- und protokollunabhängige Programme zur dynamischen Erweiterung von Server-Funktionalität. Das Servlet-Framework, welches dem Request/Response-Paradigma folgt, unterstützt dabei die interaktive Generierung von Web-Seiten.

Java Server Pages (JSP)

Java Server Pages (JSP) (Sun 2004b) stellen eine Technologie zur Verknüpfung statischer HTML-Seiten mit dynamischen Komponenten bereit. In Kombination mit der Servlet-Technik wird der Einsatz des Model-View-Controller-Entwurfsmusters ermöglicht und damit unter anderem die Wartbarkeit entschieden verbessert. Die Präsentationsschicht der Anwendung lässt sich unabhängig von den anderen Schichten zur Laufzeit anpassen.

Struts

Auch Struts (Apache 2004a), ein Open-Source-Framework für die Implementierung von Web-Applikationen, folgt dem MVC-Entwurfsmuster. Zur Unterstützung der dynamischen Struktur der Applikation wurden Teile dieses Frameworks, unter anderem Action-Mapping und Templates, bei der Implementierung verwendet. Abbildung 2 zeigt die Funktionsweise von Struts.

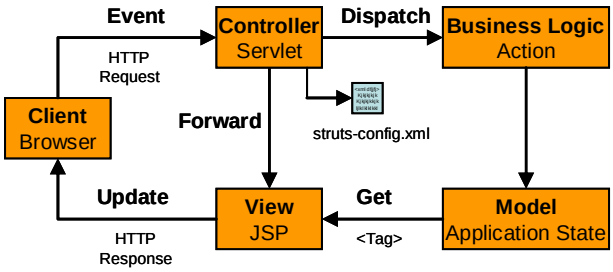


Abbildung 2: Struts (aus Davis 2001)

Relationale Datenbanken

Relationale Datenbanken verwalten Daten basierend auf dem relationalen Datenmodell von Edgar Codd in Form von zweidimensionalen Tabellen (Codd 1970). Zum Einsatz kamen beispielhaft die Datenbanken der Firmen MySQL und Oracle. Während die relationale Datenbank von Oracle eine kommerzielle und damit kostenpflichtige Möglichkeit der Datenhaltung darstellt, ist MySQL eine frei verfüg- und verwendbare Software unter Open-Source-Lizenz.

2.3.5 Integritätsprüfung

Bei der Umsetzung insbesondere der semantischen Integritätsprüfung ergaben sich verschiedene Optionen, mit welchen Mitteln und auf welcher Architekturebene diese durchgeführt werden sollte.

Der nahe liegende Weg wäre es, alle nicht-darstellungsbezogenen Integritätsbedingungen zentral auf Ebene der Datenhaltung, also durch das zu Grunde liegende Datenbank-

Management-System (DBMS) durchführen zu lassen. Im Grundschutzhandbuch des Bundesamtes für Sicherheit in der Informationstechnik heißt es:

> „Die Definition von Triggern oder Stored Procedures erfolgt im Rahmen der Realisierung des Fachkonzepts. Trigger und Stored Procedures können dabei sowohl innerhalb der Anwendung (in den Programmen), als auch der Datenbank (für Tabellen) Verwendung finden.“ (BSI 2004)

Obwohl der für relationale Datenbanken gebräuchliche SQL-Standard bereits im Jahr 1992 hierzu sogenannte *Assertions* definiert hatte, sind diese bislang in den wenigsten DBMS umgesetzt – so weder in Oracle noch in MySQL. In Letzterer fehlt auch bisher das Konzept der Trigger, welches ebenfalls Mittel zur Einhaltung der Datenplausibilität bereitstellt.

Aus diesem Grund wurde die Integritätsprüfung auf Ebene der Anwendungslogik realisiert. Auf den Einsatz einer der Prädikatenlogik folgenden, zusätzlich einzubindenden Programmiersprache wie z. B. Prolog (Belli 1988) wurde zu Gunsten der Flexibilität und Portierbarkeit bewusst verzichtet. Stattdessen wurde ein komplexer Algorithmus entwickelt, der die Korrektheit der Daten innerhalb des Frameworks überprüft und in Kapitel 5 vorgestellt wird.

2.4 Anwendungsbeispiel Kompetenznetz der Medizin HIV/AIDS

Beim Kompetenznetz HIV/AIDS handelt es sich um einen vom Bundesministerium für Bildung und Forschung geförderten Forschungsverbund wesentlicher deutscher, im Bereich der klinischen Forschung und kliniknahen Grundlagenforschung zu HIV und AIDS tätigen Einrichtungen. Hauptziel ist „der Aufbau einer national organisierten, umfassenden und repräsentativen, prospektiven Klinischen Kohorte“ (IMIBE 2004a). Insbesondere ist durch diese Arbeit folgende geforderte Funktion abgedeckt:

> „Dokumentation klinischer Verlaufs- und Behandlungsdaten zur Bearbeitung epidemiologischer Fragestellungen und als Instrument zur klinischen Surveillance; diese Daten werden durch die beteiligten Zentren mit Hilfe der Telematik-Plattform jeweils kontinuierlich aktualisiert.“ (ebd.)

Am Institut für Medizinische Informatik, Biometrie und Epidemiologie in Essen wurden die Konzepte von Leiner und Haux (Leiner, Haux 1996) auf den Dokumentationsprozess angewendet und daraus Schema und Merkmalskatalog der im Kompetenznetz zu erhebenden Daten entwickelt. Auf dieser Grundlage wurden verschiedene Möglichkeiten untersucht, die Dokumentation maschinell zu unterstützen, unter anderem

- die Studiensoftware MACRO der Firma InferMed, die trotz einiger Schwächen (Nonnemacher, Weiland 2005) produktiv zur Datenerhebung im Kompetenznetz eingesetzt wird, und
- eine generische Lösung, bei der ein auf Basis von Microsoft Access erstellter Merkmalskatalog zur automatisierten Erzeugung eines Oracle-Datenbankschemas genutzt wurde. Diese Lösung unterstützt jedoch keine Einbindung von Integritätsbedingungen.

Aus dem entwickelten, modularen Dokumentationskonzept wurde das Basismodul Erwachsene (IMIBE 2004b) in der Version 1.6 als Anwendungsbeispiel verwendet, um die Praktikabilität des mit dieser Arbeit vorgestellten Ansatzes zu überprüfen.

3 Ergebnisse Teil A: Dokumentationsmodell

3.1 Einleitung

Die computerunterstützte Generierung eines medizinischen Dokumentationssystems setzt voraus, dass eine vollständige Beschreibung der Anforderungen in Form eines *Modells* möglich ist, welches sich zu einer automatisierten Weiterverarbeitung eignet.

Definition 14: „**Modell**: Vereinfachung der Realität, die erstellt wird, um das zu entwickelnde System besser zu verstehen; semantisch geschlossene Abstraktion eines Systems.“ (Grässle, Baumann 2000)

Damit können wir folgende Definition für ein *Dokumentationsmodell* formulieren:

Definition 15: Ein **Dokumentationsmodell** ist die semantisch geschlossene, abstrakte Beschreibung eines Dokumentationssystems.

In diesem Kapitel wird eine allgemeingültige Grammatik für Dokumentationsmodelle schrittweise entwickelt und definiert. Der folgende Abschnitt 3.2 präsentiert zunächst die im Rahmen dieser Arbeit ermittelten Bausteine, aus denen sich das Dokumentationsmodell zusammensetzt. Wegen ihrer besonderen Bedeutung werden die modellierten Ausprägungen der Integritätsbedingungen separat im Abschnitt 3.3 behandelt. Abschnitt 3.4 fasst die Erkenntnisse zusammen und zeigt schematisch das Gesamtmodell eines medizinischen Dokumentationssystems.

3.2 Elemente

Das Modell eines medizinischen Dokumentationssystems setzt sich aus hierarchisch gegliederten Elementen zusammen. In diesem Abschnitt folgt nach der jeweiligen Definition der Modellbausteine eine Aufstellung der zugehörigen Attribute, bevor die Verwendung des Elements durch ein Beispiel verdeutlicht wird. Im Anschluss daran findet sich jeweils eine formale Darstellung der Modellgrammatik des Bausteins sowie dessen Veranschaulichung ebenfalls anhand eines Beispiels. Die vollständige Definition der Modellgrammatik befindet sich im Anhang dieser Arbeit.

3.2.1 Entitäten

Da es Eigenschaften auf Modellebene gibt, die allen Modellelementen innerhalb der Applikation gemeinsam sind, ist es sinnvoll, diese im Sinn der Wiederverwendung in eine modulare Einheit zusammen zu fassen, die als *Entität* bezeichnet sei.

Definition 16: Eine **Entität** ist die Repräsentation eines (konkreten oder abstrakten) Gegenstands, der für ein gegebenes Anwendungssystem von Bedeutung ist (vgl. Hesse, Barkow 1994).

Eigenschaften der untersuchten Gegenstände, welche für die Betrachtung relevant sind, werden zusammen mit den Entitäten modelliert und als *Attribute* bezeichnet.

Definition 17: Ein **Attribut**(-typ) ist eine Zuordnungsvorschrift, die jeder Entität einer Entitätsmenge zu jedem Zeitpunkt jeweils höchstens ein Element aus einem Wertebereich zuordnet (vgl. Hesse, Barkow 1994).

Attribute:

- `id`: eindeutiger Identifizierer der Entität
- `label`: Bezeichner der Entität
- `description`: optionale Beschreibung der Entität
- `name`: innerhalb einer Domäne eindeutiger Name der Entität, insbesondere zur Verwendung im generierten System.[1]

Beispiel einer Entität:

Die Entität (Merkmalsart) *Vitalstatus* wird beschrieben durch die Attribute:

- `id`: 1077068082972
- `label`: Vitalstatus
- `description`: Vitalstatus des Patienten
- `name`: Vitalstatus

Eine Entität ist ein abstraktes Element, welches selbst nicht zur Modellierung herangezogen werden kann. Als Baukasten für ein medizinisches Dokumentationssystem stehen stattdessen die im Folgenden vorgestellten konkreten Elemente bereit, die sich von *Entität* ableiten, d. h. deren Eigenschaften (Attribute) erben. Abbildung 3 gibt einen Überblick über die von *Entität* abgeleiteten Elemente

- freie Merkmalsart (Attribute),
- gebundene Merkmalsart (ContextAttribute),

[1] Domäne der Merkmalsarten ist der Kontext, die der Kontexte ist die Anwendung. Integritätsbedingungen können den Domänen Merkmalsart, Kontext oder Anwendung zugeordnet sein. Die Begriffe Anwendung, Kontext und Merkmalsart werden in den folgenden Abschnitten vorgestellt.

- Kontext (Context),
- Datenbank (Database),
- Stammdaten (MasterData) sowie
- Integritätsbedingung (Constraint).

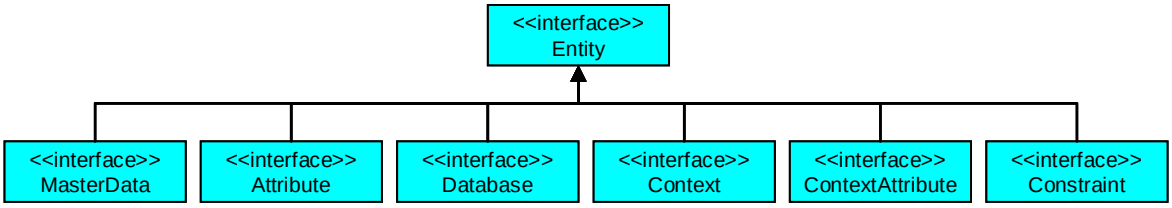


Abbildung 3: Modellelemente

3.2.2 Merkmalsarten

Zentrale Bedeutung für die Modellierung eines Dokumentationssystems hat die Menge der darin zu dokumentierenden Informationen. Diese werden durch *Merkmalsarten* definiert.

Definition 18: Ein im Rahmen einer Dokumentation interessierendes Objekt wird als **Merkmalsträger** bezeichnet (vgl. Schwarze 1994).

Definition 19: Eine **Merkmalsart** ist die Eigenschaft eines oder mehrerer Merkmalsträger, die im Rahmen einer Dokumentation erhoben werden soll.

Definition 20: Eine **Merkmalsausprägung** ist die zu einer Merkmalsart dokumentierte Menge von Werten.

Definition 21: Ein **Merkmal** ist die Vereinigungsmenge genau einer (gebundenen) Merkmalsart und der Menge der zu dieser Merkmalsart dokumentierten Merkmalsausprägungen.

Verlegt man den Modellierungs- und Begriffsfokus von der Meta-Ebene auf die Ebene des zu modellierenden medizinischen Dokumentationssystems, so besteht eine Synonymität der Begriffe Entität-Merkmalsträger sowie Attribut-Merkmalsart im Sinne der Entity-Relationship-Modellierung (Chen 1976). Unter Einbeziehung des Ansatzes relationaler Datenmodellierung nach Chen hat auch im hier entwickelten Modell (abweichend zu Mludek 2001) eine Merkmalsausprägung höchstens ein Element, da der zur Vermeidung von Anomalien und Redundanzen geforderte Normalisierungsprozess zwangsläufig zu einer Modellierung getrennter Kontexte führt (Codd 1970).

Im Sinn des hier vorgestellten Modells ist zwischen freien und gebundenen Merkmalsarten zu unterscheiden. Eine **freie Merkmalsart** ist eine Einheit, die bestimmte übergreifende Eigenschaften definiert, welche in mehreren Kontexten relevant sein können (siehe folgender Abschnitt). Eine **gebundene Merkmalsart** dagegen ist einem Kontext direkt zugeordnet. Sie kapselt die Eigenschaften der freien Merkmalsart. Durch diese Vorgehensweise wird eine Wiederverwendung von bereits erhobenen Informationen ermöglicht und damit der Modellierungsaufwand erheblich reduziert.

Freie Merkmalsarten

Attribute freier Merkmalsarten:

- `contextReferences`: Menge der Kontexte, in denen diese Merkmalsart als gebundene Merkmalsart Verwendung findet
- `constraints`: Menge der für die freie Merkmalsart gültigen Integritätsbedingungen

Beispiel einer freien Merkmalsart:

Die Postleitzahl eines Ortes in Deutschland hat bestimmte Eigenschaften (z. B. nummerischer Wertebereich, Datenlänge: fünf Zeichen), die in der freien Merkmalsart *PLZ* zusammengefasst werden.

Definition 22: Modellgrammatik freier Merkmalsarten

```
<xsd:element name="attributes">
	<xsd:complexType>
		<xsd:sequence>
			<xsd:element name="attribute“
				minOccurs="1"
				maxOccurs="unbounded"
				type="attributeType"/>
		</xsd:sequence>
	</xsd:complexType>
</xsd:element>

<xsd:complexType name="attributeType">
	<xsd:sequence>
		<xsd:element ref="constraint"
			minOccurs="1"
			maxOccurs="unbounded" />
	</xsd:sequence>
	<xsd:attribute name="id"
		type="xsd:ID"
		use="required"/>
	<xsd:attribute name="label"
		type="xsd:string"
		use="required" />
```

```
        <xsd:attribute name="description"
                type="xsd:string"
                use="optional"/>
</xsd:complexType>
```

Beispiel für das Modell einer freien Merkmalsarten: Merkmalsart *Aufenthaltsstatus*

```
<attributes>
        <attribute     id="Attribute1077065314515"
                       label="Aufenthaltsstatus"
                       description="">
                …
        </attribute>
        …
</attributes>
```

Gebundene Merkmalsarten

Attribute gebundener Merkmalsarten:

- `attribute`: zugehörige freie Merkmalsart (als *Referenz*, siehe Modellgrammatik)
- `parent`: Kontext, dem diese Merkmalsart zugeordnet ist.
- `constraints`: Menge der für die gebundene Merkmalsart gültigen Integritätsbedingungen

Das Attribut `parent` wird verwendet, um eine im Dokumentationsmodell implizit modellierte Beziehung in eine Assoziation mit Referenzattributen im Sinn der Unified Modeling Language umzuwandeln.

Beispiel einer gebundenen Merkmalsart:

Die freie Merkmalsart *PLZ* findet Verwendung in den Kontexten *Patientenadresse* sowie *Klinikadresse* und wird somit dort jeweils zu einer gebundenen Merkmalsart. Der gebundenen Merkmalsart können zusätzliche Eigenschaften (in Form von Integritätsbedingungen) zugewiesen werden, beispielsweise, dass die bei einer Erhebung eingegebene Klinikpostleitzahl stets dem Wert ‚45122' entsprechen muss.

Definition 23: Modellgrammatik gebundener Merkmalsarten

```
<xsd:element name="contextattribute"
        type="contextattributeType"
        minOccurs="0"
        maxOccurs="unbounded" />

<xsd:complexType name="contextattributeType">
        <xsd:sequence>
                <xsd:element ref="constraint"
```

```
                    minOccurs="1"
                    maxOccurs="unbounded" />
        </xsd:sequence>
        <xsd:attribute name="id"
                type="xsd:ID"
                use="required"/>
        <xsd:attribute name="label"
                type="xsd:string"
                use="required" />
        <xsd:attribute name="name"
                type="xsd:NMTOKEN"
                use="required" />
        <xsd:attribute name="description"
                type="xsd:string"
                use="optional"/>
        <xsd:attribute name="reference"
                type="xsd:IDREF"
                use="required" />
</xsd:complexType>
```

Beispiel für das Modell einer gebundenen Merkmalsart: *Pseudonym* im Kontext *Patient*

```
<contextattribute     id="ContextAttribute1077067928377"
                      label="Pseudonym"
                      name="Pseudonym"
                      description="Pseudonym des Patienten"
                      reference="Attribute1077065370015">
                …
</contextattribute>
```

3.2.3 Kontexte

Merkmalsarten können in semantischer Beziehung zueinander stehen. Die gemeinsame Zugehörigkeit zu einem Merkmalsträger wäre eine solche mögliche Beziehung. Um semantische Gruppen von Merkmalsarten zusammenfassen zu können, gibt es das Element *Kontext*.

Definition 24: Ein **Kontext** bezeichnet eine Menge von semantisch zusammengehörenden Merkmalsarten.

Ein Kontext bündelt also freie Merkmalsarten zu Einheiten und bindet sie dadurch an eine semantische Bedeutung.

Attribute:

`database`: Anwendung, zu welcher der Kontext gehört

`attributes`: Menge der im Kontext enthaltenen Merkmalsarten

`constraints`: Menge der für den Kontext gültigen Integritätsbedingungen

Beispiel:

Der Kontext *Meldung* setzt sich aus den Merkmalsarten *ID*, *Meldedatum*, *Patient*, *ID_Einrichtung*, *Vitalstatus*, *Erfassungsdatum* und *Erfassende_Person* zusammen.

Definition 25: Modellgrammatik Kontext

```
<xsd:element name="context"
	type="contextType" />

<xsd:complexType name="contextType">
	<xsd:sequence>
		<xsd:element ref="constraint"
			minOccurs="1"
			maxOccurs="unbounded"/>
		<xsd:element ref="contextattribute"
			minOccurs="0"
			maxOccurs="unbounded"/>
	</xsd:sequence>
	<xsd:attribute name="id"
		type="xsd:ID"
		use="required"/>
	<xsd:attribute name="label"
		type="xsd:string"
		use="required" />
	<xsd:attribute name="name"
		type="xsd:NMTOKEN"
		use="required" />
	<xsd:attribute name="description"
		type="xsd:string"
		use="optional"/>
</xsd:complexType>
```

Beispiel für das Modell eines Kontexts: Ausschnitt aus Kontext *Meldung*

```
<context id="Context1077068009640"
	label="Meldung"
	name="Meldung"
	description="Vorliegende Meldungen">
	…
	<contextattribute id="ContextAttribute1077068082968"
		label="ID"
		description="Interne ID für Kohorte"
		name="ID"
		reference="Attribute1077065262656">
		…
	</contextattribute>
	<contextattribute id="ContextAttribute1077068082969"
		label="Meldedatum"
```

```
                description=""
                name="Meldedatum"
                reference="Attribute1077065964093">
                …
        </contextattribute>
        <contextattribute id="ContextAttribute1077068082970"
                label="Patient"
                description=""
                name="Patient"
                reference="Attribute1077065982312">
                …
        </contextattribute>
        …
</context>
```

3.2.4 Datenbank

Auch Kontexte stehen in Beziehung zueinander. Zur Ablage semantisch zusammengehörender Kontexte benötigt man eine *Datenbank*.

Definition 26: „Eine **Datenbank** ist eine selbständige, auf Dauer und für flexiblen und sicheren Gebrauch ausgelegte Datenorganisation, umfassend einen Datenbestand (Datenbasis) und die dazugehörige Datenverwaltung.“ (Bauknecht, Zehnder 1997)

Diese technische Definition ist sehr allgemein gehalten, daher wird der Begriff der Datenbank im Rahmen der Modellierung wie folgt definiert:

Definition 27: Eine **Datenbank** im Sinn dieser Arbeit ist die Menge der Kontexte und gebundenen Merkmalsarten eines medizinischen Dokumentationssystems.

Die Datenbank fasst alle entworfenen Kontexte zu einer Einheit zusammen und enthält somit alle zum System modellierten, individuellen Informationen. Übergreifende, wieder verwendbare Informationen wie freie Merkmalsarten und die im folgenden Abschnitt vorgestellten Stammdaten werden hier nicht zur Datenbank gezählt, sondern mit dieser unter dem Begriff der *Applikation* (siehe 3.2.7 Applikation) zusammengefasst.

Attribute:

`contexts`: Menge der Kontexte der Anwendung

`constraints`: Menge der für die Anwendung gültigen Integritätsbedingungen

Beispiel:

Die Datenbank *Hivnet* besteht aus den Kontexten *Alkoholgenuss*, *Befund*, *Diagnose*, *Diarrhoe*, *Drogenkonsum*, *ErrNachweis*, *Gewichtsverlust*, *HistologieLeber*, *HIVAnamnese*, *HIVSubtyp*, *Kinder*, *Kontakt*, *Material*, *Medikation*, *Meldung*, *Mikrobiologie*, *Prozedur*, *Rauchen*, *Schwangerschaft*, *Sozialstatus*, *Symptom*, *Todesbeschein*, *Todesursache*, *Tumor*, *Tumortherapie*, *Veraenderung* sowie *VStatus*.

Definition 28: Modellgrammatik Datenbank

```
<xsd:element name="database"
        type="databaseType" />

<xsd:complexType name="databaseType">
        <xsd:sequence>
                <xsd:element ref="constraint"
                        maxOccurs="1"/>
                <xsd:element ref="context"
                        minOccurs="0"
                        maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="id"
                type="xsd:ID"
                use="required"/>
        <xsd:attribute name="label"
                type="xsd:string"
                use="required" />
        <xsd:attribute name="name"
                type="xsd:NMTOKEN"
                use="required" />
        <xsd:attribute name="description"
                type="xsd:string"
                use="optional"/>
</xsd:complexType>
```

Beispiel für das Modell einer Datenbank: Ausschnitt der Datenbank *Hivnet*

```
<database id="Database1077067859515"
        label="hivnet"
        description="Dokumentationssystem auf Grundlage des Merkmalskatalogs HIVNET"
        name="hivnet">
        <context id="Context1077067859500"
                label="Patient"
                name="Patient"
                description="Patientendaten">
                <contextattribute id="ContextAttribute1077067928375"
                        label="ID"
                        description="Interne ID für Kohorte"
                        name="ID"
                        reference="Attribute1077065262656">
```

```
                </contextattribute>
                …
        </context>
        <context id="Context1077067936703"
                label="Befund"
                description="Befunddaten"
                name="Befund">
        </context>
        …
</database>
```

3.2.5 Stammdaten

Bei einigen Merkmalsarten erscheint es sinnvoll, die Menge der möglichen Merkmalsausprägungen einzuschränken. Hierzu können *Stammdaten* genutzt werden.

Definition 29: Als **Wertebereich** bezeichnet man die Menge von möglichen Werten, die zur Beschreibung einer Eigenschaft herangezogen werden können (vgl. Hesse, Barkow 1994).

Definition 30: **Stammdaten** definieren einen Wertebereich, der als finite Menge alle möglichen Ausprägungen einer Merkmalsart beschreibt.

Mit Hilfe von Stammdaten lassen sich also distinkte Wertebereiche definieren, die zur Einschränkung der erlaubten Merkmalsausprägungen bei der Modellierung einer Merkmalsart herangezogen werden können. Durch ihren Einsatz kann die Gefahr von Fehleingaben reduziert und damit die Integrität der Daten erhöht werden. Ihre Einbeziehung in das Modell erfolgt in Form einer Integritätsbedingung (siehe Abschnitt 3.2.6) auf Ebene der jeweiligen Merkmalsart.

Attribute:

- `values`: Menge der möglichen Ausprägungen

Beispiel:

Der Wertebereich der gebundenen Merkmalsart *Vitalstatus* im Kontext *Patient* wird definiert als endliche Menge

{ als verstorben bekannt; mit Datum |
als verstorben bekannt; ohne Datum |
lost to follow up (ltf) |
ltf: zurück ins Heimatland |
ltf: Arztwechsel |
als lebend bekannt |

nicht als verstorben bekannt (unbekannt) }

und bezeichnet als *Stammdaten_Vitalstatus*.

Definition 31: Modellgrammatik Stammdaten

```
<xsd:element name="masterdata" >
	<xsd:complexType>
		<xsd:sequence>
			<xsd:element name="table“
				minOccurs="1"
				maxOccurs="unbounded"
				type="tableType"/>
		</xsd:sequence>
	</xsd:complexType>
</xsd:element>

<xsd:complexType name="tableType">
	<xsd:sequence>
		<xsd:element ref="row"
			minOccurs="0"
			maxOccurs="unbounded"/>
	</xsd:sequence>
	<xsd:attribute name="id"
		type="xsd:ID"
		use="required"/>
	<xsd:attribute name="label"
		type="xsd:string"
		use="required" />
	<xsd:attribute name="name"
		type="xsd:NMTOKEN"
		use="required" />
	<xsd:attribute name="description"
		type="xsd:string"
		use="optional"/>
</xsd:complexType>

<xsd:element name="row">
	<xsd:complexType>
		<xsd:attribute name="id"
			type="xsd:NMTOKEN"
			use="required" />
		<xsd:attribute name="value"
			type="xsd:string"
			use="required" />
	</xsd:complexType>
</xsd:element>
```

Beispiel für das Modell von Stammdaten: Stammdaten *Vitalstatus*

```
<masterdata>
	<table id="MasterData1077066632703"
```

```
                label="Stammdaten Vitalstatus"
                name="SD_Vitalstatus"
                description="Stammdaten für Merkmalsart Vitalstatus">
                        <row id="1" value="als verstorben bekannt; mit Datum" />
                        <row id="2" value="als verstorben bekannt; ohne Datum" />
                        <row id="3" value="lost to follow up (ltf)" />
                        <row id="4" value="ltf: zurück ins Heimatland" />
                        <row id="5" value="ltf: Arztwechsel" />
                        <row id="6" value="als lebend bekannt" />
                        <row id="7" value="nicht als verstorben bekannt (unbekannt)" />
        </table>
        …
</masterdata>
```

3.2.6 Integritätsbedingungen

Die Integrität der im medizinischen Dokumentationssystem verwalteten Daten, d. h. deren Vollständigkeit und Korrektheit, lässt sich durch den Einsatz von *Integritätsbedingungen* sicherstellen.

Definition 32: Eine **Integritätsbedingung** ist eine einer Entität zugeordnete logische Formel zur Sicherstellung der Integrität der im medizinischen Dokumentationssystem erfassten Daten. Sie gibt an, welche Forderungen von den beteiligten Elementen erfüllt werden müssen und welche Folge gegebenenfalls ein Nichterfüllen dieser Forderungen hat.

Wie bereits die Entität, so ist auch die Integritätsbedingung ein abstraktes Konstrukt, das erst durch die möglichen verschiedenen Ausprägungen konkret instanziiert wird. Hier zunächst eine Liste der Attribute, die für alle Integritätsbedingungen relevant sind:

Allgemeine Attribute:

- `parent`: zugeordnete Entität, auf die sich die Integritätsbedingung bezieht (siehe auch Abschnitt 3.2.2)
- `type`: Typ der Integritätsbedingung (Fehler oder Warnung)
- `active`: Schalter, der anzeigt, ob die Integritätsbedingung aktiv ist, d. h. zur Anwendung kommt
- `obligatory`: Schalter, der anzeigt, ob die Anwendung der Integritätsbedingung verpflichtend ist, d. h. diese darf nicht inaktiv sein.
- `message`: Meldungstext, der bei Verletzung der Integritätsbedingung angezeigt werden soll

Die im Rahmen dieser Arbeit modellierten Ausprägungen von Integritätsbedingungen finden sich in Abschnitt 3.3.

3.2.7 Applikation

Auf der obersten Ebene der Elementenhierarchie (siehe Abbildung 4: Dokumentationsmodell) steht das Pendant des zu dokumentierenden Realitätsausschnittes, das als *Applikation* bezeichnet wird.

Definition 33: Eine **Applikation** ist die Menge aller zu einem medizinischen Dokumentationssystem gehörenden Elemente (Datenbank, freie und gebundene Merkmalsarten, Kontexte, Stammdaten sowie Integritätsbedingungen).

Definition 34: Modellgrammatik Applikation

```
<xsd:element name="application"
	type="applicationType" />

<xsd:complexType name="applicationType">
	<xsd:sequence>
		<xsd:element ref="masterdata"
			minOccurs="1"
			maxOccurs="1"/>
		<xsd:element ref="attributes"
			minOccurs="1"
			maxOccurs="1"/>
		<xsd:element ref="database"
			minOccurs="1"
			maxOccurs="1"/>
	</xsd:sequence>
</xsd:complexType>
```

Beispiel für das Modell einer Applikation:

```
<application>
	<masterdata>
		<table id="MasterData1077066347703"
			label="Stammdaten diagnostische Verfahren"
			description=""
			name="SD_diag_Verf">
		</table>
		…
	</masterdata>
	<attributes>
		<attribute	id="Attribute1077065262656"
				label="ID"
				description="Interne ID für Kohorte">
		</attribute>
		…
```

```
        </attributes>
        <database id="Database1077067859515"
                label="hivnet"
                description="Dokumentationssystem auf Grundlage des Merkmalskatalogs HIVNET"
                name="hivnet">
                <context id="Context1077067859500"
                        label="Patient"
                        description="Patientendaten"
                        name="Patient">
                        …
                </context>
                …
        </database>
</application>
```

3.3 Modellierte Integritätsbedingungen

Mludek unterscheidet in seiner Arbeit (Mludek 2001) die folgenden Kategorien von Integritätsbedingungen:

- Datentyp/elementare Wertemenge
- Länge
- Semantische Integritätsbedingung
    - Semantische Integritätsbedingung Nullter Ordnung
    - Semantische Integritätsbedingung Höherer Ordnung
        - Semantische Integritätsbedingung Höherer Ordnung Typ A
        - Semantische Integritätsbedingung Höherer Ordnung Typ B
- Vollständigkeitsbedingung
    - Vollständigkeitsbedingung Nullter Ordnung
        - Wertigkeit
        - Wertemenge
    - Vollständigkeitsbedingung Höherer Ordnung
        - Vollständigkeitsbedingung Höherer Ordnung Typ A
        - Vollständigkeitsbedingung Höherer Ordnung Typ B

Auf Grundlage der formalen Spezifikation dieser Bedingungen, ihrer jeweiligen Relevanz für die medizinische Dokumentation sowie design- bzw. implementierungstechnischer Rahmenbedingungen (z. B. Normalisierungsprinzip im relationalen Datenmodell) entstand eine eigene Kategorisierung. Für die Verwendung im Modell wurden folgende konkrete Ausprägungen von Integritätsbedingungen identifiziert und definiert: *Datentyp*, *Datenlänge*, *Wertebereich*, *Schlüssel*, *Anzahl der Merkmalsausprägungen*, *Referenz*, *Suche*, *Ergebnis*, *Hierarchie* und *Semantische Integritätsbedingung*. Diese werden in den

folgenden Abschnitten erläutert. Hierbei findet das in Tabelle 1 dargestellte Schema Anwendung, das die Eigenschaften der jeweiligen Integritätsbedingung verdeutlicht.

Entität	{ Datenbank | Kontext | gebundene MA | freie MA }
Typ	{ Fehler | Warnung }
obligatorisch	{ ja | nein }
aktiv	{ ja | nein }
Ebene	{ Model | View | Controller }

Tabelle 1: Schema der Eigenschaften von Integritätsbedingungen

Zusätzliches Unterscheidungsmerkmal in der Tabelle ist die Architekturebene (Goertzen, Stausberg 2004). Bedingt durch die Anforderungen einer strukturierten automatisierten Generierung folgen die Kategorien dem Model-View-Controller Entwurfsmuster. Die Verwendung dieses Entwurfsmusters führt zu einer direkten Relation der relevanten Integritätsbedingungen und den jeweils betroffenen Architekturebenen eines modellierten Systems:

- **Model**: Die Integritätsbedingung betrifft die Datenbasis des medizinischen Dokumentationssystems. Sie hat wesentlichen Einfluss auf Art und Inhalt der dokumentierten Daten, indem sie zumeist statische, syntaktische Regeln formuliert, die sich auf der Datenhaltungsschicht des Dokumentationssystems abbilden lassen.
- **View**: Die Integritätsbedingung betrifft ausschließlich die Darstellung bzw. Aufbereitung. Sie hat keine inhaltlichen oder semantischen Auswirkungen auf die dokumentierten Daten.
- **Controller**: Die Integritätsbedingung betrifft semantische Eigenschaften der dokumentierten Daten. Die formulierten Regeln können statischer oder dynamischer Natur sein und lassen sich im Allgemeinen nicht in der Datenhaltungsschicht abbilden, sondern benötigen zusätzliche Funktionalitäten.

Auch für die Deklaration von Integritätsbedingungen ist eine Grammatik zu definieren. Das folgende XML-Schema zeigt den Teil der Grammatik, der allen Integritätsbedingungen gemein ist. Individuelle Grammatikbestandteile werden in den folgenden Abschnitten ergänzt.

Definition 35: Modellgrammatik Integritätsbedingungen

```
<xsd:element name="constraint" type="constraintType" />

<xsd:complexType name="constraintType">
       <xsd:sequence>
              <!—- für Hierarchie-Integritätsbedingung -->
```

```
                <xsd:element ref="child"
                        minOccurs="0"
                        maxOccurs="unbounded" />
                <!—- für Wertebereich- und Referenz-Integritätsbedingung -->
                <xsd:element ref="foreign-key"
                        minOccurs="0"
                        maxOccurs="1" />
                <!—- für Semantische Integritätsbedingung -->
                <xsd:element ref="and"
                        minOccurs="0"
                        maxOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="type"
                type="constraintTypeType"
                use="required"/>
        <xsd:attribute name="handle"
                type="constraintHandleType"
                default="error" />
        <xsd:attribute name="obligatory"
                type="constraintBoolType"
                use="optional" />
        <xsd:attribute name="active"
                type="constraintBoolType"
                default="true"/>
        <xsd:attribute name="message"
                type="xsd:string"
                use="optional"/>
        …
</xsd:complexType>

<xsd:simpleType name="constraintTypeType">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="type"/>
                <xsd:enumeration value="length"/>
                <xsd:enumeration value="domain"/>
                <xsd:enumeration value="key"/>
                <xsd:enumeration value="quantity"/>
                <xsd:enumeration value="reference"/>
                <xsd:enumeration value="search"/>
                <xsd:enumeration value="result"/>
                <xsd:enumeration value="hierarchy"/>
                <xsd:enumeration value="semantic"/>
                <!-- … Erweiterung hier möglich -->
        </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="constraintHandleType">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="error"/>
                <xsd:enumeration value="warning"/>
        </xsd:restriction>
```

```
</xsd:simpleType>

<xsd:simpleType name="constraintBoolType">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="true"/>
		<xsd:enumeration value="false"/>
	</xsd:restriction>
</xsd:simpleType>
```

3.3.1 Datentyp-Integritätsbedingung

Durch eine Datentyp-Integritätsbedingung wird festgelegt, welchem Datentyp jede Ausprägung einer Merkmalsart angehören muss.

Entität	freie Merkmalsart
Typ	Fehler
obligatorisch	ja
aktiv	ja
Ebene	Model

Tabelle 2: Eigenschaften Datentyp-Integritätsbedingung

Attribute:

- `type`: Datentyp, dem jede Ausprägung der Merkmalsart angehören muss. Derzeit mögliche abstrakte Datentypen sind Varchar, Numeric, Integer, Real, Date, Time, Timestamp und Ref.

Beispiel:

Das Geburtsdatum eines Patienten muss vom Datentyp ‚Datum' sein.

Definition 36: Modellgrammatik Datentyp-Integritätsbedingung

```
<xsd:attribute name="datatype"
	type="datatypeType"
	use="optional"/>

<xsd:simpleType name="datatypeType">
	<xsd:restriction base="xsd:string">
		<xsd:enumeration value="character"/>
		<xsd:enumeration value="number"/>
		<xsd:enumeration value="integer"/>
		<xsd:enumeration value="real"/>
		<xsd:enumeration value="date"/>
		<xsd:enumeration value="time"/>
		<xsd:enumeration value="timestamp"/>
		<xsd:enumeration value="reference"/>
	</xsd:restriction>
</xsd:simpleType>
```

Beispiel für das Modell einer Datentyp-Integritätsbedingung:

```
<attribute id="Attribute1077065566765"
        label="Geburtsdatum"
        description="Geburtsdatum einer Person">
        <constraint type="type"
                handle=“error“
                active=“yes“
                datatype="date"
                message="Fehlerhafter Datentyp" />
        …
</attribute>
```

Beim vorangegangen Beispiel sind die Angaben zu *handle* und *active* fakultativ, da sie implizit durch die Angabe des Typs vorgegeben werden und somit nicht explizit deklariert werden müssen.

3.3.2 Datenlänge-Integritätsbedingung

Durch eine Datenlänge-Integritätsbedingung wird festgelegt, welche Länge die Merkmalsausprägungen haben dürfen.

Entität	freie Merkmalsart
Typ	Fehler
obligatorisch	ja
aktiv	ja
Ebene	Model

Tabelle 3: Eigenschaften Datenlänge-Integritätsbedingung

Attribute:

- `minLength`: minimale Länge der Daten
- `maxLength`: maximale Länge der Daten
- `decimals`: Anzahl der Nachkommastellen bei Zahlen

Beispiel:

Die Postleitzahl eines (deutschen) Ortes muss die Datenlänge ‚5' haben. Damit sind die Attribute `minLength` und `maxLength` auf jeweils ‚5' zu setzen.

Definition 37: Modellgrammatik Datenlänge-Integritätsbedingung

```
<xsd:attribute name="min"
        type="xsd:nonNegativeInteger"
        use="optional"/>
<xsd:attribute name="max"
```

```
        type="xsd:nonNegativeInteger"
        use="optional"/>
<xsd:attribute name="dec"
        type="xsd:nonNegativeInteger"
        use="optional"/>
```

Beispiel für das Modell einer Datenlänge-Integritätsbedingung:

```
<attribute id="Attribute1077065566732"
        label="PLZ"
        description="Postleitzahl eines Ortes">
        <constraint type="length"
                min="5"
                max=“5“
                dec=“0“
                message="Fehlerhafte Feldlänge" />
        …
</attribute>
```

3.3.3 Wertebereich-Integritätsbedingung

Eine Wertebereich-Integritätsbedingung beschränkt die durch den Datentyp aufgespannte Menge möglicher Merkmalsausprägungen durch die Angaben von Ober- und/oder Untergrenze bzw. durch einen Verweis auf zugehörige Stammdaten.

Entität	freie Merkmalsart
Typ	Fehler
obligatorisch	ja
aktiv	ja
Ebene	(Model)/Controller

Tabelle 4: Eigenschaften Wertebereich-Integritätsbedingung

Da nur wenige Datenbanksysteme bisher die zwar standardisierten, aber wenig verbreiteten Möglichkeiten semantischer Integritätsprüfung (z. B. *CHECK*, *ASSERTION*) umsetzen, erfolgt hier eine Zuordnung zur Controller-Ebene (siehe hierzu ausführlicher Kapitel 4).

Attribute:

- `reference`: Referenz auf Stammdaten
- `from`: kleinstmögliche Ausprägung aus dem Wertebereich
- `to`: größtmögliche Ausprägung aus dem Wertebereich

Beispiel für Beschränkung durch Ober- und Untergrenze:

Jede dokumentierte Ausprägung zur Merkmalsart *Lebensalter in Jahren* muss zwischen den Werten ‚0' und ‚122' liegen.

Definition 38: Modellgrammatik Wertebereich-Integritätsbedingung

```
<xsd:attribute name="from"
        type="xsd:integer"
        use="optional"/>
<xsd:attribute name="to"
        type="xsd:integer"
        use="optional"/>
```

Beispiel für Stammdaten-Wertebereich:

Der *Vitalstatus* im Kontext *Patient* muss aus der Menge der in den *Stammdaten_Vitalstatus* definierten Merkmalsausprägungen gewählt werden.

Definition 39: Modellgrammatik Stammdaten-Wertebereich-Integritätsbedingung

```
<xsd:element name="foreign-key"
        type="foreignKeyType" />

<xsd:complexType name="foreignKeyType">
        <xsd:sequence>
                <xsd:element name="reference"
                        minOccurs="1"
                        maxOccurs="1">
                        <xsd:complexType>
                                <xsd:attribute name="local"
                                        type="xsd:IDREF"
                                        use="required" />
                                <xsd:attribute name="foreign"
                                        type="xsd:string"
                                        use="required" />
                        </xsd:complexType>
                </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="foreignTable"
                type="xsd:IDREF"
                use="required"/>
</xsd:complexType>
```

Beispiel für das Modell einer Wertebereich-Integritätsbedingung:

```
<contextattribute id="ContextAttribute1077068082972"
        label="Vitalstatus"
        description="Vitalstatus eines Patienten"
        name="Vitalstatus"
        reference="Attribute1077066036953">
```

```
        <constraint type="domain"
                    message="Wert außerhalb des Wertebereichs">
                <foreign-key foreignTable="MasterData1077066347703">
                        <reference local="Attribute1077066178656" foreign="id" />
                </foreign-key>
        </constraint>
        …
</contextattribute>
```

3.3.4 Schlüssel-Integritätsbedingung

Die Schlüssel-Integritätsbedingung gibt an, ob die (gebundene) Merkmalsart den zugehörigen Kontext eindeutig identifiziert.

Entität	gebundene Merkmalsart
Typ	Fehler
obligatorisch	ja
aktiv	ja
Ebene	Model

Tabelle 5: Eigenschaften Schlüssel-Integritätsbedingung

Attribute:

- `key`: Schalter, der aussagt, ob die Merkmalsart Schlüsselattribut ist oder nicht

Beispiel:

Ein Patient wird durch die gebundene Merkmalsart *ID* eindeutig referenziert.

Definition 40: Modellgrammatik Schlüssel-Integritätsbedingung

```
<xsd:attribute name="primaryKey"
        type="constraintBoolType"
        default="false"/>
```

Beispiel für das Modell einer Schlüssel-Integritätsbedingung:

```
<contextattribute id="ContextAttribute1077067928375"
        label="ID"
        description="Interne ID für Kohorte"
        name="ID"
        reference="Attribute1077065262656">
        <constraint type="key"
                primaryKey="true" />
        …
</contextattribute>
```

3.3.5 Anzahl-Integritätsbedingung

Durch die Anzahl-Integritätsbedingung wird festgelegt, wie viele Merkmalsausprägungen zu einer Merkmalsart erfasst werden können bzw. müssen.

Entität	gebundene Merkmalsart
Typ	Fehler
obligatorisch	ja
aktiv	ja
Ebene	Model

Tabelle 6: Eigenschaften Anzahl-Integritätsbedingung

Attribute:

- `minQuantity`: minimale Anzahl der Merkmalsausprägungen
- `maxQuantity`: maximale Anzahl der Merkmalsausprägungen

Beispiel:

Für jeden Patienten ist genau ein Geburtsdatum zu erfassen (Pflichtfeld). Die Werte der Attribute `minQuantity` und `maxQuantity` sind jeweils ‚1'.

Definition 41: Modellgrammatik Anzahl-Integritätsbedingung

```
<xsd:attribute name="min"
       type="xsd:nonNegativeInteger"
       use="optional"/>
<xsd:attribute name="max"
       type="xsd:nonNegativeInteger"
       use="optional"/>
```

Beispiel für das Modell einer Anzahl-Integritätsbedingung:

```
<contextattribute id="ContextAttribute1077067928375"
       label="ID"
       description="Interne ID für Kohorte"
       name="ID"
       reference="Attribute1077065262656">
       <constraint type="quantity"
              min="1"
              max="1"
              message="Falsche Anzahl von Merkmalsausprägungen" />
       …
</contextattribute>
```

3.3.6 Referenz-Integritätsbedingung

Eine Referenz-Integritätsbedingung legt fest, dass es sich bei der Merkmalsausprägung um eine Referenz auf eine andere gebundene Merkmalsart, also um eine Fremdschlüsselbeziehung handelt.

Entität	gebundene Merkmalsart
Typ	Fehler
obligatorisch	ja
aktiv	ja
Ebene	Model

Tabelle 7: Eigenschaften Referenz-Integritätsbedingung

Attribute:

- `reference`: eine referenzierte gebundene Merkmalsart

Beispiel:

Die gebundene Merkmalsart *Patient* im Kontext *Meldung* ist eine Referenz auf die gebundene Merkmalsart *ID* im Kontext *Patient*. Dadurch kann die Merkmalsart *Patient* nur Ausprägungen aufweisen, die in *ID* vorhanden sind.

Definition 42: Modellgrammatik Referenz-Integritätsbedingung

→ siehe Wertebereich-Integritätsbedingung

Beispiel für das Modell einer Referenz-Integritätsbedingung:

```
<contextattribute id="ContextAttribute1077068082970"
        label="Patient"
        description="ID des Patienten"
        name="Patient"
        reference="Attribute1077065982312">
        <constraint type="reference">
                <foreign-key foreignTable="Context1077067859500">
                        <reference local="ContextAttribute1077067993272"
                                foreign="ContextAttribute1077067928375" />
                </foreign-key>
        </constraint>
        …
</contextattribute>
```

3.3.7 Suche-Integritätsbedingung

Mittels der Suche-Integritätsbedingung wird definiert, dass es sich bei der Merkmalsart um ein Suchfeld handeln soll. Durch Suche-Integritätsbedingungen wird eine Teilmenge der Merkmalsarten eines Kontextes als Suchfelder definiert. Schlüsselfelder des Kontextes sind automatisch Bestandteil dieser Teilmenge.

Entität	gebundene Merkmalsart
Typ	Fehler
obligatorisch	ja
aktiv	ja
Ebene	View

Tabelle 8: Eigenschaften Suche-Integritätsbedingung

Attribute:

- `searchColumn`: Schalter, der anzeigt, ob die Merkmalsart Suchattribut ist

Beispiel:

Das Pseudonym des Patienten (Merkmalsart *Pseudonym* im Kontext *Patient*) soll in der Anwendung als Suchfeld bereitstehen.

Definition 43: Modellgrammatik Suche-Integritätsbedingung

```
<xsd:attribute name="searchColumn"
        type="constraintBoolType"
        default="false"/>
```

Beispiel für das Modell einer Suche-Integritätsbedingung:

```
<contextattribute id="ContextAttribute1077067928377"
        label="Pseudonym"
        description="Pseudonym eines Patienten"
        name="Pseudonym"
        reference="Attribute1077065370015">
        <constraint type="search"
                searchColumn="true" />
        …
</contextattribute>
```

3.3.8 Ergebnis-Integritätsbedingung

Eine Ergebnis-Integritätsbedingung zeigt an, ob eine Merkmalsart nach einer Suchoperation zu der angezeigten Teilmenge von Merkmalsarten gehören soll, welche die Suchergebnisse in Listenform darstellen. Die Informationen in der Teilmenge der Ergebnis-Merkmalsarten eines Kontextes sollen es dem Anwender ermöglichen, anhand der darge-

stellten Ergebnisse eine Entscheidung darüber zu treffen, welchen Datensatz er zur weiteren Bearbeitung auswählt.

Entität	gebundene Merkmalsart
Typ	Fehler
obligatorisch	ja
aktiv	ja
Ebene	View

Tabelle 9: Eigenschaften Ergebnis-Integritätsbedingung

Attribute:

- `resultColumn`: Schalter, der anzeigt, ob die Merkmalsart Ergebnisattribut ist

Beispiel:

Die Merkmalsart *Geburtsdatum* im Kontext *Patient* soll nach einer Suchaktion nach dem Pseudonym (siehe Abschnitt 3.3.7) als Bestandteil der Ergebnismenge angezeigt werden.

Definition 44: Modellgrammatik Ergebnis-Integritätsbedingung

```
<xsd:attribute name="resultColumn"
        type="constraintBoolType"
        default="false"/>
```

Beispiel für das Modell einer Ergebnis-Integritätsbedingung:

```
<contextattribute id="ContextAttribute1077067928384"
        label="Geburtsdatum"
        description="Geburtsdatum einer Person"
        name="Geburtsdatum"
        reference="Attribute1077065566765">
        <constraint type="result"
                resultColumn="true" />
        …
</contextattribute>
```

3.3.9 Hierarchie-Integritätsbedingung

Durch eine Hierarchie-Integritätsbedingung wird der Aufbau eines medizinischen Dokumentationssystems definiert. Insbesondere wird festgelegt, wie zwischen den modellierten Kontexten navigiert werden kann. Obwohl durch jede Hierarchie-Integritätsbedingung eine hierarchische Struktur modelliert wird, kann auf Grund der Tatsache, dass eine Entität in mehreren solcher Bedingungen referenziert sein kann, in der Gesamtsicht eine Netzwerkstruktur entstehen. Auf diese Weise lässt sich zum Beispiel modellieren, dass auf die Daten des Kontextes Meldung sowohl direkt (durch Suchen, siehe Abschnitt 3.3.7) als auch aus der Detailansicht eines bestimmten Patienten zugegriffen werden kann.

Damit eine Hierarchie-Integritätsbedingung überhaupt sinnvoll eingesetzt werden kann, müssen zwischen den beteiligten Kontexten entsprechende Schlüssel-/Fremdschlüsselbeziehungen (respektive Schlüssel-/Referenz-Integritätsbedingungen) bestehen.

Entität	{ Datenbank | Kontext }
Typ	Fehler
obligatorisch	ja
aktiv	ja
Ebene	View

Tabelle 10: Eigenschaften Hierarchie-Integritätsbedingung

Attribute:

- `children`: Menge der Entitäten, die dieser Entität untergeordnet sind

Beispiel:

Auf der obersten Hierarchie-Ebene der Anwendung hat der Benutzer Zugriff auf die Kontexte *Patient* und *Meldung*. Vom Kontext *Patient* kann er in die Kontexte *Meldung, …* sowie *Befund* navigieren (usw.)

Definition 45: Modellgrammatik Hierarchie-Integritätsbedingung

```
<xsd:element name="child"
	type="childType" minOccurs="1"
	maxOccurs="unbounded" />

<xsd:complexType name="childType">
	<xsd:attribute name="context"
		type="xsd:IDREF"
		use="required" />
</xsd:complexType>
```

Beispiel für das Modell einer Hierarchie-Integritätsbedingung:

```
<database id="Database1077067859515"
	label="hivnet"
	description="Dokumentationssystem auf Grundlage des Merkmalskatalogs HIVNET"
	name="hivnet">
	<constraint type="hierarchy">
		<child context="Context1077067859500" />
		<child context="Context1077068009640" />
	</constraint>
	…
</database>
```

3.3.10 Semantische Integritätsbedingung

Semantische Integritätsbedingungen definieren Regeln über semantische Eigenschaften der dokumentierten Daten, die in logischen Formeln Ausdruck finden.

Entität	gebundene Merkmalsart
Typ	{ Fehler | Warnung }
obligatorisch	nein
aktiv	{ ja | nein }
Ebene	(Model)/Controller

Tabelle 11: Eigenschaften Semantische Integritätsbedingung

Auch hier gilt eine Zuordnung zum Bereich Controller, da die entsprechenden Mechanismen durch geläufige Datenbankmanagementsysteme zurzeit noch nicht unterstützt werden.

Attribute:

- `statement`: logische Aussage, die für diese Merkmalsart erfüllt sein muss
- `condition`: Bedingung, unter der die logische Aussage (`statement`) gilt

Beispiel:

Wenn die Merkmalsart *Schwangerschaft* die Ausprägung ‚ja' hat (`condition`), dann muss die Merkmalsart *Geschlecht* im Kontext *Patient* die Ausprägung ‚weiblich' haben (`statement`).

Definition 46: Modellgrammatik Semantische Integritätsbedingung

```
<xsd:attribute name="operator"
       type="constraintOperatorType" />
<xsd:attribute name="argumenttype"
       type="constraintArgType" />
<xsd:attribute name="argument"
       type="xsd:string" />

<xsd:element name="and"
       type="andType" />
<xsd:complexType name="andType">
       <xsd:sequence>
              <xsd:element ref="statement"
                     minOccurs="0"
                     maxOccurs="unbounded" />
              <xsd:element ref="and"
                     minOccurs="0"
                     maxOccurs="unbounded" />
              <xsd:element ref="or"
```

```
                    minOccurs="0"
                    maxOccurs="unbounded" />
        </xsd:sequence>
</xsd:complexType>

<xsd:element name="or"
        type="orType" />
<xsd:complexType name="orType">
        <xsd:sequence>
                <xsd:element ref="statement"
                        minOccurs="0"
                        maxOccurs="unbounded" />
                <xsd:element ref="and"
                        minOccurs="0"
                        maxOccurs="unbounded" />
                <xsd:element ref="or"
                        minOccurs="0"
                        maxOccurs="unbounded" />
        </xsd:sequence>
</xsd:complexType>

<xsd:element name="statement"
        type="statementType" />
<xsd:complexType name="statementType">
        <xsd:attribute name="argument1"
                type="xsd:string"
                use="required" />
        <xsd:attribute name="operator"
                type="constraintOperatorType"
                use="required" />
        <xsd:attribute name="argumenttype"
                type="constraintArgType"
                use="required" />
        <xsd:attribute name="argument2"
                type="xsd:string"
                use="required" />
</xsd:complexType>

<xsd:simpleType name="constraintOperatorType">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="EQ"/>
                <xsd:enumeration value="NE"/>
                <xsd:enumeration value="GT"/>
                <xsd:enumeration value="LT"/>
                <xsd:enumeration value="GE"/>
                <xsd:enumeration value="LE"/>
        </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="constraintArgType">
        <xsd:restriction base="xsd:string">
```

```
                <xsd:enumeration value="value"/>
                <xsd:enumeration value="null"/>
                <xsd:enumeration value="reference"/>
        </xsd:restriction>
</xsd:simpleType>
```

Beispiel für das Modell einer Semantischen Integritätsbedingung:

```
<contextattribute id="ContextAttribute1077067928393"
        label="Geschlecht"
        description="Geschlecht einer Person"
        name="Geschlecht"
        reference="Attribute1077065844500">
        <constraint type="semantic"
                handle="error"
                active="true"
                message="Geschlecht muss bei Schwangerschaft weiblich sein."
                operator="EQ"
                argumenttype="value"
                argument="2"> <!-- Repräsentant für ‘weiblich’ -->
                <and>
                        <statement argument1="Patient.Schwangerschaft"
                                operator="EQ"
                                argumenttype="value"
                                argument2="2" /> <!-- Repräsentant für ‘ja’ -->
                </and>
        </constraint>
        …
</contextattribute>
```

3.3.11 Weitere Integritätsbedingungen

Das Dokumentationsmodell wurde so entwickelt und in das integrierte System *InCoMe* eingebettet, dass eine Erweiterung der Menge der Integritätsbedingungen - beispielsweise um zusätzliche View-Integritätsbedingungen zur Präsentationssteuerung - leicht möglich ist.

3.4 Zusammenfassung

Es wurden die Modellelemente vorgestellt, aus denen sich ein medizinisches Dokumentationssystem modellieren lässt. In der Gesamtsicht stellt sich das Dokumentationsmodell hierarchisch dar. Die Angaben der jeweiligen Kardinalitäten folgen der DTD-Notation (Harold, Means 2001):

*	beliebig viele Elemente, auch gar keines
+	mehrere Elemente, aber wenigstens eins
sonst:	genau ein Element

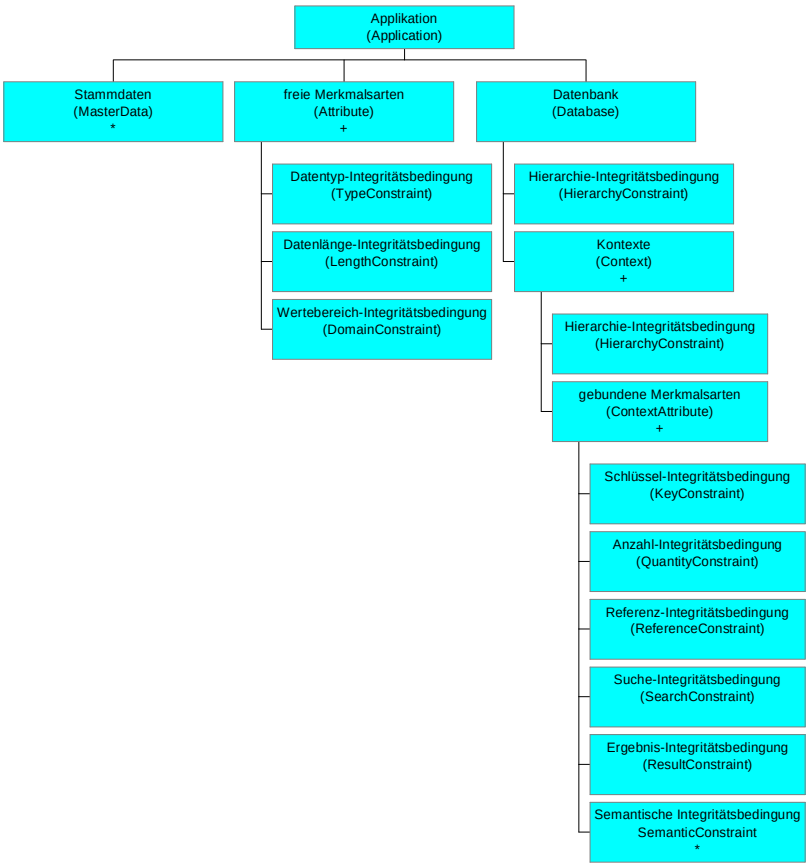


Abbildung 4: Dokumentationsmodell

4 Ergebnisse Teil B: *InCoMe* - Integriertes System

4.1 Einleitung

Das Hauptziel dieser Arbeit ist die Entwicklung eines integrierten Systems zur computerunterstützten Generierung medizinischer Dokumentationssysteme (*InCoMe*). Im vorangegangenen Kapitel wurde ein Dokumentationsmodell vorgestellt, mit dem sich die für die Generierung relevanten Informationen erfassen lassen.

In diesem Kapitel wird gezeigt, welche Systemkomponenten benötigt werden, um einen effektiven, integrierten und nicht zuletzt benutzerfreundlichen Generierungsprozess zu ermöglichen. Der letzte Abschnitt stellt beispielhaft eine mit diesem integrierten System entwickelte Applikation auf Grundlage des Merkmalskatalogs des Kompetenznetzes HIV/AIDS vor.

In Abgrenzung zu dem als *InCoMe* bezeichneten, integrierten Entwicklungssystem soll im Folgenden das zu generierende medizinische Dokumentationssystem als Clinical Data Management Systems (*CDMS*) bezeichnet werden.

4.2 Architektur

Zentraler Bestandteil von *InCoMe* ist das Dokumentationsmodell. Von diesem ausgehend wird zunächst eine Komponente als Entwurfsmodul (CDMS-Designer) benötigt, die eine Benutzeroberfläche zur komfortablen Erfassung und Erzeugung eines solchen Dokumentationsmodells bereitstellt.

Die Überführung des Modells in eine Anwendung erfolgt über ein Generierungsmodul (CDMS-Creator), welches über die Einbindung individueller Datenbank- und Spracherweiterungen einen flexiblen Transformationsprozess ermöglicht.

Die so erzeugten Anwendungskomponenten werden über die Einbindung in ein Applikationsframework (CDMS-Framework) zu einer Gesamtanwendung, welche dem Benutzer die modellierte Funktionalität zur Verfügung stellt. Das Framework hat dabei die Funktion, wiederkehrende Anforderungen übergreifender Art zu kapseln und damit den Generierungsaufwand zu minimieren. Abbildung 5 zeigt die Architektur des entwickelten integrierten Systems.

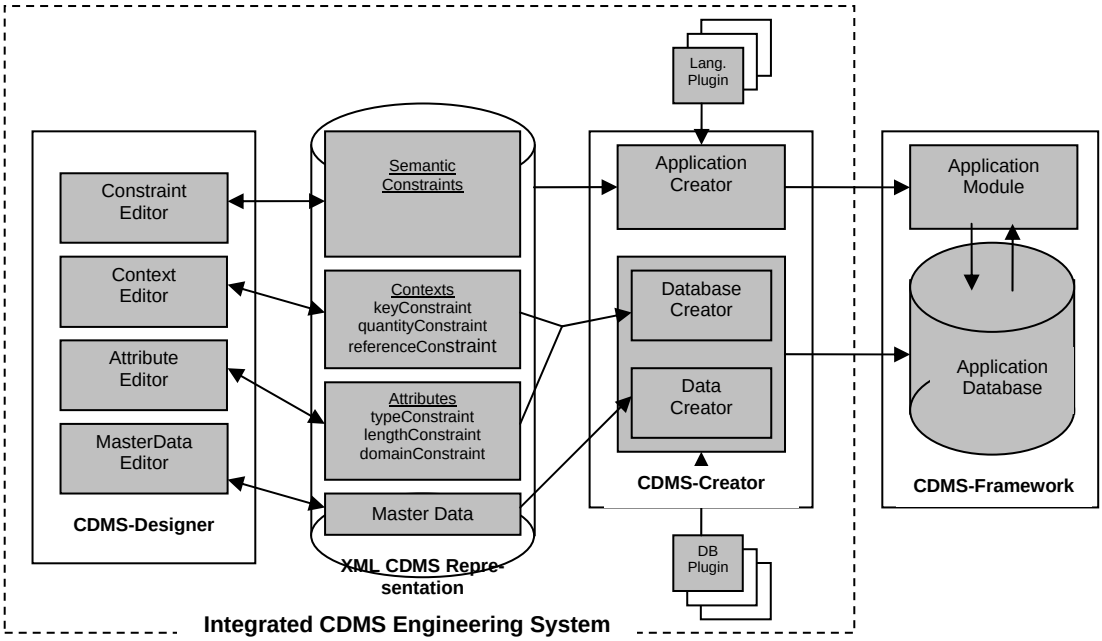


Abbildung 5: Architektur InCoMe

In den folgenden Abschnitten werden die einzelnen Systemkomponenten detailliert dargestellt.

4.3 Entwurfsmodul (CDMS-Designer)

Mit Hilfe des Entwurfsmoduls lässt sich das Dokumentationsmodell über eine grafische Benutzeroberfläche erstellen. Es bietet alle Funktionalitäten, die zur Erfassung, Bearbeitung und Zusammenstellung der beteiligten Modellelemente benötigt werden, in Form von Editoren an. Diese Editoren sollen nicht nur die Erfassung eines Modells erleichtern, sondern auch mögliche Fehlerquellen bei der Modellierung ausschalten, indem z. B. bereits während der Erfassung Plausibilitätsprüfungen vorgenommen werden und so die Integrität des Modells selbst unterstützt wird.

Das Dokumentationsmodell besteht aus den Elementen Merkmalsarten, Kontexten, Datenbank, Stammdaten und Integritätsbedingungen. Im vorliegenden Abschnitt wird kurz dargestellt, wie sich diese Elemente im Entwurfsmodul zum Modell zusammenstellen lassen.

Jedes Element leitet sich von der Schnittstelle Entity ab und muss damit folgende Methoden implementieren:

- `createNode()`: Erzeugung eines XML-Dokumentenknotens zur Speicherung des Elements im Dokumentationsmodell

- `createNodeDB()`: Erzeugung eines XML-Dokumentenknotens zum Datenbeschreibungs-Export
- `createNodeAppl()`: Erzeugung eines XML-Dokumentenknotens zum Applikations-Export
- `build(Element pElement)`: Einlesen und Transformation eines XML-Knotens des Dokumentationsmodells

Zunächst werden die freien Merkmalsarten eines Dokumentationssystems unter dem Menüeintrag *Ansicht→Merkmalsarten* erfasst (siehe Abbildung 6). An dieser Stelle werden zusätzlich die Integritätsbedingungen *Datentyp*, *Datenlänge* und *Wertebereich* modelliert.

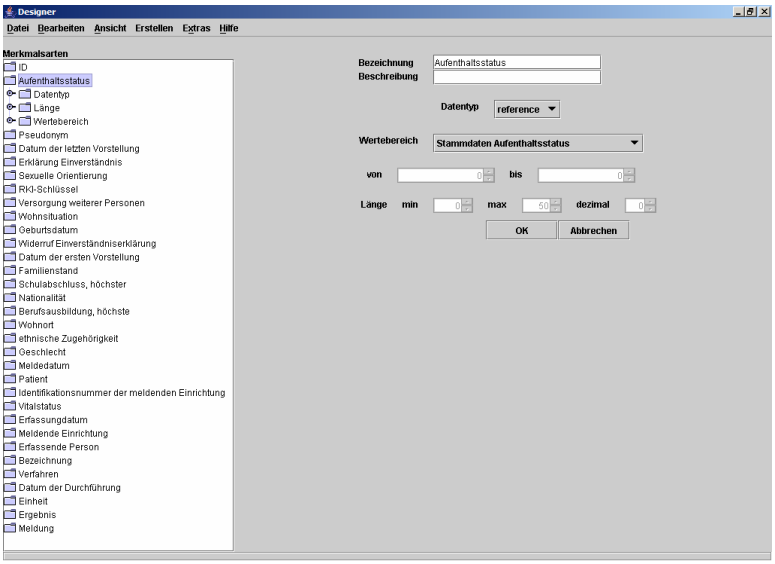


Abbildung 6: GUI Freie Merkmalsarten

Bei Merkmalsarten, deren Wertebereich durch Stammdaten eingeschränkt werden soll, lassen sich die Einträge über *Ansicht→Stammdaten* erfassen (Abbildung 7).

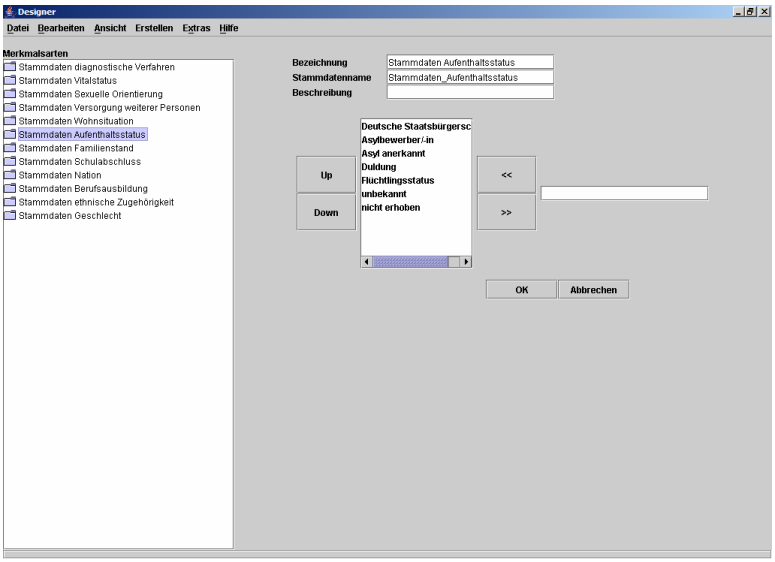


Abbildung 7: GUI Stammdaten

Mit dem Menüeintrag *Ansicht→Kontext* gelangt man in den Bereich der Anwendung, in dem Merkmalsarten zu Kontexten zusammengefasst werden können. Hier ist zunächst die Datenbank zu definieren (siehe Abbildung 8).

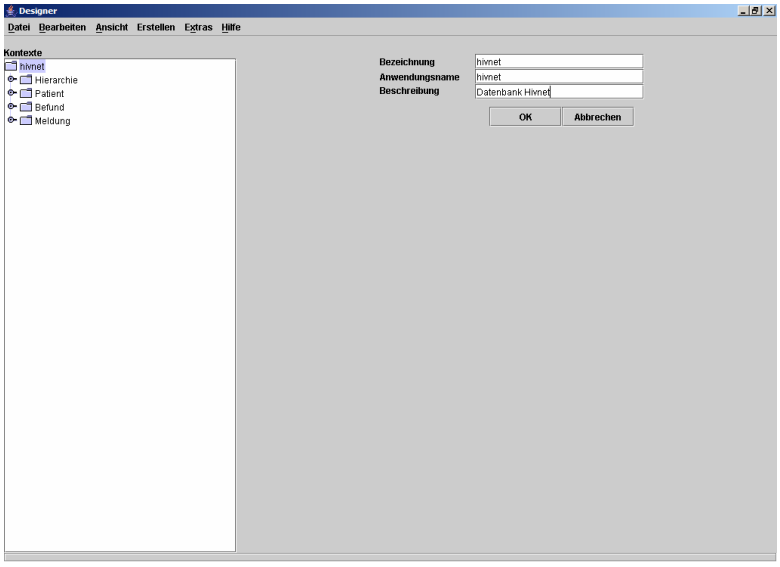


Abbildung 8: GUI Datenbank

Unterhalb der Datenbank werden Kontexte angelegt (Abbildung 9).

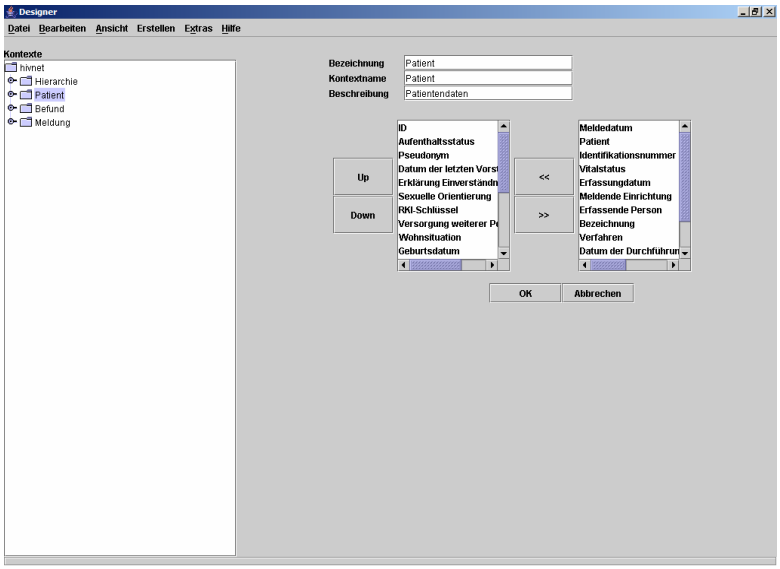


Abbildung 9: GUI Kontext

Den Kontexten schließlich werden Merkmalsarten zugeordnet (Abbildung 10). Hier sind zusätzlich die Integritätsbedingungen *Schlüssel*, *Anzahl*, *Referenz*, *Suche* und *Ergebnis* festzulegen.

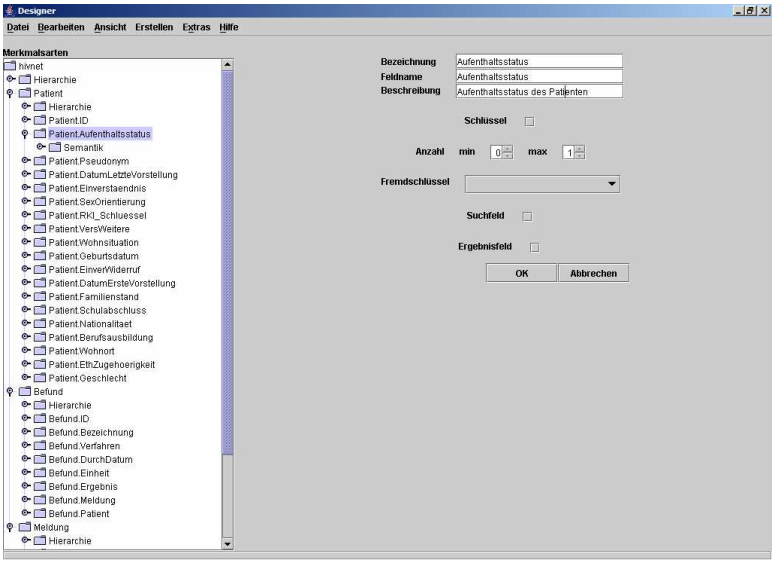


Abbildung 10: GUI Gebundene Merkmalsart

Zu jeder gebundenen Merkmalsart können beliebig viele semantische Integritätsbedingungen modelliert werden. Hierarchie-Integritätsbedingungen werden auf Ebene der Datenbank und der jeweiligen Kontexte angelegt.

Das in Kapitel 3 vorgestellte Dokumentationsmodell lässt sich über einen Menüeintrag exportieren oder direkt über das im nächsten Abschnitt vorgestellte Generierungsmodul verarbeiten.

4.4 Generierungsmodul (CDMS-Creator)

Das Generierungsmodul bietet Schnittstellen zur Transformation des Dokumentationsmodells in eine Datenbeschreibung und eine Anwendungsbeschreibung. Im Rahmen dieser Arbeit wurden verschiedene Transformationsalgorithmen implementiert, die im Folgenden vorgestellt werden. Zur benutzerfreundlichen Handhabung der Generierungsschritte wurde das Generierungsmodul in das Entwurfsmodul integriert. Über eine Konfigurationseinstellung (Menü *Extras→Konfiguration*) lassen sich die erforderlichen Transformationsroutinen für die Datenbasis und die Applikation einbinden und über den Menüpunkt *Erstellen* auf das Dokumentationsmodell anwenden. Durch dieses Vorgehen wurde eine größtmögliche Flexibilität bei der Integration differierender Transformationsalgorithmen (z. B. für andere Zieldatenbanken) sichergestellt. Abbildung 11 zeigt die Konfigurationsoption des Generierungsmoduls.

Abbildung 11: GUI Konfigurationseditor

4.4.1 Datenbeschreibung

Die Erzeugung der Datenbeschreibung aus dem Dokumentationsmodell erfolgt als zweistufiger Prozess. In einem ersten Schritt wird die Menge der für die Datenbasis relevanten Informationen aus dem Modell extrahiert (Datenbankexport). Auf diese Teilmenge lassen sich individuelle, skriptbasierte Transformationsalgorithmen zur Datenbankerzeugung anwenden, welche wie oben beschrieben in das System eingebunden werden können.

Datenbankexport:

Ziel des Datenbankexports ist es, eine vom eingesetzten Datenbank-Managementsystem unabhängige Beschreibung der Datenbasis des medizinischen Dokumentationssystems zu erzeugen, welche zur Weiterverarbeitung extern gespeichert werden kann (über das Menü *Erstellen→Datenbank exportieren*).

Die durch den Datenbankexport erzeugte Datei folgt grundlegend einem Standardformat in XML zur unabhängigen Beschreibung von Datenbank-Ressourcen (Apache 2004b),

das um eine Grammatik für die Beschreibung von Stammdaten erweitert wurde. Damit stehen vielfältige Möglichkeiten offen, die erhobenen Informationen zu nutzen. Der folgende Ausschnitt zeigt beispielhaft die exportierte Struktur des Kontextes *Befund*:

```
<table name="Befund" description="">
<column name="ID" primaryKey="true" required="true" type="NUMERIC" size="10" />
<column name="Bezeichnung" primaryKey="false" type="NUMERIC" size="50" />
<column name="Verfahren" primaryKey="false" type="VARCHAR" size="50" />
<column name="DurchDatum" primaryKey="false" type="DATE" />
<column name="Einheit" primaryKey="false" type="VARCHAR" size="50" />
<column name="Ergebnis" primaryKey="false" type="VARCHAR" size="50" />
<column name="Meldung" primaryKey="false" type="NUMERIC" size="10" />
<column name="Patient" primaryKey="false" type="NUMERIC" size="10" />
<foreign-key foreignTable="SD_diagVerf">
<reference local="Bezeichnung" foreign="id" />
</foreign-key>
</table>
```

Jede Entität implementiert die Methode `createNoteDB()`, die individuell festlegt, welche Informationen für die Datensicht relevant sind und diese in Form eines XML-Elements zurückgibt.

Datenbankerzeugung:

Der zweite Schritt der Generierung der Datenbasis besteht aus einer Transformation der exportierten Informationen (über Menü *Erstellen→Datenbank erzeugen*). Dieses zweistufige Vorgehen erhöht die Flexibilität bei der Integration unterschiedlicher DBM-Systeme. Darüber hinaus ergeben sich trotz Standardisierung der Syntax bereits bei SQL-Datenbanken deutliche Abweichungen, wie folgende Ausschnitte zeigen:

SQL-Statements für MySQL:

```
CREATE TABLE Befund ( ID INT (10) NOT NULL,
                      Bezeichnung INT (50),
                      Verfahren VARCHAR (50),
                      DurchDatum DATE ,
                      Einheit VARCHAR (50),
                      Ergebnis VARCHAR (50),
                      Meldung INT (10),
                      Patient INT (10),
                      PRIMARY KEY (ID));
ALTER TABLE Befund ADD CONSTRAINT Befund_Bezeichnung_FK
       FOREIGN KEY(Bezeichnung) REFERENCES SD_diagVerf(id);
```

SQL-Statements für Oracle:

```
CREATE TABLE Befund ( ID NUMBER (10) NOT NULL,
```

```
                    Bezeichnung NUMBER (50),
                    Verfahren VARCHAR2 (50),
                    DurchDatum DATE ,
                    Einheit VARCHAR2 (50),
                    Ergebnis VARCHAR2 (50),
                    Meldung NUMBER (10),
                    Patient NUMBER (10),
                    CONSTRAINT PK_Befund PRIMARY KEY (ID));
ALTER TABLE Befund ADD CONSTRAINT Befund_Bezeichnung_FK
       FOREIGN KEY(Bezeichnung) REFERENCES SD_diagVerf(id);
```

Durch die verwendete Skripttechnik ist es problemlos möglich, sowohl solche Nuancen abzubilden, als auch komplett differierende, weitere Datenbanksysteme zu integrieren.

4.4.2 Applikation

Über den gleichen, skriptbasierten Mechanismus lässt sich auf Grundlage der modellierten Informationen auch die zugehörige Applikation generieren.
Im Folgenden wird ein Ansatz präsentiert, der mit einem Applikationsframework arbeitet und die modellindividuellen Komponenten in dieses integriert. Dieses Vorgehen hat u. a. folgende Vorteile:

- keine Notwendigkeit, die erstellte Applikation zu kompilieren
- Plattformunabhängigkeit
- sofortige Einsatzfähigkeit der Applikation
- Minimierung des Generierungsaufwandes durch Wiederverwendung
- Erhöhung der Wartbarkeit
- Stabilität des Systems

Dennoch ist es auch möglich, bei Bedarf mittels des integrierten Systems beispielsweise völlig andere Programmiersprachen zu nutzen, indem das zu konfigurierende Generierungsskript entsprechend angepasst wird.

Die Festlegung, welche Informationen einer Entität für die Applikationserstellung relevant sind, erfolgt in der jeweiligen Methode `createNodeAppl()`, die analog zum Datenbankexport ein XML-Element zurückliefert. Auf diese Weise wird eine vollständige XML-Beschreibung der Applikation erzeugt (über Menü *Erstellen→Anwendung exportieren*), welche über das konfigurierbare Applikations-Erstellungsskript transformiert werden kann (Menü *Erstellen→Anwendung erzeugen*).

In Verbindung mit dem im nächsten Abschnitt beschriebenen Applikationsframework wird die vollständige Applikationsbeschreibung (Export) als Eingabe zur Festlegung der Funktionalität der Applikation genutzt. Das eingesetzte Skript erzeugt eine XML-Datei zur Kon-

figuration des Servers (*web.xml*), die bei dessen Start automatisch herangezogen wird und somit die Applikation im Netzwerk verfügbar macht.

4.5 Applikationsframework (CDMS-Framework)

Das Applikationsframework stellt ein Rahmenwerk bereit, welches wiederkehrende Funktionalitätsanforderungen an medizinische Dokumentationssysteme implementiert und dadurch den Aufwand einer jeweiligen Neuimplementierung dieser Komponenten minimiert.
Zu integrieren sind damit nur noch die jeweiligen individuellen Spezifika des modellierten Systems.
Zunächst wird im Folgenden die Einbeziehung der Datenbank in das integrierte System vorgestellt, bevor eine Erläuterung des Anwendungsmoduls erfolgt.

4.5.1 Datenbank

Die Einbindung der im System verwendeten Datenbank erfolgt über Eintragungen in der Server-Konfigurationsdatei *web.xml*. Folgender Ausschnitt zeigt das Beispiel einer Konfiguration für eine Oracle-Datenbank:

```
…
<init-param>
        <param-name>dbDriver</param-name>
        <param-value>oracle.jdbc.driver.OracleDriver</param-value>
</init-param>
<init-param>
        <param-name>dbUrl</param-name>
        <param-value>jdbc:oracle:thin:@localhost:1521:hivnet</param-value>
</init-param>
…
```

Hier wird neben einem gültigen Treiber für die jeweilige Datenbank auch deren Lokation angegeben. Dies ist die URL, unter welcher die durch die Datenbankerzeugung eingerichtete Datenbank im Netzwerk verfügbar ist.

4.5.2 Anwendungsmodul

Das Anwendungsmodul des Applikationsframeworks ist schichtweise aufgebaut (siehe Abbildung 12).

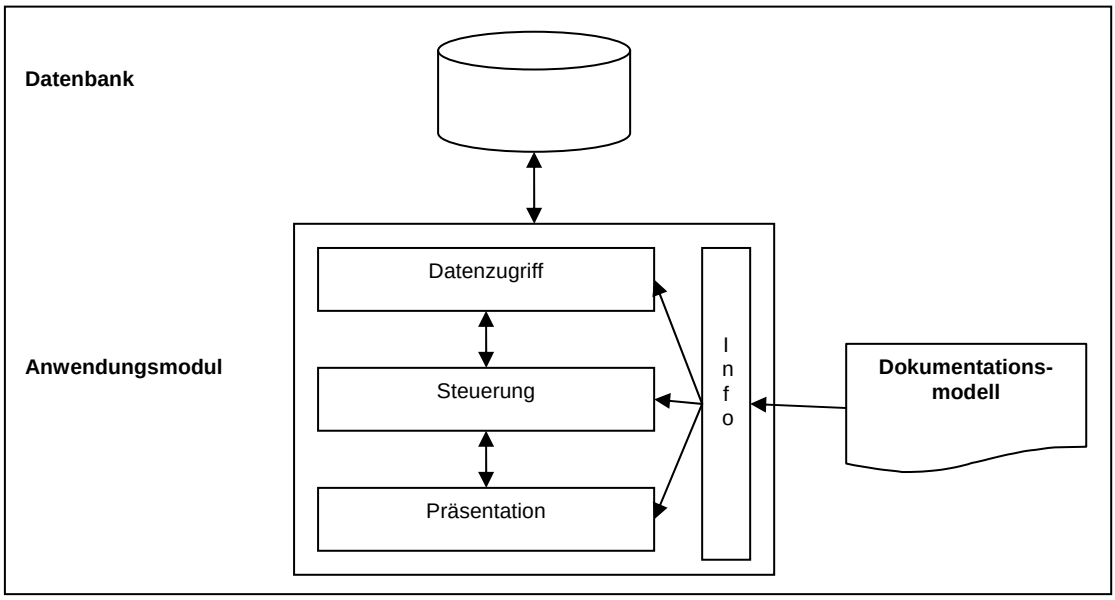


Abbildung 12: Schichtenmodell Applikationsframework

Die Datenzugriffe erfolgen über eine abstrakte Ebene des Frameworks (Paket `model`) unabhängig vom eingesetzten Datenbanksystem.

Die Integritätsprüfung (Pakete `constraints`, `logical`, `types`) ist in diese Schicht integriert. Die Steuerung der Verarbeitung übernehmen auf Servlet-Technologie basierende Klassen (Pakete `servlets`, `login`). Die Präsentationsebene (Paket `view`) ist mit Hilfe von Java Server Pages so ausgestaltet, dass eine Anpassung des Designs unabhängig von sonstigen Komponenten leicht möglich ist. Alle Bildschirmmasken des Systems werden dynamisch anhand der vorliegenden Modellierung erzeugt.

Die Informationen des zu Grunde liegenden Modells werden zentral über Klassen des Pakets `info` verwaltet und der Anwendung bereitgestellt.

4.6 Anwendungsbeispiel: Medizinisches Dokumentationssystem (CDMS-Applikation)

Das in diesem Kapitel vorgestellte System *InCoMe* stellt zusammen mit dem zuvor präsentierten Dokumentationsmodell die Werkzeuge bereit, ein medizinisches Dokumentationssystem in Form einer in einem Browser lauffähigen, integrierten Anwendung zu generieren. Die folgenden Abbildungen zeigen Bildschirmausschnitte eines generierten Systems zur Dokumentation von Patienten im Umfeld des HIV-Kompetenznetzes.

Abbildung 13 zeigt den Anmeldebildschirm des Dokumentationssystems.

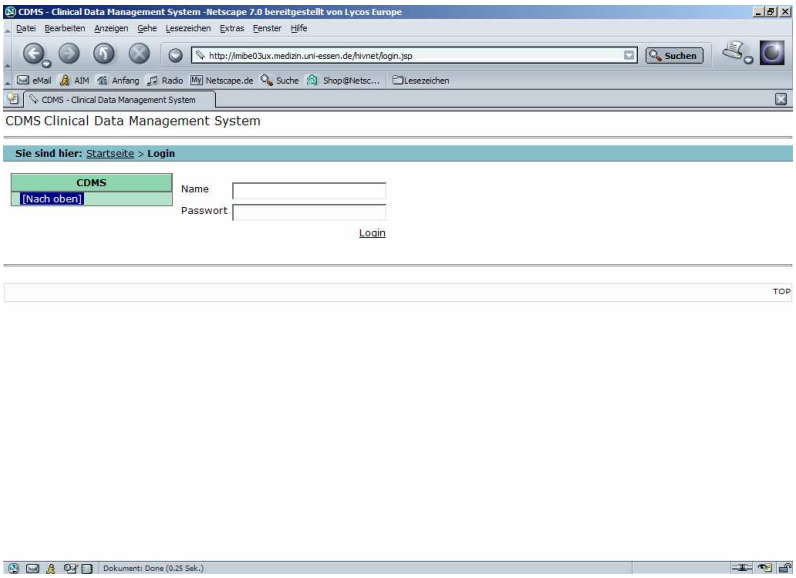


Abbildung 13: CDMS Anmeldung

Nach erfolgreicher Anmeldung hat der Benutzer für jeden Kontext die Möglichkeit, eine Suche nach durch Suche-Integritätsbedingungen definierten Merkmalsarten durchzuführen (Abbildung 14).

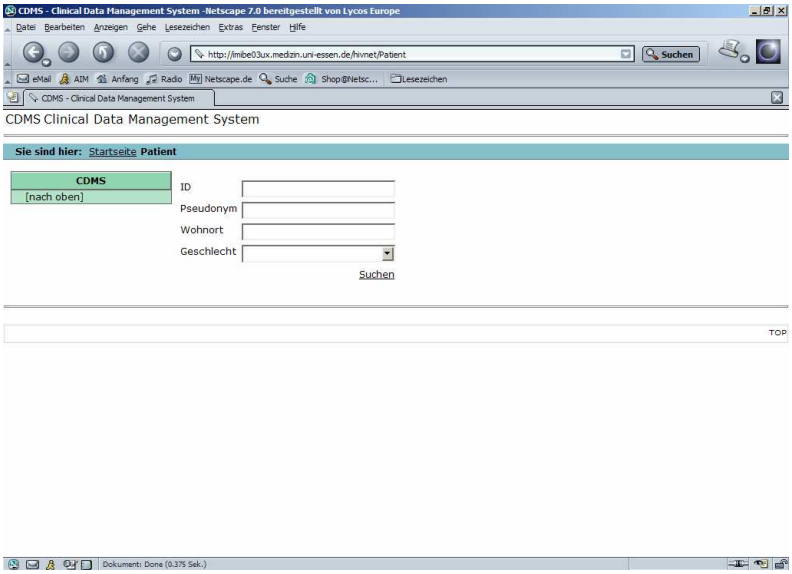


Abbildung 14: CDMS Suchauswahl

Die Ergebnisse der durchgeführten Suche werden in einer Ergebnisliste angezeigt, wie sie in Abbildung 15 dargestellt ist. Dokumentiert werden die durch Ergebnis-Integritätsbedingungen definierten Merkmalsarten.

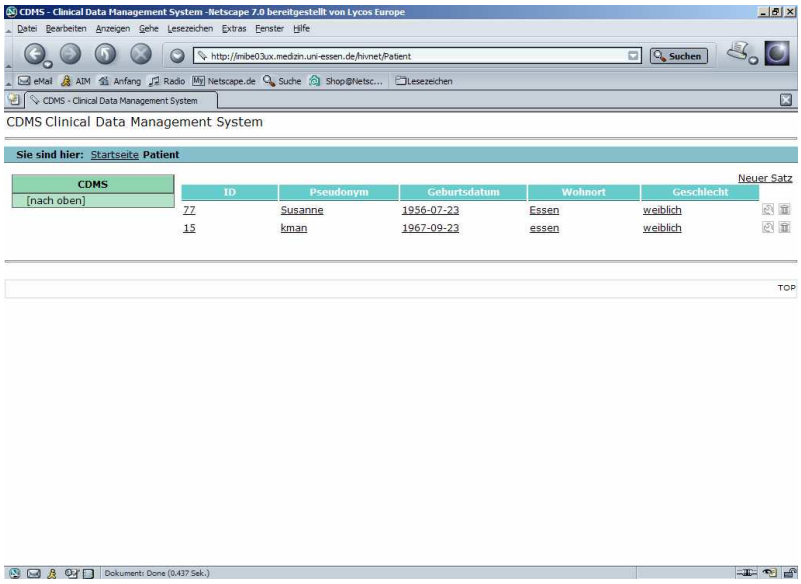


Abbildung 15: CDMS Ergebnisliste

Aus der Liste heraus kann der Benutzer Datensätze zur weiteren Verwendung auswählen. Ihm stehen die Funktionen Anzeige, Bearbeiten und Löschen zur Verfügung. Abbildung 16 zeigt die Detailanzeige eines Datensatzes.

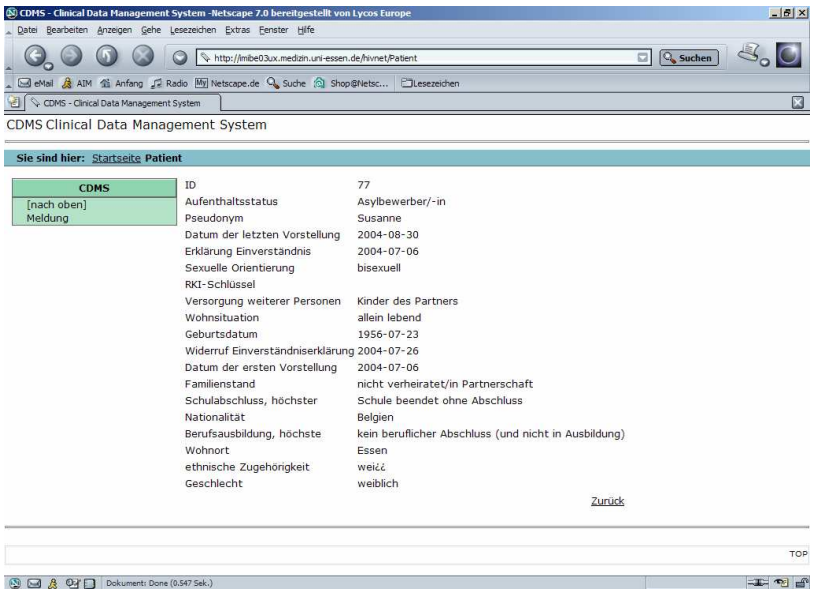


Abbildung 16: CDMS Detailanzeige

5 Ergebnisse Teil C: Integritätsprüfungen

5.1 Einleitung

Anforderungen an die Datenintegrität einer medizinischen Dokumentation haben entscheidende Auswirkungen auf Entwurf und Implementierung eines Dokumentationssystems. Die in Kapitel 3 vorgestellten Modellintegritätsbedingungen müssen auf der Implementierungsseite abgebildet werden, da jede modellierte Integritätsbedingung auch Effekte auf das generierte medizinische Dokumentationssystem hat. Diese Implikationen können sich auf jeweils unterschiedliche Ebenen erstrecken: auf die Datenhaltung (Model), die Anwendungsschicht (Controller) oder die Präsentation (View).

Je nach betroffener Architekturebene stellt sich auch die Integration der Integritätsbedingungen in das Applikationsframework unterschiedlich dar.
Insbesondere für Integritätsbedingungen der Controller-Ebene ist es notwendig, die durch das Framework vorgegebene Funktionalität dynamisch zu erweitern, indem zusätzliche Prüfalgorithmen eingebunden werden können. Daher steht im Framework eine Schnittstelle *Constraint* bereit (siehe Abbildung 17).

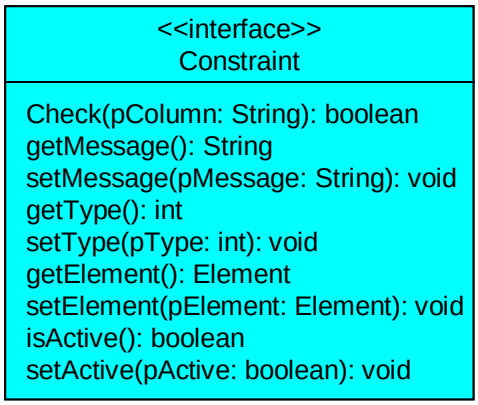


Abbildung 17: Schnittstelle Constraint (Framework)

Modellierte Integritätsbedingungen, die Auswirkungen auf der Controller-Ebene des Applikationsframeworks haben, werden als Implementierung dieser Schnittstelle in die Applikation eingefügt.

Eine wichtige Methode dieser Schnittstelle ist die Methode `check(…)`, in welcher jeweils der Prüfalgorithmus der Bedingung programmiert ist. Im Fall der Verletzung der Integritätsbedingung wird eine `ConstraintException` geworfen, die durch das Framework entsprechend ihrer Ausprägung behandelt wird.

5.2 Implementierte Integritätsbedingungen

In den folgenden Abschnitten werden die im Rahmen dieser Arbeit entworfenen und implementierten Integritätsbedingungen vorgestellt und dabei insbesondere hinsichtlich ihrer Auswirkungen auf das Applikationsframework untersucht.

5.2.1 Datentyp-Integritätsbedingung (TypeConstraint)

Schnittstelle:

Die Schnittstelle der Datentyp-Integritätsbedingung wird in Abbildung 18 dargestellt.

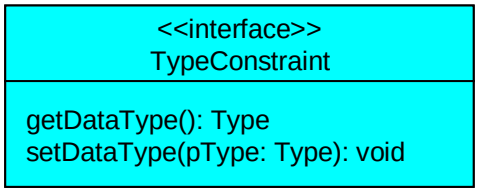


Abbildung 18: Schnittstelle TypeConstraint

Auswirkungen auf Model-Ebene:

Die Datentyp-Integritätsbedingung hat zunächst direkte Auswirkungen auf die Erzeugung der Datenbasis, indem der in ihr hinterlegte abstrakte Datentyp in einen konkreten Datentyp der verwendeten Datenbank transformiert wird. Der modellierte Datentyp *integer* wird beispielsweise in einer MySQL-Datenbank zum Datentyp *INT*. (Bereits hier zeigen sich erhebliche Unterschiede je nach verwendeter Datenbank und damit der Nutzen einer generischen Lösung, wie sie hier vorgestellt wird. In einer Oracle-Datenbank ist der entsprechende Datentyp *NUMBER*.)

Auswirkungen auf Controller-Ebene:

Theoretisch wäre es möglich, die Funktionalität der eingesetzten Datenbank zu nutzen, um eine Eingabe falscher Datentypen zu verhindern. Um aber von den möglichen (und eventuell unterschiedlichen) Reaktionen zu abstrahieren, wurde eine Abstraktionsebene in das Framework eingefügt, welche unabhängig von der jeweiligen Datenbank eine Überprüfung der eingegebenen Daten vornimmt.

Auswirkungen auf View-Ebene:

Keine.

5.2.2 Datenlänge-Integritätsbedingung (LengthConstraint)

Schnittstelle:

Die Schnittstelle der Datenlänge-Integritätsbedingung wird in Abbildung 19 dargestellt.

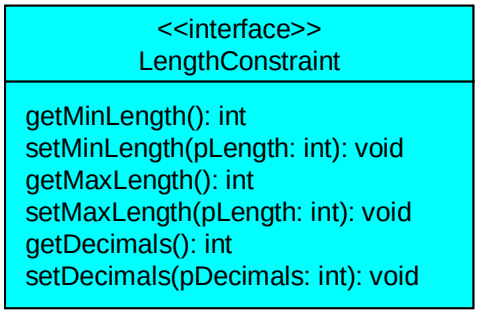


Abbildung 19: Schnittstelle LengthConstraint

Auswirkungen auf Model-Ebene:

Die Angaben in der Datenlänge-Integritätsbedingung schlagen sich direkt in Form von Längenangaben bei der Erstellung der Datenbank nieder, wie folgendes Beispiel zeigt:

```
CREATE TABLE Befund ( ID NUMBER (10) NOT NULL, …)
```

Auswirkungen auf Controller-Ebene:

Wie bei der Überprüfung des Datentyps wird auch bei der Überprüfung der Datenlänge von der Datenhaltungsschicht abstrahiert.

Auswirkungen auf View-Ebene:

Keine. (Die Datenlänge könnte sich auch auf die Benutzeroberfläche auswirken, indem die Größe der Eingabefelder entsprechend dieser Länge angepasst wird. Aus Design-Gründen wurde allerdings die Länge der Eingabefelder für alle Merkmalsarten gleich definiert.)

5.2.3 Wertebereich-Integritätsbedingung (DomainConstraint)

Schnittstelle:

Die Schnittstelle der Wertebereich-Integritätsbedingung wird in Abbildung 20 dargestellt.

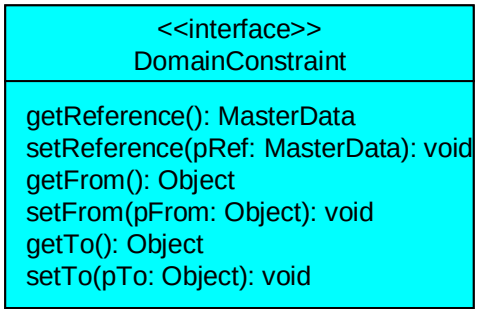


Abbildung 20: Schnittstelle DomainConstraint

Auswirkungen auf Model-Ebene:

Stammdaten-Referenzen werden auf Datenhaltungs-Ebene als Fremdschlüsselbeziehungen abgebildet.

Auswirkungen auf Controller-Ebene:

Die Prüfung der from/to-Einschränkungen erfolgt auf Controller-Ebene. Die Controller-Ebene stellt zusätzlich sicher, dass in Verbindung mit der View-Ebene bei Stammdaten-Referenzen keine anderen Angaben gemacht werden können, als in der jeweiligen Stammdaten-Tabelle vorhanden sind.

Auswirkungen auf View-Ebene:

Für Stammdaten-Referenzen werden Eingabefelder in Form von Comboboxen dargestellt. Eine Combobox ist ein Darstellungselement (*Widget*) der Grafischen Benutzeroberfläche, das eine beschränkte Auswahl an Feldinhalten zulässt (Beispiel siehe Abbildung 21).

Abbildung 21: Stammdaten in Combobox

5.2.4 Schlüssel-Integritätsbedingung (KeyConstraint)

Schnittstelle:

Die Schnittstelle der Schlüssel-Integritätsbedingung wird in Abbildung 22 dargestellt.

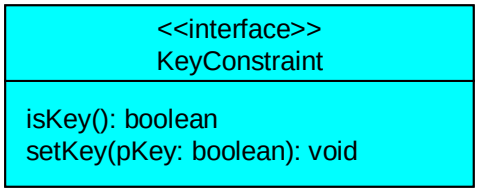


Abbildung 22: Schnittstelle KeyConstraint

Auswirkungen auf Model-Ebene:

Eine Schlüssel-Integritätsbedingung geht als Schlüssel-Constraint in die Datenbasis ein. Das heißt insbesondere, dass seine Ausprägungen eindeutig und gefüllt (Pflichtmerkmal) sein müssen.

Auswirkungen auf Controller-Ebene:

Schlüsselfelder werden auf Controller-Ebene dazu benutzt, Datensätze eindeutig zu identifizieren. Dies ist im Besonderen für die Navigation zwischen Kontexten (Fremdschlüsselbeziehungen) relevant.

Auswirkungen auf View-Ebene:

Schlüsselfelder sind automatisch Elemente der Such- und Ergebnismenge zu einem Kontext und werden daher auch in den entsprechenden Bildschirmpräsentationen angeboten.

5.2.5 Anzahl-Integritätsbedingung (QuantityConstraint)

Schnittstelle:

Die Schnittstelle der Anzahl-Integritätsbedingung wird in Abbildung 23 dargestellt.

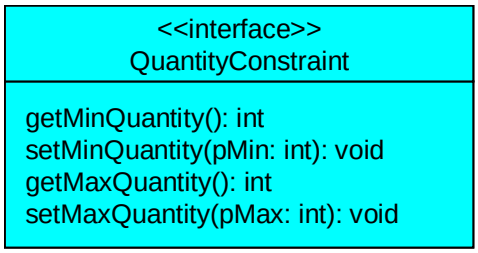


Abbildung 23: Schnittstelle QuantityConstraint

Auswirkungen auf Model-Ebene:

Da sich bei Verwendung des relationalen Datenmodells der Normalisierungsprozess dahingehend auswirkt, dass ein Kontext nicht mehrere Ausprägungen zu einer Merkmalsart enthalten darf, sind praktisch nur folgende Kombinationen für minQuantity/maxQuantity sinnvoll und möglich:

- (0,0): Merkmalsart darf keine Ausprägung haben
- (0,1): eine Ausprägung darf, muss aber nicht vorhanden sein
- (1,1): Pflichtmerkmalsart; eine Ausprägung ist zwingend anzugeben

Auswirkungen auf Controller-Ebene:

Auch diese Integritätsbedingung wird auf Controller-Ebene getestet.

Auswirkungen auf View-Ebene:

Keine.

5.2.6 Referenz-Integritätsbedingung (ReferenceConstraint)

Schnittstelle:

Die Schnittstelle der Referenz-Integritätsbedingung wird in Abbildung 24 dargestellt.

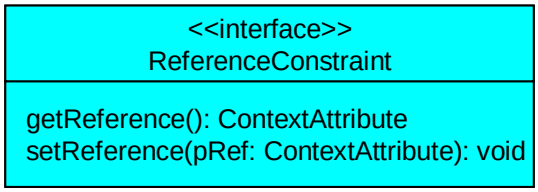


Abbildung 24: Schnittstelle ReferenceConstraint

Auswirkungen auf Model-Ebene:

Eine Referenz-Integritätsbedingung geht als Fremdschlüsselbeziehung in die Model-Ebene ein.

Auswirkungen auf Controller-Ebene:

Eine Prüfung erfolgt auf Controller-Ebene in Kombination mit der View-Ebene durch Beschränkung der Eingabemöglichkeiten.

Auswirkungen auf View-Ebene:

Auch für Referenz-Integritätsbedingungen kommt eine Combobox zum Einsatz, welche die Ausprägungsauswahl auf die Menge der Elemente des referenzierten Kontextes beschränkt.

5.2.7 Suche-Integritätsbedingung (SearchConstraint)

Schnittstelle:

Die Schnittstelle der Suche-Integritätsbedingung wird in Abbildung 25 dargestellt.

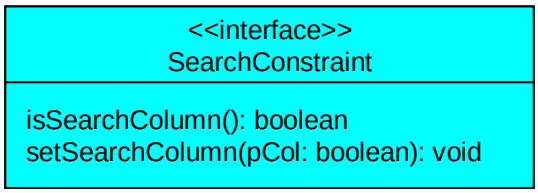


Abbildung 25: Schnittstelle SearchConstraint

Auswirkungen auf Model-Ebene:

Keine.

Auswirkungen auf Controller-Ebene:

Auf Controller-Ebene wird die Steuerung der Selektionsmechanismen abgebildet.

Auswirkungen auf View-Ebene:

Eine Such-Merkmalsart ist Bestandteil des Suchbildschirms eines Kontextes (siehe Abbildung 14).

5.2.8 Ergebnis-Integritätsbedingung (ResultConstraint)

Schnittstelle:

Die Schnittstelle der Ergebnis-Integritätsbedingung wird in Abbildung 26 dargestellt.

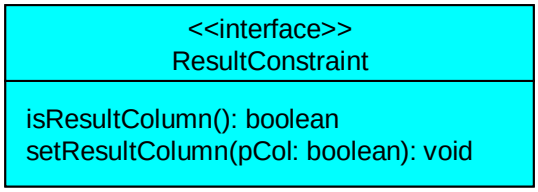


Abbildung 26: Schnittstelle ResultConstraint

Auswirkungen auf Model-Ebene:

Keine.

Auswirkungen auf Controller-Ebene:

Die Controller-Ebene steuert den Datenfluss zwischen Datenebene und Präsentation.

Auswirkungen auf View-Ebene:

Eine Ergebnis-Merkmalsart ist Bestandteil des Ergebnisbildschirms eines Kontextes, welcher die Ergebnisse eines Suchvorgangs in Listenform darstellt (siehe Abbildung 15).

5.2.9 Hierarchie-Integritätsbedingung (HierarchyConstraint)

Schnittstelle:

Die Schnittstelle der Hierarchie-Integritätsbedingung wird in Abbildung 27 dargestellt.

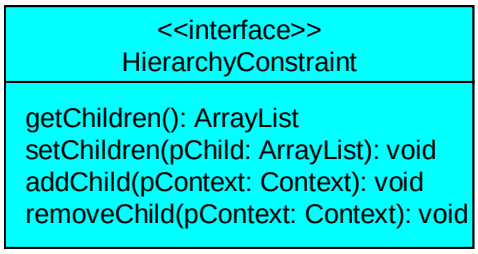


Abbildung 27: Schnittstelle HierarchyConstraint

Auswirkungen auf Model-Ebene:

Der Hierarchie entsprechende Referenz-Integritätsbedingungen müssen modelliert sein.

Auswirkungen auf Controller-Ebene:

Die Verknüpfungen zwischen durch Hierarchie-Integritätsbedingungen verbundenen Kontexten werden auf Controller-Ebene durch vorhandene Fremdschlüssel-Beziehungen der Modellebene hergestellt.

Auswirkungen auf View-Ebene:

Hierarchie-Integritätsbedingungen wirken sich auf die Menü- und Pfadstruktur der Anwendung und damit direkt auf die Navigierbarkeit aus.

5.2.10 Semantische Integritätsbedingung (SemanticConstraint)

Schnittstelle:

Die Schnittstelle der semantischen Integritätsbedingung wird in Abbildung 28 dargestellt.

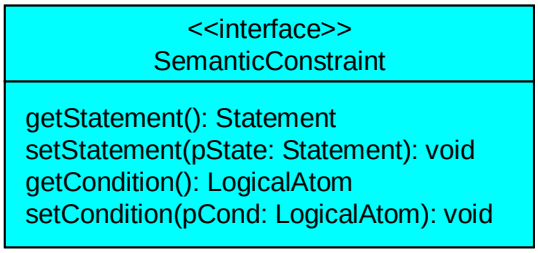


Abbildung 28: Schnittstelle SemanticConstraint

Auswirkungen auf Model-Ebene:

Keine.

Auswirkungen auf Controller-Ebene:

Bei der semantischen Integritätsbedingung handelt es sich bei weitem um die komplexeste. Mit ihr lassen sich beliebige, logisch verknüpfte Abhängigkeiten auf semantischer Ebene prüfen. Eine Einschränkung ergibt sich lediglich aus der fehlenden Integration von Formeln (siehe hierzu Kapitel 6).

Jede semantische Integritätsbedingung besitzt eine Aussage (`statement`) und kann zusätzlich eine Bedingung (`condition`) enthalten, unter der diese Aussage zur Anwendung kommt. Die Bedingung wiederum besteht aus logisch verknüpften Teilaussagen.

Anhand eines Beispiels soll die Verwendung semantischer Integritätsbedingungen verdeutlicht werden. Aus der Anforderung

„Nur weibliche Patienten können schwanger werden."

wird die Integritätsbedingung

„Die Merkmalsart *Geschlecht* im Kontext *Patient* muss die Ausprägung ‚weiblich' haben, wenn die Merkmalsart *Schwangerschaft* die Ausprägung ‚ja' hat."

Für dieses Beispiel wurde im Abschnitt 3.3.10 das Modell gezeigt. Die Aussage, dass das Geschlecht eines Patienten ‚weiblich' sein muss, zeigt Abbildung 29 im Entwurfsmodul (Die Stammdaten-Codierung für ‚weiblich' ist ‚2').

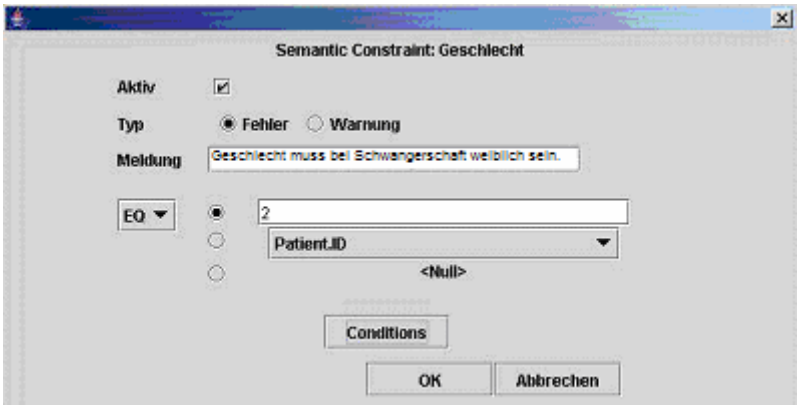


Abbildung 29: Semantische Integritätsbedingung - Aussage

Diese Aussage soll genau dann gelten, wenn für den Patienten eine Schwangerschaft vorliegt (Die Stammdaten-Codierung für ‚Schwangerschaft' ist ‚2'). Diese Bedingung, welche mit der obigen Aussage verknüpft ist, wird in Abbildung 30 gezeigt.

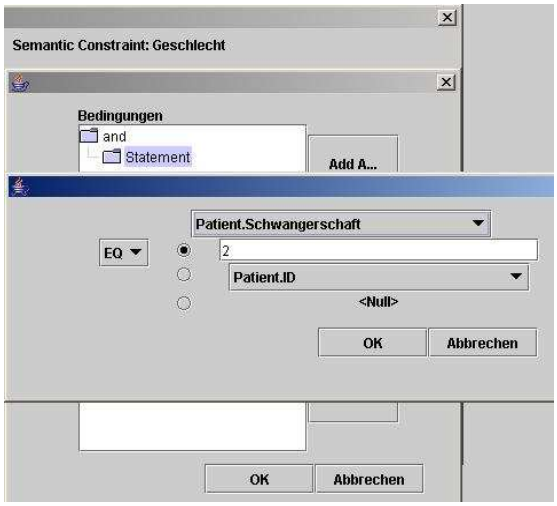


Abbildung 30: Semantische Integritätsbedingung - Bedingung

Die Überprüfung einer semantischen Integritätsbedingung in der Applikation läuft wie folgt ab (Abbildung 31):

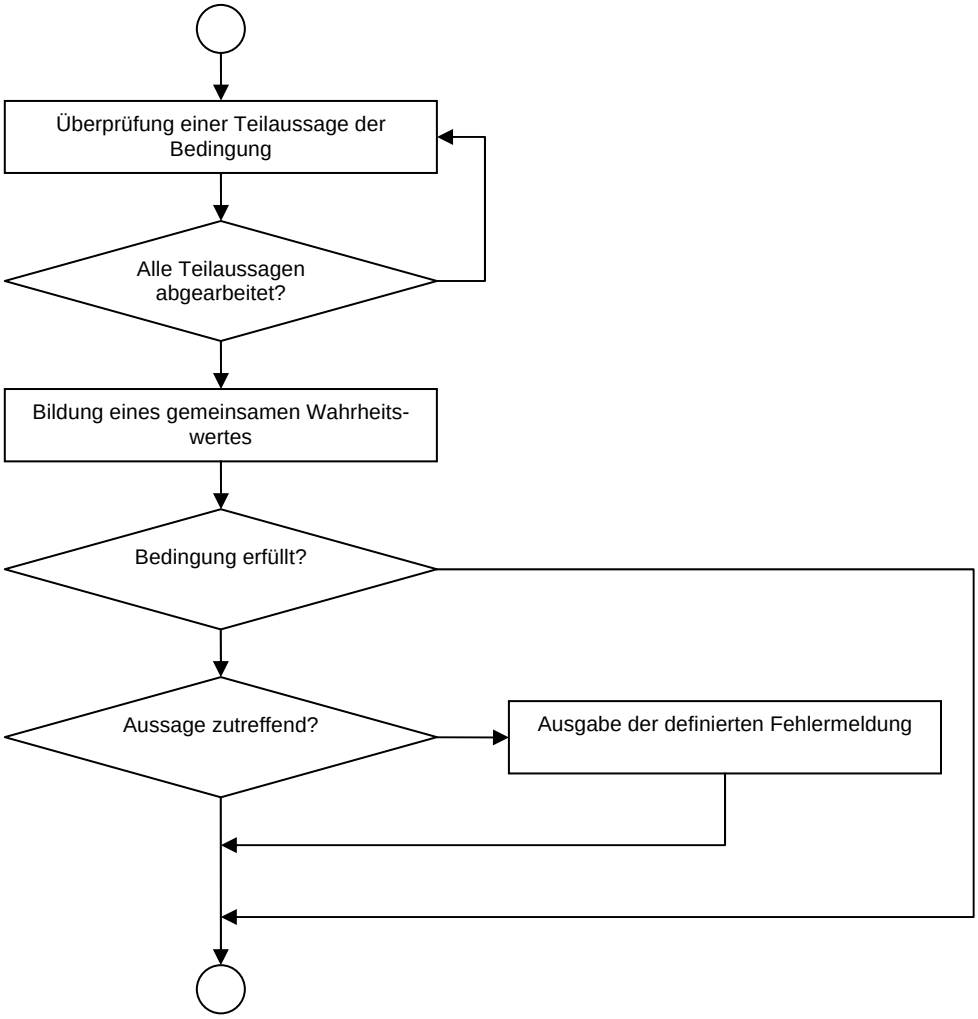


Abbildung 31: Überprüfung einer Semantischen Integritätsbedingung

Die Überprüfung der Teilaussagen einer Bedingung sowie die Bildung des Gesamtwahrheitswertes geschehen dabei rekursiv (Ausschnitt siehe Abbildung 32). Referenzen auf Ausprägungen anderer Merkmalsarten - auch anderer Kontexte - werden durch einen Algorithmus aufgelöst und durch ihre konkreten Ausprägungen zur Laufzeit ersetzt.

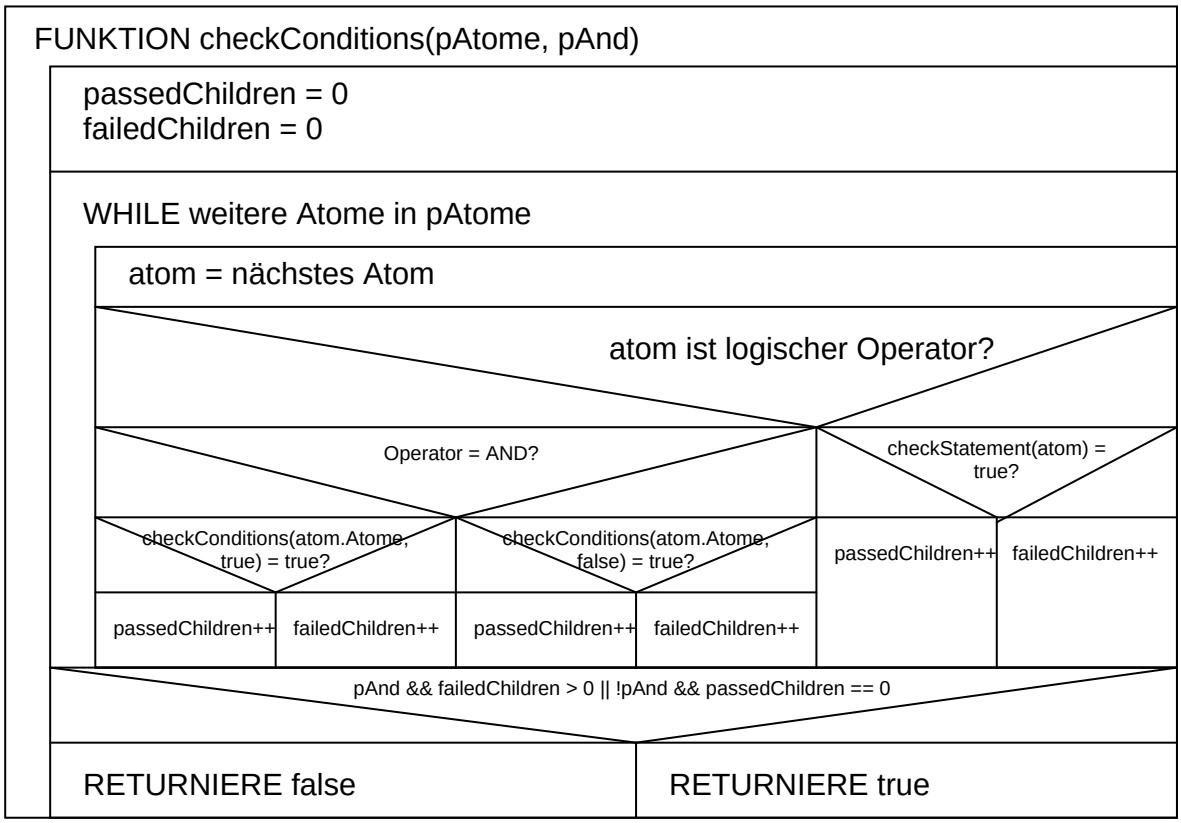


Abbildung 32: Wahrheitswertfindung einer Bedingung

Auswirkungen auf View-Ebene:

Keine.

5.2.11 Weitere Integritätsbedingungen

Die Integration weiterer Integritätsbedingungen in das Applikationsframework und damit eine Erweiterung der Plausibilitätsprüfungen ist leicht möglich. Die hinzuzufügende Bedingung muss die Schnittstelle `constraints.Constraint` des Applikationsframeworks implementieren, sich im selben Package befinden und folgende Namenskonvention befolgen:

```
<Typeref>ConstraintImpl
```

`<Typeref>` ist dabei der im Dokumentationsmodell verwendete Integritätsbedingungsbezeichner beginnend mit einem Großbuchstaben. Es soll z. B. eine Wertprüfungsbedingung implementiert werden, die wie folgt definiert ist:

```
<constraint type="value" value="25" message="Fehlerhafter Wert"/>
```

Dann muss die Prüfklasse den Namen `ValueConstraintImpl` bekommen. Über den Reflection-Mechanismus von Java wird diese Klasse automatisch in das System integriert und zur Prüfung herangezogen.

5.3 Prüfprozess

Bei jeder Datenmanipulation werden für die beteiligten Merkmalsarten alle vorhandenen Integritätsbedingungen ausgewertet und im Fall einer Verletzung die modellierten Meldungstexte zur Ansicht gebracht. Handelt es sich bei der betroffenen Integritätsbedingung um eine Bedingung vom Typ Fehler, so ist eine Speicherung der fehlerhaften Daten nicht möglich (Abbildung 33). Handelt es sich lediglich um Warnungen, so können die Daten nach erfolgter Kenntnisnahme gespeichert werden.

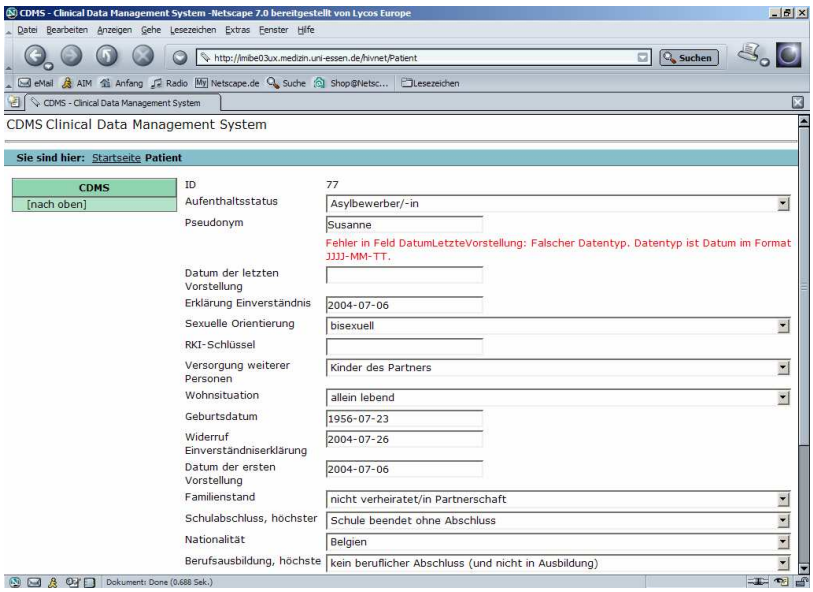


Abbildung 33: Datenprüfung

6 Diskussion

Diese Arbeit stellt Konzepte vor, mit denen sich die Qualität der in einem Dokumentationssystem erfassten und verwalteten Daten nachhaltig verbessern lässt. Durch die Definition eines Modells zur Beschreibung der Anforderungen medizinischer Dokumentationssysteme wird die Grundlage geschaffen, die Erfordernisse einer Erhebung vollständig und unabhängig von einer späteren Umsetzung zu formulieren. Der Einsatz von XML als Beschreibungsmittel der Modellierung ist zugleich dazu geeignet, eine automatisierte Generierung zu unterstützen.

Ein wichtiger Aspekt bei der Modellierung einer Anwendung ist ein Anforderungsmanagement, das auf eine vollständige Erfassung aller bekannten Rahmenbedingungen ausgerichtet ist. Ein System kann stets nur die bekannten, expliziten Anforderungen erfüllen. Essenziell hierfür ist eine frühzeitige und dauerhafte Einbindung der späteren Benutzer in den Modellierungsprozess sowie deren daraus resultierende Sensibilisierung hinsichtlich des Nutzens durch vielfach zuverlässigere und plausiblere Daten. Dies wird durch den Einsatz des Entwurfsmoduls des integrierten Systems *InCoMe* gefördert, über welches ein Benutzer gemeinsam mit einem Systemdesigner alle Anforderungen mittels einer grafischen Oberfläche modelliert. Untersuchungen haben ergeben, dass gerade solche personenbedingte Faktoren Auswirkungen auf die Qualität einer Erhebung haben können (Stausberg 1997). Beispiele für weitere derart ‚weiche' Einflüsse auf eine Verbesserung der Datenqualität sind:

- Die Datenerhebung muss für die erfassende Person und seine Arbeit erkennbare Vorteile bringen.
- Der Erfassungsvorgang ist idealerweise in die Arbeitsabläufe eingebunden.
- Die Menge der erhobenen Daten ist nicht zu groß (d. h. der Aufwand ist dem Nutzen angepasst).
- Eine Nutzung der dokumentierten Daten erfolgt auch in anderen Zusammenhängen (d. h. es findet - möglicherweise - eine Überprüfung durch andere statt).

Eine Vernachlässigung dieser Faktoren führt häufig zum so genannten „regelkonformen Schwindeln", um lästige Prüfungen zu umgehen und somit im schlimmsten Fall die Daten invalide und damit unbrauchbar zu machen.

Zur Sicherstellung der Datenqualität und eine dahingehende Unterstützung des Dokumentationspersonals ist der konsequente Einsatz von Plausibilitätsprüfungen unabdingbar (Gaus 1995, FDA 1999). Das entwickelte System *InCoMe* zielt auf eine durchgängige und

übergreifende Sicherung der Qualität auf Daten- und Systemebene ab und bildet den Rahmen für eine weitestgehend computerunterstützte Generierung medizinischer Dokumentationssysteme. Eine vollständig maschinelle Umsetzung ohne erforderliche manuelle Eingriffe wie z. B. eine Treibereinbindung für die spezifisch genutzte Datenbank oder das Starten eines Generierungsskriptes im verwendeten DBMS ist auf dem Stand heutiger Software-Technik nicht realisierbar. Dessen ungeachtet ist *InCoMe* durch die gewählten Techniken plattformunabhängig und damit auf den unterschiedlichsten Betriebssystemen lauffähig, ohne jeweils eine Umwandlung (Kompilierung) zu erfordern.

Bei der Entwicklung der Architektur wurde großer Wert auf die Kriterien Flexibilität sowie eine leichte Erweiterbarkeit der Komponenten gelegt. Mögliche zukünftige Erweiterungen sind zum Beispiel:

- Import und Export für Stammdaten und freie Merkmalsarten,
- Import aus bzw. Schnittstellen zu anderen Systemen (Dokumentationsumgebungen) sowie
- die Einbeziehung von Formeln in die Definition von semantischen Integritätsbedingungen.

Auch eine Ergänzung zusätzlicher Formen von Integritätsbedingungen ist, wie gezeigt wurde, leicht realisierbar.

Durch die Verwendung von Cascading Style Sheets (CSS) und Java Server Pages (JSP) lässt sich das Erscheinungsbild des generierten Dokumentationssystems auf einfache Weise an beliebige Design-Bedürfnisse adaptieren. Die grafische Benutzeroberfläche des Entwurfsmoduls dagegen ist statisch konzipiert, da sie nicht von einem Endbenutzer, sondern von einem Systemdesigner zu nutzen ist.

Der Einsatz von Prädikatenlogik in Form von Nebenbedingungen-unterstützenden Programmiersprachen (Jaffar, Lassez 1987; Lassez 1987) oder Wissensbasis-Regeln in Arden Syntax (Sailors 1999) böte weitere Aspekte, welche die Beschreibung und Verwendung von Integritätsprüfungen begünstigen, ist aber in der medizinischen Wissensverarbeitung bisher sehr beschränkt (Köhler, Meyer zu Bexten 2002). Durch die Definition von Regeln lassen sich Abhängigkeiten von Entitäten unter Einbeziehung eventuell benötigter weiterer Berechnungen deklarieren. Folgendes vereinfachtes Beispiel beschreibt schematisch die Beziehung der Entitäten Geburtsdatum im Kontext Patient sowie Meldedatum im Kontext Meldung:

```
meld_geburt(Patient.Geburtsdatum,Meldung.Meldedatum):-
        Meldung.Meldedatum>Patient.Geburtsdatum.
```

Für alle Vorkommnisse von Meldedatum und Geburtsdatum in den jeweiligen Kontexten muss gelten, dass das Meldedatum größer als das Geburtsdatum sein muss. Vorteil einer solchen Regel ist, dass die Beziehung nicht für jede der beteiligten Entitäten, sondern nur einmalig formuliert werden muss. Dies würde den Modellierungsaufwand weiter reduzieren. Auf Grund hierzu konkurrierender Anforderungen wie Portabilität sowie der Gefahr von Nicht-Determination bei der Regelüberprüfung wurde jedoch in dieser Arbeit auf einen Einsatz solcher Werkzeuge verzichtet.

Des Weiteren werden in der derzeitigen Version des Systems *Reverse Engineering* sowie *Modellversionierung* nicht unterstützt. Durch Reverse Engineering ließe sich aus einem vorhandenen Datenbankschema ein mit *InCoMe* les- und editierbares Modell erstellen und der Ansatz somit auf bereits bestehende Dokumentationssysteme ausweiten. Eine Modellversionierung würde es erlauben, nachträglich Modifikationen am Aufbau bereits bestehender Implementierungen vorzunehmen, ohne dass es dabei zu Datenverlusten oder Inkonsistenten käme. Allerdings wären hierzu sehr komplexe Algorithmen notwendig, die eine Transformation der Modelländerungen auf Datenbankschema- und Datenebene vornehmen.

Die Untersuchung des Einsatzes eines kommerziellen Studiensoftware-Werkzeugs für das Register des Kompetenznetzes HIV/AIDS (Nonnemacher, Weiland 2005) ergab, dass sich die der Software zu Grunde liegenden Konzepte nicht ohne weiteres auf die Dokumentation von Patientenverläufen übertragen ließen, da es insbesondere an benötigter Flexibilität fehle. Vorgeschlagen wird als Lösung die Eigenentwicklung eines auf einer SQL-Datenbank aufbauenden Frontends, welcher mit dem hier vorgestellten Ansatz verfolgt wurde.

Eine sich letztlich stellende Frage bleibt, inwiefern sich eine vollständige Integrität der verwalteten Daten überhaupt erreichen lässt. Nach Dierstein stoßen bisher alle Versuche, Plausibilität als dem Menschen immanent für seine Entscheidungen zur Verfügung stehende Funktion sinnvoll auf die semantische Ebene maschineller Systeme abzubilden, auf erhebliche Schwierigkeiten (Dierstein 2004).
Durch ein geeignetes, durchgängiges Vorgehensmodell (Moll, Broy 2004) sowie eine adäquate Umsetzung technischer Maßnahmen zur Sicherstellung der Integrität medizinischer Dokumentation lässt sich jedoch der Prüfprozess weitestgehend unterstützen und damit die Qualität der erhobenen Daten deutlich verbessern. Die Ergebnisse dieser Arbeit mögen dazu beitragen.

7 Zusammenfassung

Die Qualität der in einem medizinischen Dokumentationssystem erfassten Daten und damit deren Verwertbarkeit hängt wesentlich von der Unterstützung der dokumentierenden Personen in ihrem Bestreben ab, die Dokumentation vollständig und valide durchzuführen. Die hierzu eingesetzten Integritätsbedingungen sind bereits bei der Entwicklung durch ein geeignetes, durchgängiges Vorgehensmodell sowie durch eine adäquate Umsetzung bei der Implementierung eines Dokumentationssystems zu berücksichtigen. Ziel dieser Arbeit ist daher die Bereitstellung eines integrierten Systems zur computerunterstützten Entwicklung medizinischer Dokumentationssysteme unter besonderer Berücksichtigung von Integritätsbedingungen.

Zunächst wurden die Entitäten Datenbank, Kontext, Stammdaten, freie und gebundene Merkmalsart sowie Integritätsbedingung als maßgebliche Bausteine eines medizinischen Dokumentationssystems identifiziert. Auf dieser Grundlage folgte die Entwicklung eines unabhängigen, abstrakten Modells zur Beschreibung der Anforderungen an ein spezifisches System, welches sich darüber hinaus zur automatisierten Verarbeitung eignet. Der Generierungsprozess wird durch das vorgestellte integrierte System *InCoMe* unterstützt. Bestandteile der Architektur des Systems sind neben dem Modell ein Entwurfsmodul, ein Generierungsmodul und ein Applikationsframework. Mit Hilfe des Entwurfsmoduls lassen sich die Anforderungen eines Dokumentationssystems grafisch modellieren. Das Generierungsmodul erzeugt die anforderungsspezifischen Komponenten des Systems aus dem zuvor entworfenen Modell. Über das Applikationsframework wird eine Plattform zur Bereitstellung einer lauffähigen Anwendung zur Verfügung gestellt. Zentraler Bestandteil des Frameworks ist ein Algorithmus zur Auswertung und Überprüfung komplexer logischer Aussagen in Integritätsbedingungen. Die Architekturbestandteile werden in dieser Arbeit jeweils detailliert erläutert, modelliert und implementiert.

Angemessenheit und Mächtigkeit des Dokumentationsmodells sowie die Anwendbarkeit der Lösung wurden anhand des Merkmalskataloges für Erwachsene aus dem Kompetenznetz HIV/AIDS überprüft. Dieser Merkmalskatalog definiert den Datenumfang einer national organisierten, umfassenden und repräsentativen, prospektiven Klinischen Kohorte. Mit *InCoMe* ließen sich sowohl die dort definierten Merkmale als auch exemplarische Erweiterungen problemlos umsetzen. Die Lösung selbst bedarf für einen praktischen Einsatz noch ergänzender Module insbesondere zur Kommunikation, das Dokumentationsmodell erweitert das Methodenspektrum für Entwicklung und Betrieb medizinischer Dokumentationssysteme.

8 Literaturverzeichnis

[1] Alexander, C., Ishikawa, S., Silverstein, M. (1977): Pattern Language: Towns, Buildings, Construction. New York: Oxford University Press.

[2] Ammenwerth, E., Haux, R., Knaup, P., Pohl, U. (2000): Computer-based documentation systems and their integration into hospital information systems. In: Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics. Vol X. 419-424.

[3] Apache (2004a): Struts. Online-Publikation; http://struts.apache.org/.

[4] Apache (2004b): Torque. Online-Publikation; http://db.apache.org/torque/.

[5] Bauknecht, K., Zehnder, C. A. (1997): Grundlagen für den Informatikeinsatz. Stuttgart: B.G. Teubner.

[6] Belli, F. (1988): Einführung in die logische Programmierung mit Prolog. Mannheim, Wien, Zürich: BI-Wissenschafts-Verlag.

[7] Booch, G. (1994): Object-oriented design with applications. 2nd Edition. Redwood City: Benjamin/Cummings.

[8] Bundesärztekammer (2004): Glossar Ärztliches Qualitätsmanagement. Online-Publikation; http://www.bundesaerztekammer.de/30/Qualitaetssicherung/78Glossar.html.

[9] Bundesamt für Sicherheit in der Informationstechnik BSI (2004): IT-Grundschutzhandbuch, Kapitel M 2.130, Online-Publikation; http://www.bsi.de/gshb/deutsch/m/m02130.html.

[10] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. (1996): Pattern-Oriented Software Architecture: A System Of Patterns. West Sussex, England: John Wiley & Sons Ltd.

[11] Campbell H., Sweatman, J. (2000): Quality Assurance and clinical data mangaement. In: Rondel, R. (Ed.): Clinical Data Management. 2. Ed. Chichester: Wiley; S. 123-141.

[12] Chen, P. (1976): The Entity-Relationship Model, Toward a Unified View of Data, ACM ToDS 1:1, 9-36.

[13] Clayton, P.D., Narus, S.P., Huff, S.M. et al. (2003): Building a comprehensive clinical information system from components. The approach at Intermountain Health Care. Methods Inf Med 42(1):1-7.

[14] Codd, E.F. (1970): A relational model for large shared data banks, Comm. ACM 13:6, 377-387.

[15] Davis, M. (2001): Struts, an Open Source MVC Implementation. Online-Publikation; http://www-106.ibm.com/developerworks/library/j-struts/?dwzone=java.

[16] Dierstein, R. (2004): Sicherheit in der Informationstechnik. Informatik Spektrum 27:4, 343-353.

[17] DIN – Deutsches Institut für Normung (1996): DIN EN ISO 9241-10. Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten – Teil 10. Berlin, Wien, Zürich: Beuth.

[18] EU – Europäisches Parlament (1997) : Vorschlag für eine Richtlinie des Europäischen Parlaments und des Rates zur Angleichung der Rechts- und Verwaltungsvorschriften über die Anwendung der Guten Klinischen Praxis bei der Durchführung von klinischen Prüfungen mit Humanarzneimitteln. Amtsblatt Nr. C 306, S. 9-15.

[19] Flanagan, D. (1999): Java in a Nutshell. 3rd Edition. Köln: O'Reilly.

[20] FDA - Food and Drug Administration (1999): Guidance for Industry. Computerized Systems used in Clinical Trials. Online-Publikation; http://www.fda.gov/ora/compliance_ref/bimo/ffinalcct.htm.

[21] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995): Design Patterns. Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley.

[22] Gaus, W. (1995): Dokumentations- und Ordnungslehre. 2. Auflage. Berlin: Springer.

[23] Goertzen, R., Stausberg, J. (2004): Ein Modell zur computerunterstützten Generierung von medizinischen Dokumentationssystemen. Ger Med Sci GM04gmds305. Online-Publikation; http://www.egms.de/en/meetings/gmds2004/04gmds305.shtml.

[24] Grässle, P., Baumann, H., Baumann, P. (2000): UML projektorientiert. Bonn: Galileo Computing.

[25] Härder, T. (2003): Integritätskontrolle und aktives Verhalten. Skript zur Vorlesung Datenbankanwendung. Technische Universität Kaiserslautern. Online-Publikation; http://wwwdvs.informatik.uni-kl.de/courses/DBAW/WS0304/Vorlesungsunterlagen/Kapitel.09.full.pdf.

[26] Härder, T., Rahm E. (2001): Datenbanksysteme – Konzepte und Techniken der Implementierung. Berlin, Heidelberg: Springer.

[27] Harold, E. R., Means, W. S. (2001): XML in a Nutshell. Köln: O'Reilly.

[28] Hesse, W., Barkow, G., von Braun, H., Kittlaus, H.-B., Scheschonk, G. (1994): Terminologie in der Softwaretechnik - Ein Begriffssystem für die Analyse und Modellierung von Anwendungssystemen. Teil 1: Begriffssystematik und Grundbegriffe. Informatik Spektrum 17, 39-47.

[29] Hesse, W., Barkow, G., von Braun, H., Kittlaus, H.-B., Scheschonk, G. (1994): Terminologie in der Softwaretechnik - Ein Begriffssystem für die Analyse und Modellierung von Anwendungssystemen. Teil 2: Tätigkeits- und ergebnisbezogene Elemente. Informatik Spektrum 17, 96-105.

[30] Institut für Medizininformatik, Biometrie und Epidemiologie; Lehrstuhl für Medizinische Informatik: Definition grundlegender Begriffe aus den Themenbereichen Medizinische Dokumentation und Informationssysteme im Gesundheitswesen. Online-Publikation; http://www.imi.med.uni-erlangen.de/lehre/ss04/gloss_zus.html.

[31] Institut für Medizinische Informatik, Biometrie und Epidemiologie, Universitätsklinikum Essen (2004a): Kompetenznetz HIV/AIDS: Erhebungshandbuch Erwachsene.

[32] Institut für Medizinische Informatik, Biometrie und Epidemiologie, Universitätsklinikum Essen (2004b): Kompetenznetz HIV/AIDS: Anwenderhandbuch Erwachsene.

[33] Institut für Medizinische Informatik, Biometrie und Epidemiologie, Universitätsklinikum Essen (2004b): Kompetenznetz HIV/AIDS: Handbuch zum Datenimport.

[34] Jaffar, J., Lassez, J.J. (1987): Constraint Logic Programming. Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 111-119.

[35] Jakarta (2004): Tomcat. Online-Publikation; http://jakarta.apache.org/tomcat/.

[36] Kelter, U. (1991): CASE – Computer Aided Software Engineering. Berlin, Heidelberg: Springer. Auch: http://www.gi-ev.de/informatik/lexikon/inf-lex-case.shtml.

[37] Kittlaus, H. B. (2003): Software-Engineering und Software-Fabrik. Informatik Spektrum 18, 8-12.

[38] Knaup, P., Garde, S., Haux, R. (2004): Ein Meta-Modell für kooperative Dokumentationsumgebungen. Ger Med Sci GM04gmds305. Online-Publikation; http://www.egms.de/en/meetings/gmds2004/04gmds033.shtml.

[39] Köhler, C. O. (2003): Historie der Medizinischen Informatik in Deutschland von den Anfängen bis 1980. Keynote Medizinische Informatik. 48. Jahrestagung der GMDS 2003. Online-Publikation;
http://www.gmds.de/texte/onlinedocs/fachbereich_medinf/fbmi_vortrag_koehler.pdf.

[40] Köhler, C. O., Meyer zu Bexten, E., Lehmann, T. M. (2002): Historie und Umfeld. In Lehmann, Meyer zu Bexten (Hrsg.): Handbuch der Medizinischen Informatik. München, Wien: Carl Hanser Verlag.

[41] Lassez, C. (1987): Constraint Logic Programming. BYTE 12:9, 171-176.

[42] Leiner, F., Gaus, W., Haux, R. (2003): Medizinische Dokumentation. 4. Auflage. Stuttgart: Schattauer.

[43] Leiner, F., Haux, R. (1996): Systematic Planning of Clinical Documentation. Methods Inf Med 35, 25-34.

[44] Lenz, R., Elstner, T., Siegele, H., Kuh, K. A. (2002): A Practical Approach to Process Support in Health Information Systems. J Am Med Inform Assoc 9:6, 571-585.

[45] Mahmoud, Q. H. (2003): Servlets and JSP Pages Best Practice. Online-Publikation; http://java.sun.com/developer/technicalArticles/javaserverpages/servlets_jsp/.

[46] McLaughlin, B. (2001): Java und XML. Köln: O'Reilly.

[47] Mludek, V. (2001): Modellierung der Integritätsbedingungen eines rechnerunterstützten Dokumentationssystems zur Studiendokumentation in einem Verbund multizentrischer Therapiestudien der Pädiatrischen Onkologie. Dissertation; Universität Kassel.

[48] Moll, K.-R., Broy, M., Pizka, M., Seifert, T., Bergner, K., Rausch, A. (2004): Erfolgreiches Management von Software-Projekten. Informatik Spektrum 27:5, 419-432.

[49] Nonnemacher, M., Weiland, D., Hildebrand, J., Stausberg, J. (2005): Nutzung von Studiensoftware zur Online-Erfassung in Patientenregistern: Erfahrungen und Empfehlungen aus dem Kompetenznetz HIV/AIDS. In: Jäckel (Hrsg.): Telemedizinführer Deutschland, Ausgabe 2005. Ober-Mörlen.

[50] Oestereich, B. (1998): Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language. 4. Auflage. München, Wien: R. Oldenbourg Verlag.

[51] Patel, P. (2000): Data Validation. In: Rondel, R. (Ed.): Clinical Data Management. 2. Ed. Chichester: Wiley; S. 109-121.

[52] Royce, W. W. (1970): Managing the Development of Large Software Systems. In: Proceedings of IEEE Wescon, Aug. 1970; S. 1-9.

[53] Sailors, R. M. (1999): A Gentle Introduction to Arden Syntax. Online-Publikation; http://utsurg.uth.tmc.edu/Medical_Informatics/projects/ArdenSyntax/GentleIntro/index.htm.

[54] Schwarze, J. (1994): Grundlagen der Statistik. Herne, Berlin: Verlag Neue Wirtschafts-Briefe.

[55] Seelos, H.-J. (1990): Wörterbuch der Medizinischen Informatik. Berlin: de Gruyter.

[56] Sens, B., Fischer, B. (2003): Begriffe und Konzepte des Qualitätsmanagements. In: Informatik, Biometrie und Epidemiologie in Medizin und Biologie. 34:1, 1-64.

[57] Stausberg, J. (1997): Evaluation of Quality Control Methods for Medical Documentation. Stud Health Technol Inform. 43 Pt B:864-8.

[58] Sun Microsystems (2004a): Java. Online-Publikation; http://java.sun.com/.

[59] Sun Microsystems (2004b): Java Server Pages. Online-Publikation; http://java.sun.com/products/jsp/.

[60] Sun Microsystems (2004c): Java Servlets. Online-Publikation; http://java.sun.com/products/servlet/.

[61] van der Lans, R. F. (1990): SQL: Der ISO-Standard. München, Wien: Hanser.

[62] Vetter, M. (1993): Strategie der Anwendungssoftware-Entwicklung. Methoden, Techniken, Tools einer ganzheitlichen, objektorientierten Vorgehensweise. Stuttgart: B.G. Teubner.

[63] WHO (1992): Guidelines for good clinical practice for trials on pharmaceutical products. Online-Publikation;
http://www.who.int/medicines/library/par/ggcp/GCPGuidePharmatrials.pdf.

[64] W3C (2001). XML-Schema Spezifikation. Online-Publikation; http://www.w3.org/XML/Schema.

[65] W3C (1999). XSLT Spezifikation. Online-Publikation; http://www.w3c.org/TR/xslt.

[66] Zehnder, C.A. (1998): Informationssysteme und Datenbanken. Stuttgart: B.G. Teubner.

Alle genannten Online-Ressourcen wurden zuletzt am 20.02.2005 überprüft.

Anhang

A Abkürzungsverzeichnis

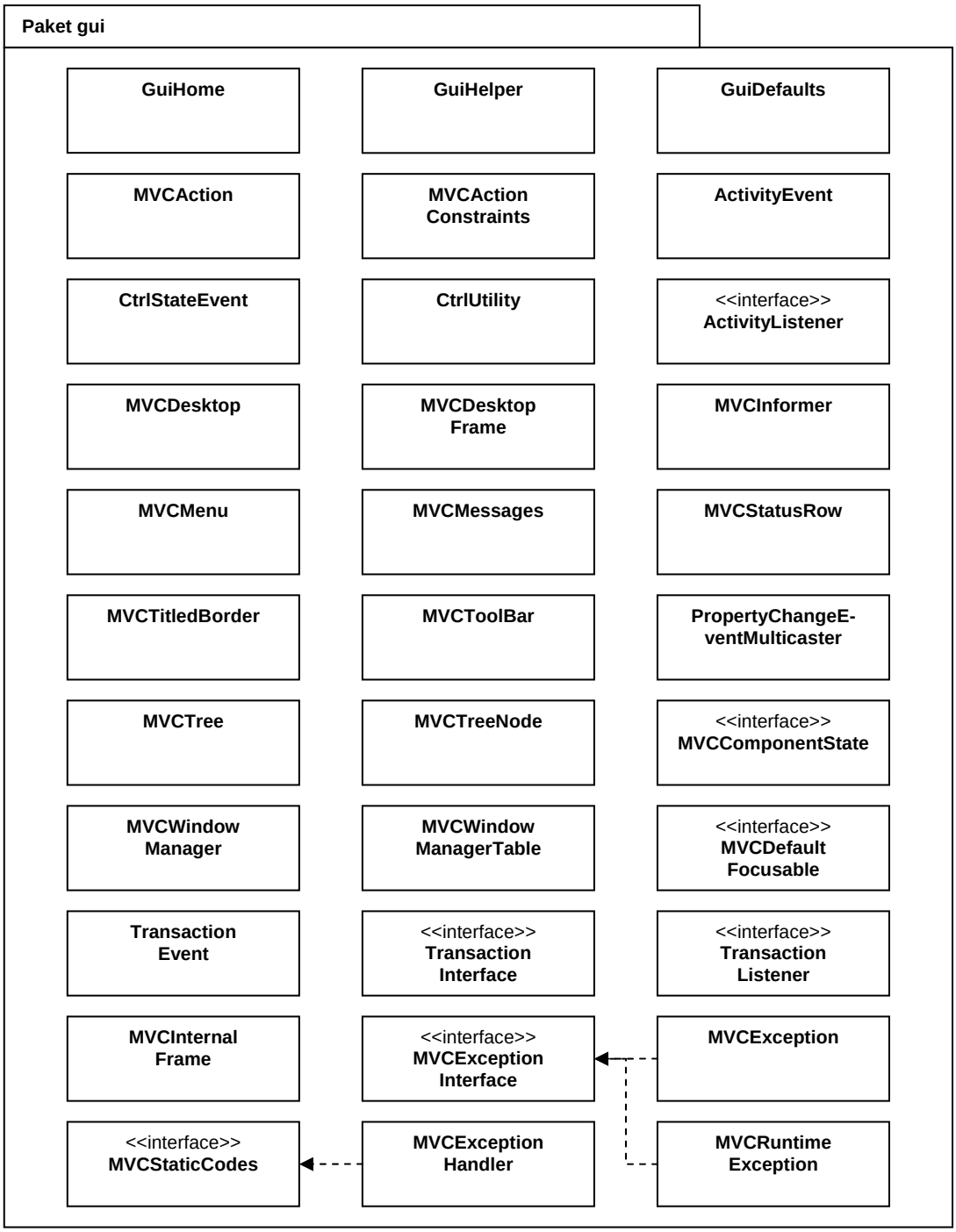
AIDS	Acquired Immune Deficiency Syndrome
BO	Business Object
BSI	Bundesamt für Sicherheit in der Informationstechnik
CDM	Clinical Data Management
CDMS	Clinical Data Management System
CSS	Cascading Style Sheets
Ctrl	Controller
DBMS	Database Management System
DTD	Document Type Definition
EU	Europäische Union
FDA	Food and Drug Administration
GCP	Good Clinical Practice
GUI	Graphical User Interface
HIV	Human Immunodeficiency Virus
HTML	Hypertext Markup Language
Impl	Implementation
JFC	Java Foundation Classes
MVC	Model View Controller
OOA	Objektorientierte Analyse
OOD	Objektorientiertes Design
SDK	Software Development Kit
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WHO	World Health Organization
XML	Extensible Markup Language
XSTL	Extensible Stylesheet Language Transformations

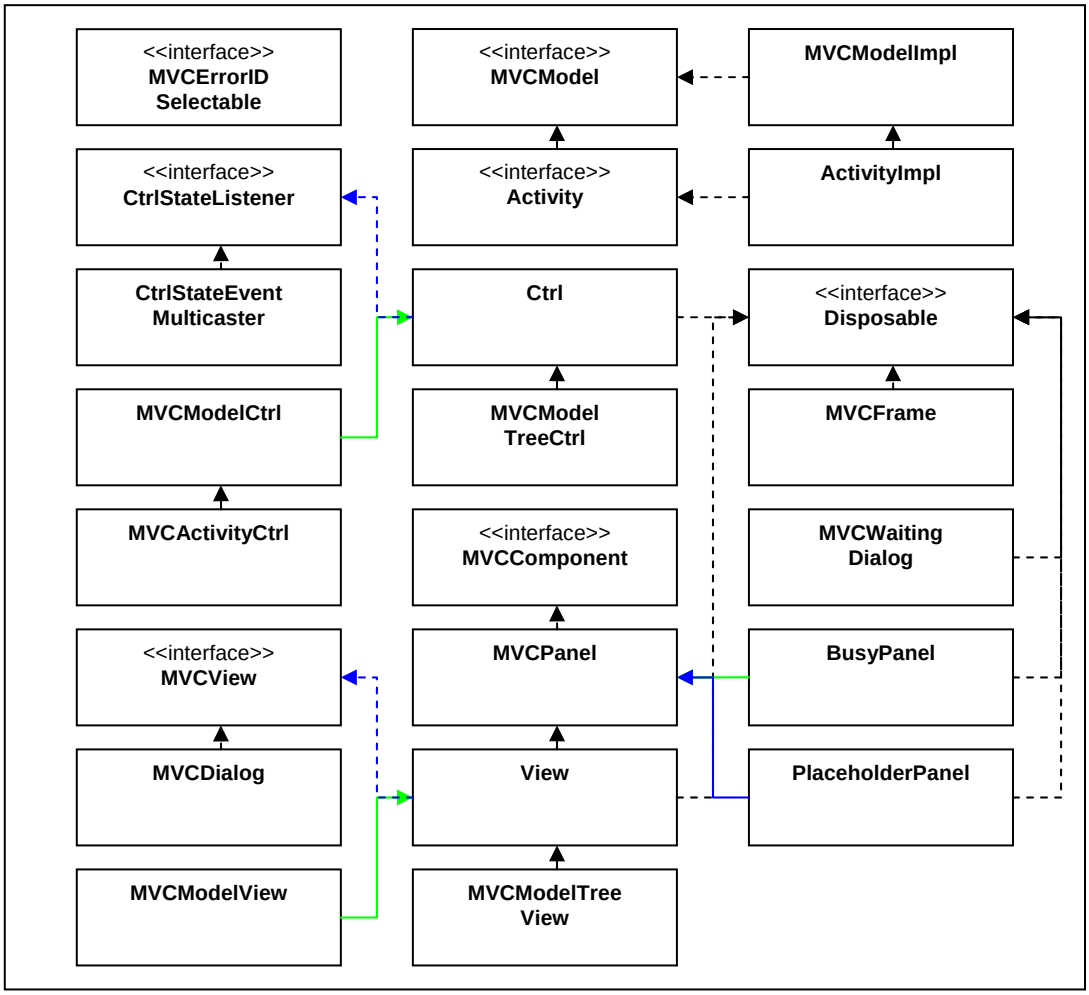
B Abbildungsverzeichnis

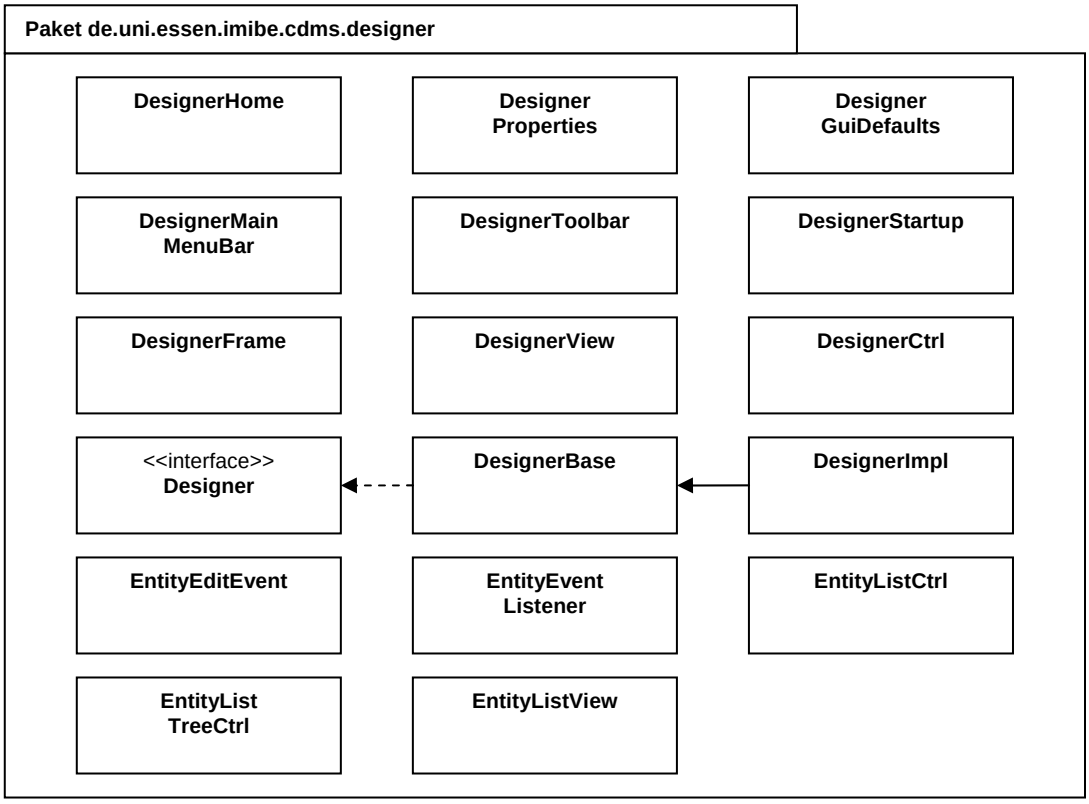
C Tabellenverzeichnis

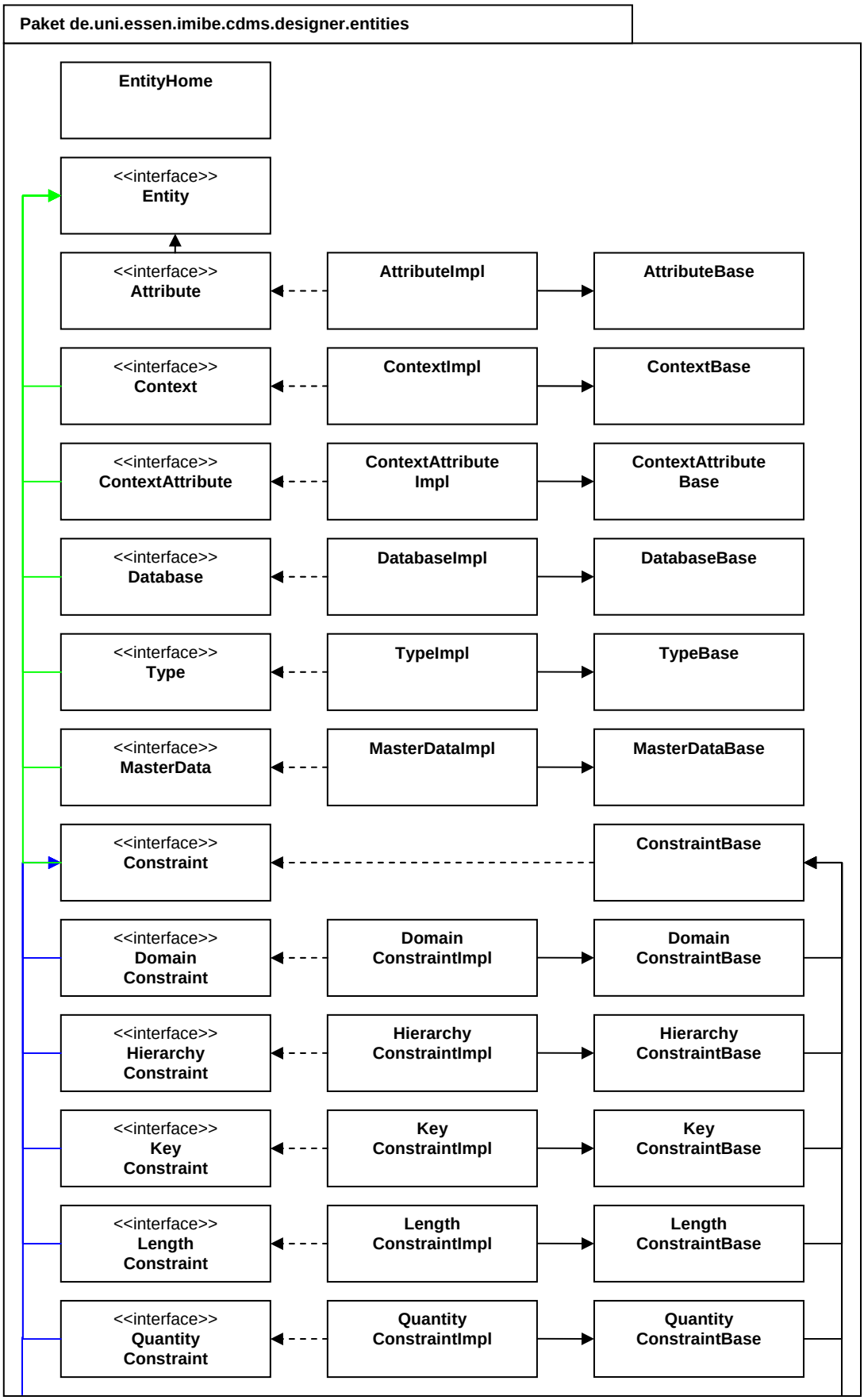
D Implementierte Klassen und Interfaces

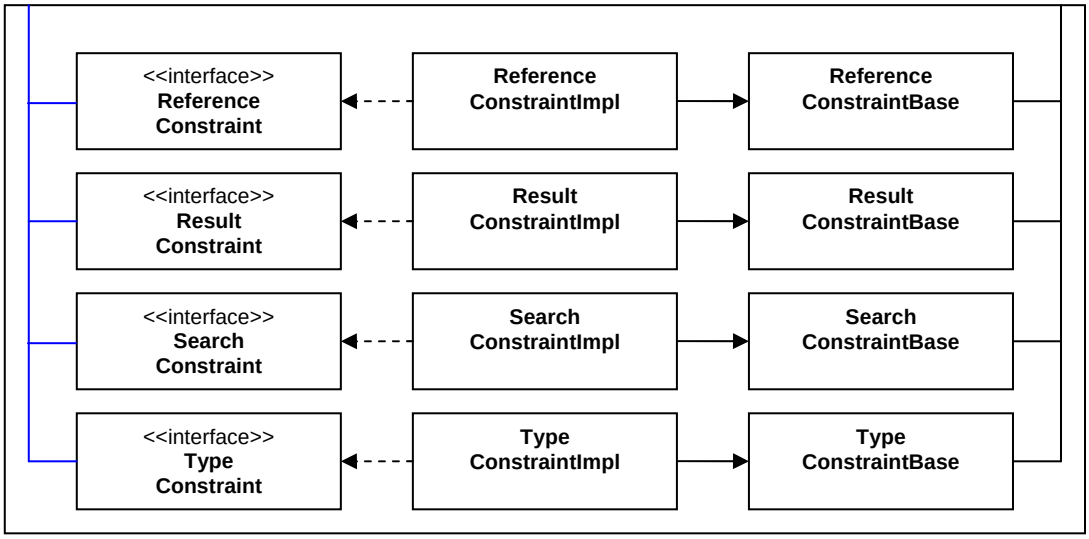
Designer

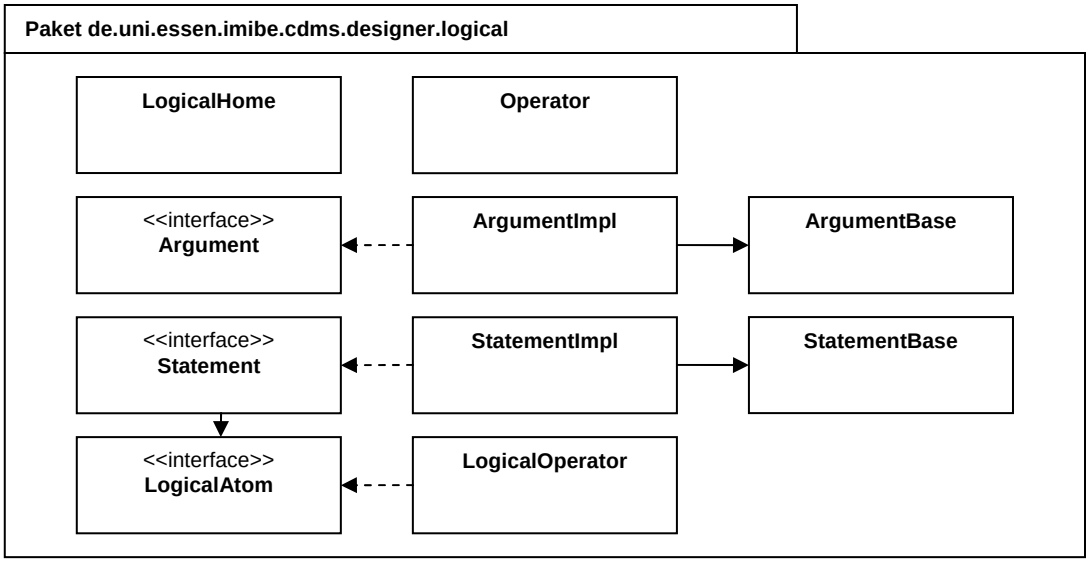


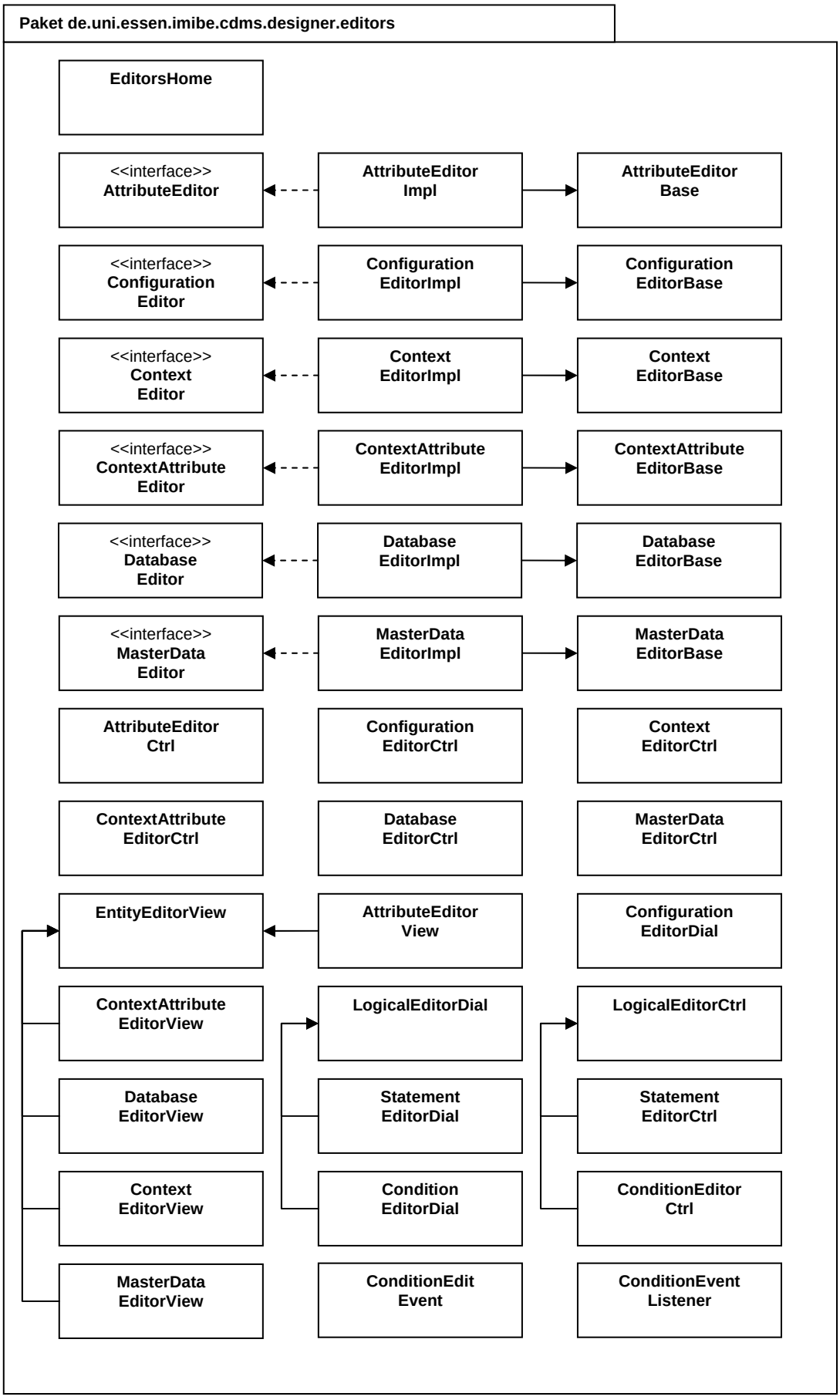

<<interface>>
MVCErrorID
Selectable
<<interface>>
MVCModel
MVCModelImpl
<<interface>>
CtrlStateListener
<<interface>>
Activity
ActivityImpl
CtrlStateEvent
Multicaster
Ctrl
<<interface>>
Disposable
MVCModelCtrl
MVCModel
TreeCtrl
MVCFrame
MVCActivityCtrl
<<interface>>
MVCComponent
MVCWaiting
Dialog
<<interface>>
MVCView
MVCPanel
BusyPanel
MVCDialog
View
PlaceholderPanel
MVCModelView
MVCModelTree
View

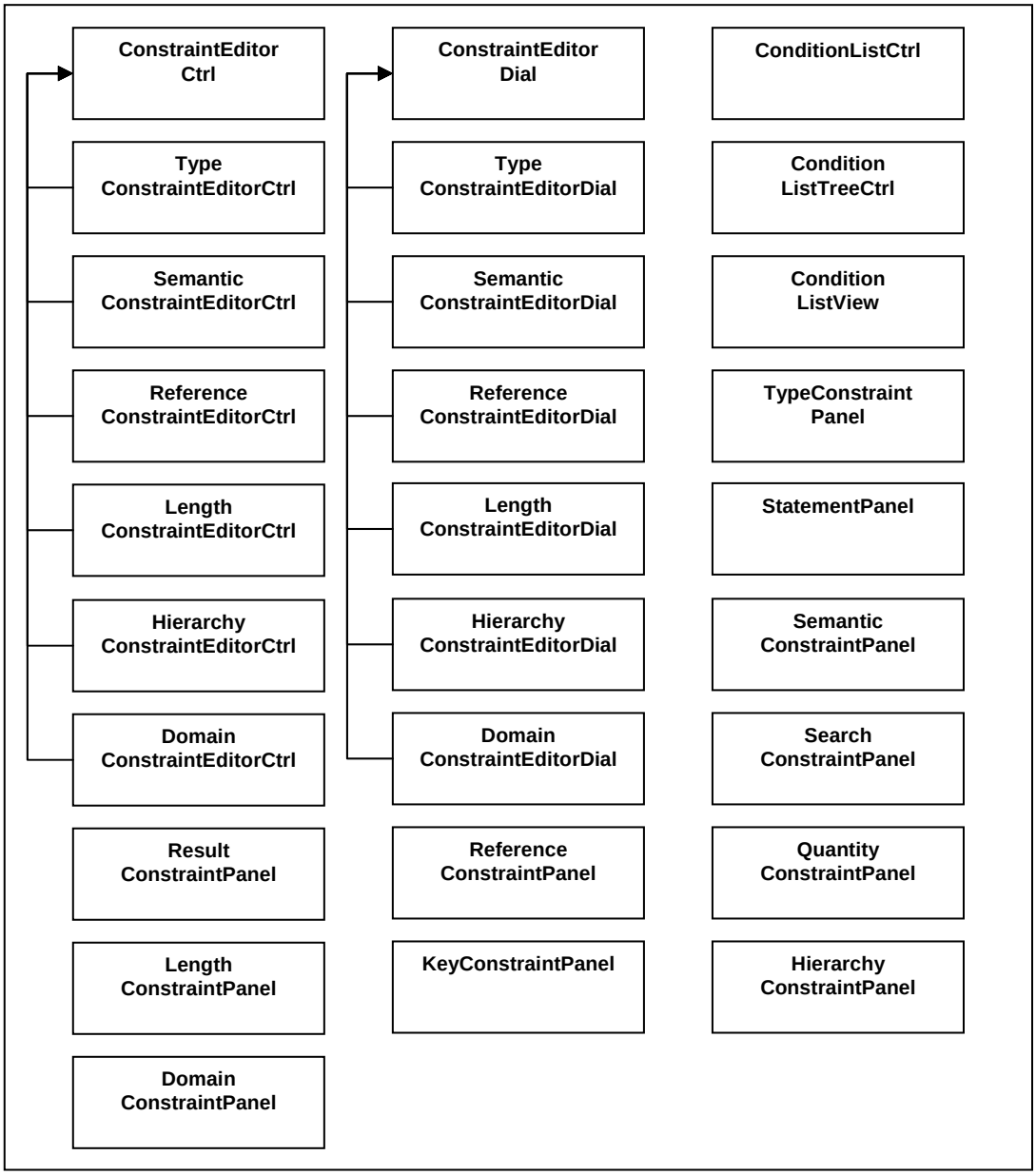

Paket de.uni.essen.imibe.cdms.designer
DesignerHome
Designer
Properties
Designer
GuiDefaults
DesignerMain
MenuBar
DesignerToolbar
DesignerStartup
DesignerFrame
DesignerView
DesignerCtrl
<<interface>>
Designer
DesignerBase
DesignerImpl
EntityEditEvent
EntityEvent
Listener
EntityListCtrl
EntityList
TreeCtrl
EntityListView


Paket de.uni.essen.imibe.cdms.designer.entities
EntityHome
<<interface>>
Entity
<<interface>>
Attribute
AttributeImpl
AttributeBase
<<interface>>
Context
ContextImpl
ContextBase
<<interface>>
ContextAttribute
ContextAttribute
Impl
ContextAttribute
Base
<<interface>>
Database
DatabaseImpl
DatabaseBase
<<interface>>
Type
TypeImpl
TypeBase
<<interface>>
MasterData
MasterDataImpl
MasterDataBase
<<interface>>
Constraint
ConstraintBase
<<interface>>
Domain
Constraint
Domain
ConstraintImpl
Domain
ConstraintBase
<<interface>>
Hierarchy
Constraint
Hierarchy
ConstraintImpl
Hierarchy
ConstraintBase
<<interface>>
Key
Constraint
Key
ConstraintImpl
Key
ConstraintBase
<<interface>>
Length
Constraint
Length
ConstraintImpl
Length
ConstraintBase
<<interface>>
Quantity
Constraint
Quantity
ConstraintImpl
Quantity
ConstraintBase

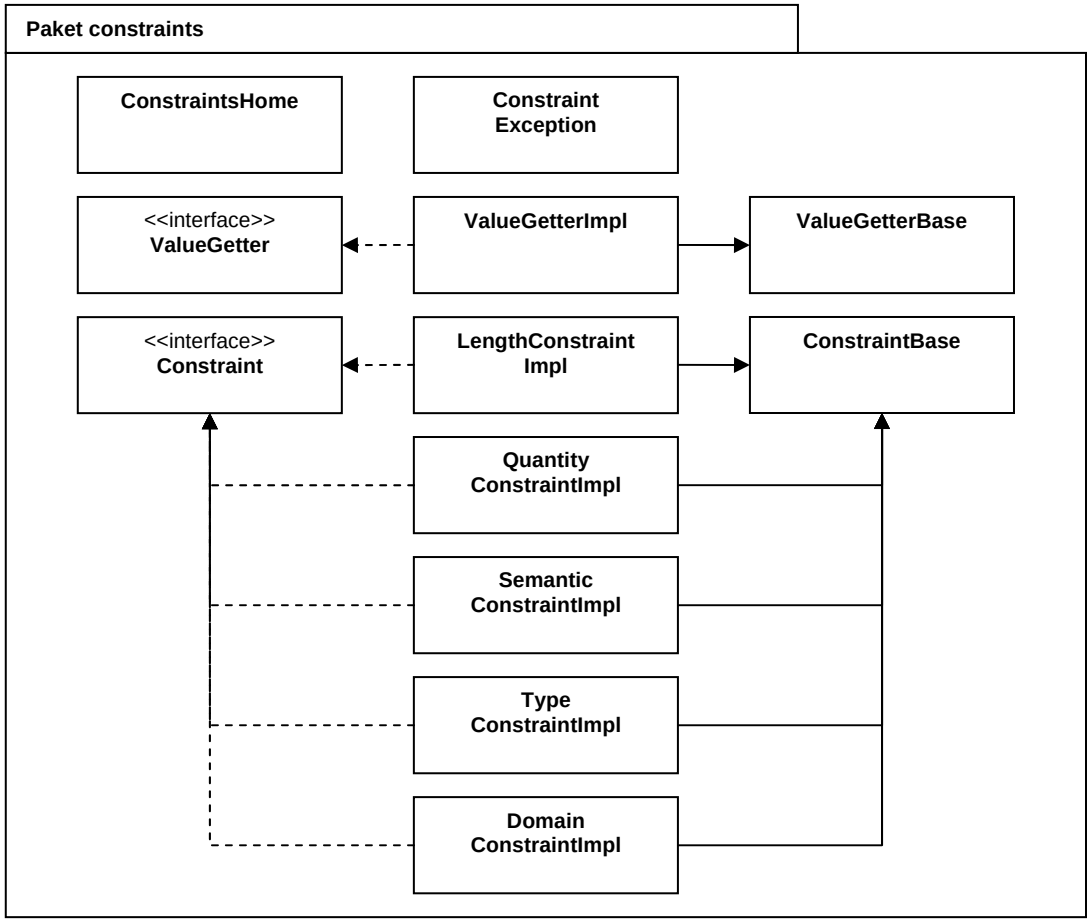

<<interface>>
Reference
Constraint
Reference
ConstraintImpl
Reference
ConstraintBase
<<interface>>
Result
Constraint
Result
ConstraintImpl
Result
ConstraintBase
<<interface>>
Search
Constraint
Search
ConstraintImpl
Search
ConstraintBase
<<interface>>
Type
Constraint
Type
ConstraintImpl
Type
ConstraintBase

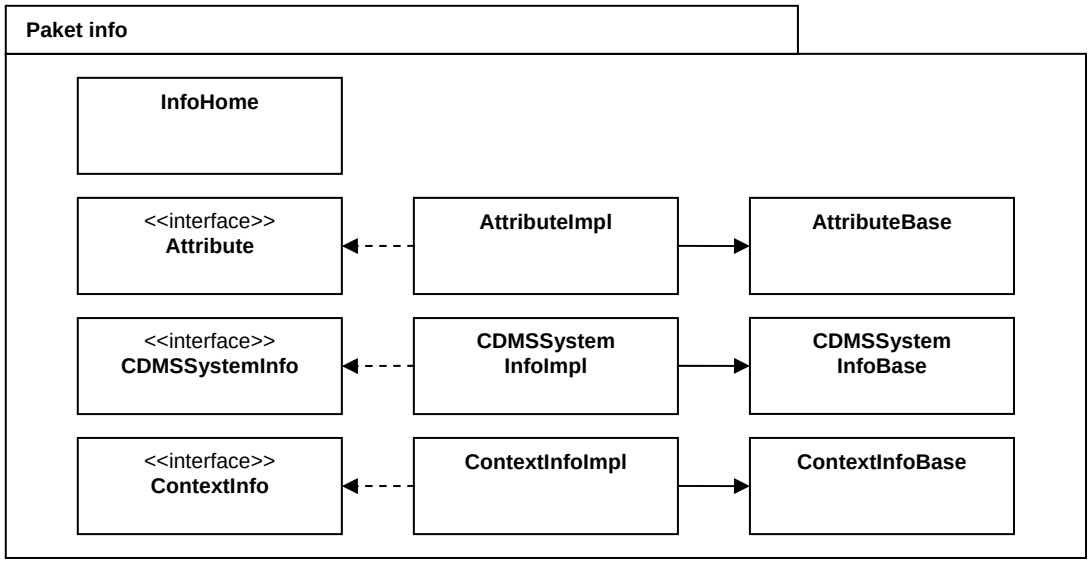

Paket de.uni.essen.imibe.cdms.designer.logical
LogicalHome
Operator
<<interface>>
Argument
ArgumentImpl
ArgumentBase
<<interface>>
Statement
StatementImpl
StatementBase
<<interface>>
LogicalAtom
LogicalOperator

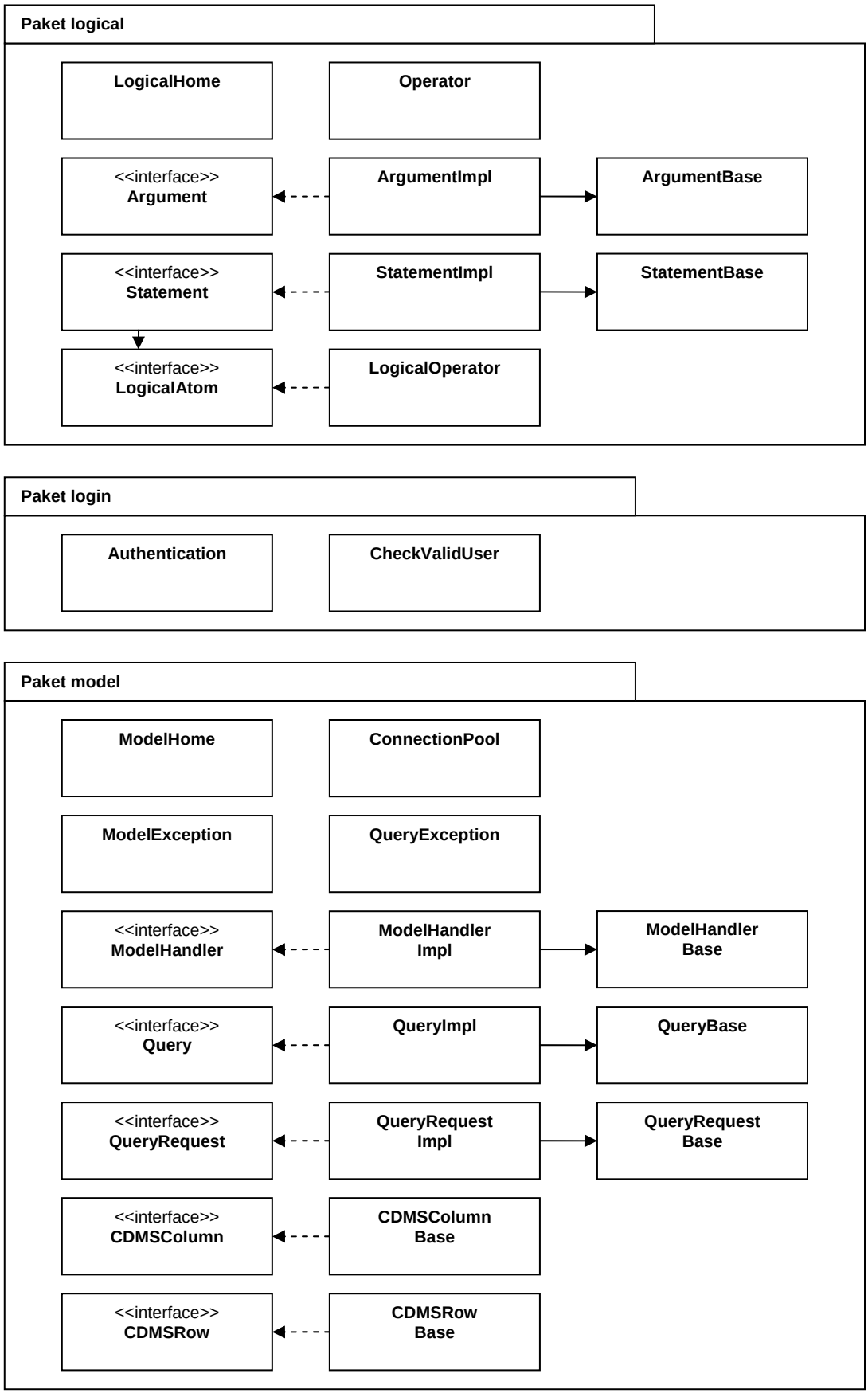

Paket de.uni.essen.imibe.cdms.designer.editors
EditorsHome
<<interface>>
AttributeEditor
AttributeEditor
Impl
AttributeEditor
Base
<<interface>>
Configuration
Editor
Configuration
EditorImpl
Configuration
EditorBase
<<interface>>
Context
Editor
Context
EditorImpl
Context
EditorBase
<<interface>>
ContextAttribute
Editor
ContextAttribute
EditorImpl
ContextAttribute
EditorBase
<<interface>>
Database
Editor
Database
EditorImpl
Database
EditorBase
<<interface>>
MasterData
Editor
MasterData
EditorImpl
MasterData
EditorBase
AttributeEditor
Ctrl
Configuration
EditorCtrl
Context
EditorCtrl
ContextAttribute
EditorCtrl
Database
EditorCtrl
MasterData
EditorCtrl
EntityEditorView
AttributeEditor
View
Configuration
EditorDial
ContextAttribute
EditorView
LogicalEditorDial
LogicalEditorCtrl
Database
EditorView
Statement
EditorDial
Statement
EditorCtrl
Context
EditorView
Condition
EditorDial
ConditionEditor
Ctrl
MasterData
EditorView
ConditionEdit
Event
ConditionEvent
Listener

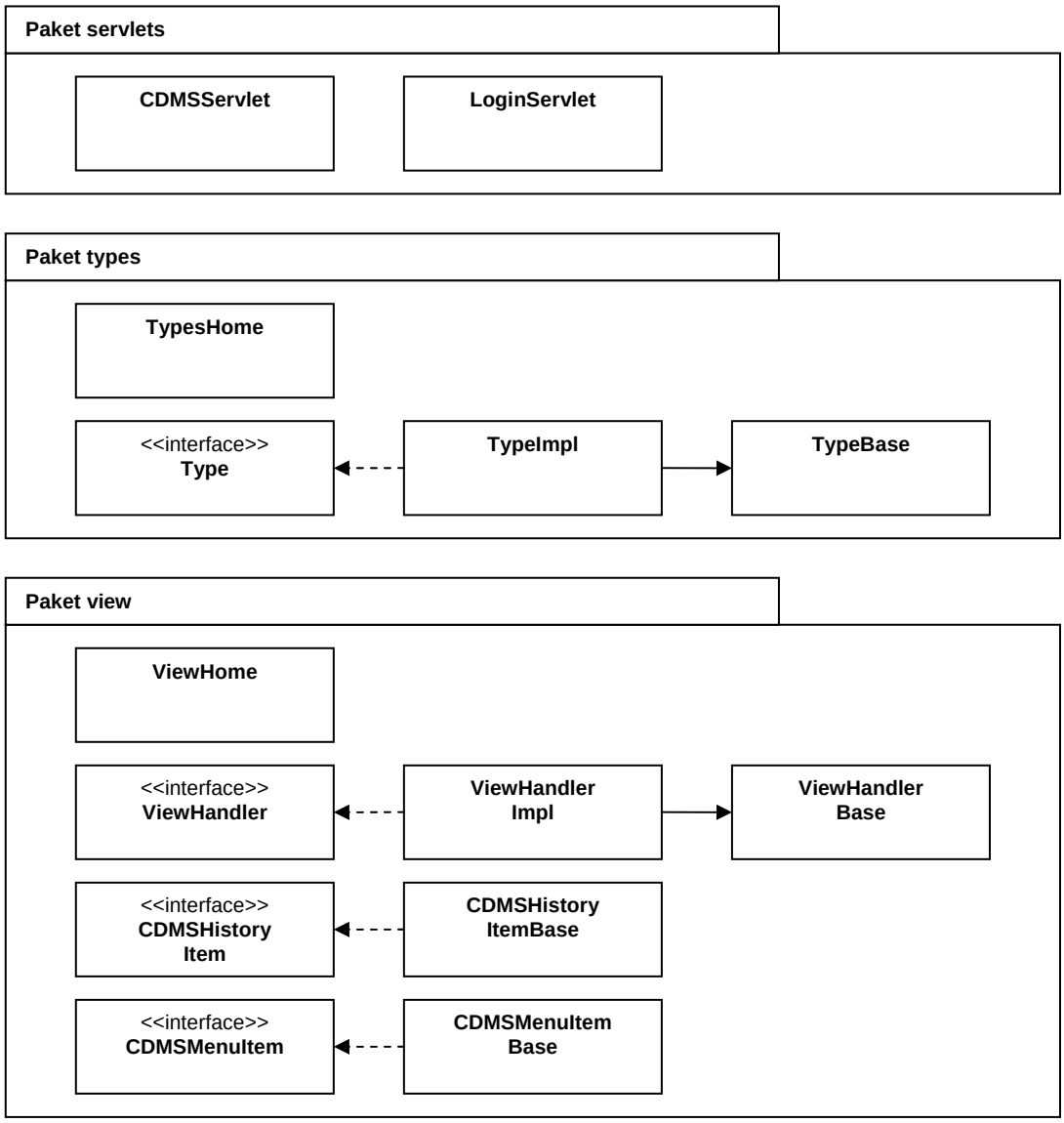

ConstraintEditor Ctrl
Type ConstraintEditorCtrl
Semantic ConstraintEditorCtrl
Reference ConstraintEditorCtrl
Length ConstraintEditorCtrl
Hierarchy ConstraintEditorCtrl
Domain ConstraintEditorCtrl
Result ConstraintPanel
Length ConstraintPanel
Domain ConstraintPanel
ConstraintEditor Dial
Type ConstraintEditorDial
Semantic ConstraintEditorDial
Reference ConstraintEditorDial
Length ConstraintEditorDial
Hierarchy ConstraintEditorDial
Domain ConstraintEditorDial
Reference ConstraintPanel
KeyConstraintPanel
ConditionListCtrl
Condition ListTreeCtrl
Condition ListView
TypeConstraint Panel
StatementPanel
Semantic ConstraintPanel
Search ConstraintPanel
Quantity ConstraintPanel
Hierarchy ConstraintPanel

Framework


Paket constraints
ConstraintsHome
Constraint Exception
<<interface>> ValueGetter
ValueGetterImpl
ValueGetterBase
<<interface>> Constraint
LengthConstraint Impl
ConstraintBase
Quantity ConstraintImpl
Semantic ConstraintImpl
Type ConstraintImpl
Domain ConstraintImpl


Paket info
InfoHome
<<interface>> Attribute
AttributeImpl
AttributeBase
<<interface>> CDMSSystemInfo
CDMSSystem InfoImpl
CDMSSystem InfoBase
<<interface>> ContextInfo
ContextInfoImpl
ContextInfoBase


Paket logical
LogicalHome
Operator
<<interface>>
Argument
ArgumentImpl
ArgumentBase
<<interface>>
Statement
StatementImpl
StatementBase
<<interface>>
LogicalAtom
LogicalOperator
Paket login
Authentication
CheckValidUser
Paket model
ModelHome
ConnectionPool
ModelException
QueryException
<<interface>>
ModelHandler
ModelHandler
Impl
ModelHandler
Base
<<interface>>
Query
QueryImpl
QueryBase
<<interface>>
QueryRequest
QueryRequest
Impl
QueryRequest
Base
<<interface>>
CDMSColumn
CDMSColumn
Base
<<interface>>
CDMSRow
CDMSRow
Base


Paket servlets
CDMSServlet
LoginServlet
Paket types
TypesHome
<<interface>>
Type
TypeImpl
TypeBase
Paket view
ViewHome
<<interface>>
ViewHandler
ViewHandler
Impl
ViewHandler
Base
<<interface>>
CDMSHistory
Item
CDMSHistory
ItemBase
<<interface>>
CDMSMenuItem
CDMSMenuItem
Base

E Komplette Modellbeschreibungs-Grammatik

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="application" type="applicationType"/>
<xsd:complexType name="applicationType">
      <xsd:sequence>
            <xsd:element ref="masterdata" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="attributes" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="database" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
</xsd:complexType>

<xsd:element name="masterdata" >
      <xsd:complexType>
            <xsd:sequence>
                        <xsd:element name="table"
                        minOccurs="1"
                        maxOccurs="unbounded"
                        type="tableType"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
<xsd:complexType name="tableType">
      <xsd:sequence>
            <xsd:element ref="row" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="required"/>
      <xsd:attribute name="label" type="xsd:string" use="required" />
      <xsd:attribute name="name" type="xsd:NMTOKEN" use="required" />
      <xsd:attribute name="description" type="xsd:string"
                        use="optional"/>
</xsd:complexType>
<xsd:element name="row">
      <xsd:complexType>
            <xsd:attribute name="id" type="xsd:NMTOKEN" use="required" />
            <xsd:attribute name="value" type="xsd:string"
                              use="required" />
      </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="attributes">
      <xsd:complexType>
            <xsd:sequence>
                        <xsd:element name="attribute"
                        minOccurs="1"
                        maxOccurs="unbounded"
                        type="attributeType"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:element>
<xsd:complexType name="attributeType">
      <xsd:sequence>
            <xsd:element ref="constraint" minOccurs="1"
                              maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="required"/>
      <xsd:attribute name="label" type="xsd:string" use="required" />
      <xsd:attribute name="description" type="xsd:string"
                        use="optional"/>
</xsd:complexType>

<xsd:element name="database" type="databaseType" />
<xsd:complexType name="databaseType">
      <xsd:sequence>
            <xsd:element ref="constraint" maxOccurs="1"/>
            <xsd:element ref="context"  minOccurs="0"
                              maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="required"/>
      <xsd:attribute name="label" type="xsd:string" use="required" />
      <xsd:attribute name="name" type="xsd:NMTOKEN" use="required" />
      <xsd:attribute name="description" type="xsd:string"
                        use="optional"/>
</xsd:complexType>

<xsd:element name="context" type="contextType" />
<xsd:complexType name="contextType">
      <xsd:sequence>
            <xsd:element ref="constraint"  minOccurs="0"
                              maxOccurs="unbounded"/>
```

```
            <xsd:element ref="contextattribute"  minOccurs="0"
                              maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="required"/>
      <xsd:attribute name="label" type="xsd:string" use="required" />
      <xsd:attribute name="name" type="xsd:NMTOKEN" use="required" />
      <xsd:attribute name="description" type="xsd:string"
                        use="optional"/>
</xsd:complexType>

<xsd:element name="contextattribute" type="contextattributeType" />
<xsd:complexType name="contextattributeType">
      <xsd:sequence>
            <xsd:element ref="constraint" minOccurs="1"
                              maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID" use="required"/>
      <xsd:attribute name="label" type="xsd:string" use="required" />
      <xsd:attribute name="name" type="xsd:NMTOKEN" use="required" />
      <xsd:attribute name="description" type="xsd:string"
                        use="optional"/>
      <xsd:attribute name="reference" type="xsd:IDREF" use="required" />
</xsd:complexType>

<xsd:element name="constraint" type="constraintType" />
<xsd:complexType name="constraintType">
      <xsd:sequence>
            <xsd:element ref="child" minOccurs="0"
                              maxOccurs="unbounded" />
            <xsd:element ref="foreign-key" minOccurs="0" maxOccurs="1" />
            <xsd:element ref="and" minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
      <xsd:attribute name="type" type="constraintTypeType"
                        use="required"/>
      <xsd:attribute name="handle" type="constraintHandleType"
                        default="error" />
      <xsd:attribute name="obligatory" type="constraintBoolType"
                        use="optional" />
      <xsd:attribute name="active" type="constraintBoolType"
                        default="true"/>
      <xsd:attribute name="message" type="xsd:string" use="optional"/>
```

```
      <xsd:attribute name="datatype" type="datatypeType" use="optional"/>
      <xsd:attribute name="min" type="xsd:nonNegativeInteger"
                        use="optional"/>
      <xsd:attribute name="max" type="xsd:nonNegativeInteger"
                        use="optional"/>
      <xsd:attribute name="dec" type="xsd:nonNegativeInteger"
                        use="optional"/>
      <xsd:attribute name="from" type="xsd:integer" use="optional"/>
      <xsd:attribute name="to" type="xsd:integer" use="optional"/>
      <xsd:attribute name="primaryKey" type="constraintBoolType"
                        default="false"/>
      <xsd:attribute name="searchColumn" type="constraintBoolType"
                        default="false"/>
      <xsd:attribute name="resultColumn" type="constraintBoolType"
                        default="false"/>
      <xsd:attribute name="operator" type="constraintOperatorType" />
      <xsd:attribute name="argumenttype" type="constraintArgType" />
      <xsd:attribute name="argument" type="xsd:string" />
</xsd:complexType>
<xsd:simpleType name="constraintTypeType">
      <xsd:restriction base="xsd:string">
            <xsd:enumeration value="type"/>
            <xsd:enumeration value="length"/>
            <xsd:enumeration value="domain"/>
            <xsd:enumeration value="key"/>
            <xsd:enumeration value="quantity"/>
            <xsd:enumeration value="reference"/>
            <xsd:enumeration value="search"/>
            <xsd:enumeration value="result"/>
            <xsd:enumeration value="hierarchy"/>
            <xsd:enumeration value="semantic"/>
      </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="constraintHandleType">
      <xsd:restriction base="xsd:string">
            <xsd:enumeration value="error"/>
            <xsd:enumeration value="warning"/>
      </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="constraintBoolType">
      <xsd:restriction base="xsd:string">
```

```
            <xsd:enumeration value="true"/>
            <xsd:enumeration value="false"/>
      </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="datatypeType">
      <xsd:restriction base="xsd:string">
            <xsd:enumeration value="character"/>
            <xsd:enumeration value="number"/>
            <xsd:enumeration value="integer"/>
            <xsd:enumeration value="real"/>
            <xsd:enumeration value="date"/>
            <xsd:enumeration value="time"/>
            <xsd:enumeration value="timestamp"/>
            <xsd:enumeration value="reference"/>
      </xsd:restriction>
</xsd:simpleType>
<xsd:element name="foreign-key" type="foreignKeyType" />
<xsd:complexType name="foreignKeyType">
      <xsd:sequence>
            <xsd:element name="reference" minOccurs="1" maxOccurs="1">
                  <xsd:complexType>
                        <xsd:attribute name="local" type="xsd:IDREF"
                              use="required" />
                        <xsd:attribute name="foreign" type="xsd:string"
                              use="required" />
                  </xsd:complexType>
            </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="foreignTable" type="xsd:IDREF"
                        use="required"/>
</xsd:complexType>
<xsd:element name="child" type="childType" />
<xsd:complexType name="childType">
      <xsd:attribute name="context" type="xsd:IDREF" use="required" />
</xsd:complexType>

<xsd:element name="and" type="andType" />
<xsd:complexType name="andType">
      <xsd:sequence>
            <xsd:element ref="statement" minOccurs="0"
                              maxOccurs="unbounded" />
```

```
            <xsd:element ref="and" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element ref="or" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:element name="or" type="orType" />
<xsd:complexType name="orType">
      <xsd:sequence>
            <xsd:element ref="statement" minOccurs="0"
                              maxOccurs="unbounded" />
            <xsd:element ref="and" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element ref="or" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
</xsd:complexType>
<xsd:element name="statement" type="statementType" />
<xsd:complexType name="statementType">
      <xsd:attribute name="argument1" type="xsd:string" use="required" />
      <xsd:attribute name="operator" type="constraintOperatorType"
                        use="required" />
      <xsd:attribute name="argumenttype" type="constraintArgType"
                        use="required" />
      <xsd:attribute name="argument2" type="xsd:string" use="required" />
</xsd:complexType>
<xsd:simpleType name="constraintOperatorType">
      <xsd:restriction base="xsd:string">
            <xsd:enumeration value="EQ"/>
            <xsd:enumeration value="NE"/>
            <xsd:enumeration value="GT"/>
            <xsd:enumeration value="LT"/>
            <xsd:enumeration value="GE"/>
            <xsd:enumeration value="LE"/>
      </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="constraintArgType">
      <xsd:restriction base="xsd:string">
            <xsd:enumeration value="value"/>
            <xsd:enumeration value="null"/>
            <xsd:enumeration value="reference"/>
      </xsd:restriction>
</xsd:simpleType>
</xsd:schema>
```

F Modellierungsablauf *InCoMe*

1. Entwurf des Modells im Entwurfsmodul
    a. Erfassung der freien Merkmalsarten
    b. Erfassung der benötigten Stammdaten
    c. Bündelung der Kontexte
    d. Definition der Integritätsbedingungen
2. Export des Modells
    a. Export des Datenbankskriptes
    b. Export der Anwendungskonfiguration (`web.xml`)
    c. Export der Anwendung (`appl.xml`)
3. Erzeugen der Datenbank
    a. Anlage eines Schemas im verwendeten Datenbank-Managementsystem
    b. Import des Datenbankskriptes in die Datenbank
    c. Anlegen einer Tabelle USER_ADMIN mit jeweils 10-stellig alphanummerischen Feldern USERNAME und PW für die Authentifikation; Eintragung der gewünschten Benutzer
4. Konfiguration der Anwendung
    a. Eintragung des Datenbanktreibers und der –URL in die Anwendungskonfiguration
    b. Speicherung der Anwendungskonfiguration in das Verzeichnis WEB-INF der Webapplikation
    c. Kopie der Anwendung in das Verzeichnis WEB-INF/classes/info der Webapplikation
5. Starten des Applikationsservers

G Inhalt der CD

Verzeichnis	Erklärung
Creator	Generierungsmodul
- ApplCreator	Generierung Applikation
- servlets	Skript zur Applikationsgenerierung
- DBCreator	Generierung Datenbank
- mysql	Skript zur Datenbeschreibungsgenerierung MySql
- oracle	Skript zur Datenbeschreibungsgenerierung Oracle
Designer	Entwurfsmodul
- classes	Bytecode Designer
- doc	Dokumentation Designer (javadoc)
- src	Quellcode Designer
Framework	Applikationsframework
- doc	Dokumentation Framework (javadoc)
- webapps	Framework
Hivnet	Beispiel-Merkmalskatalog
IDE	Entwicklungsressourcen

Lebenslauf

Personalien

Vorname:	Ralf
Name:	Goertzen
Geburtsdatum:	19.08.1968
Geburts-/Wohnort:	Oberhausen
Familienstand:	ledig

Schulischer Werdegang

08.1975 - 07.1979	Hirschkamp-Grundschule, Oberhausen
08.1979 - 05.1988	Sophie-Scholl-Gymnasium, Oberhausen Abschluss: Allgemeine Hochschulreife

Universitärer Werdegang

10.1994 - 07.1998	Studium der Wirtschaftsinformatik an der UGH Essen Abschluss: Diplom-Wirtschaftsinformatiker (DI)
10.1998 - 12.2002	Studium der Wirtschaftsinformatik an der UGH Essen Abschluss: Diplom-Wirtschaftsinformatiker (DII)
04.2004 - 10.2005	Promotionsstudium an der Medizinischen Fakultät der Universität Duisburg-Essen

Beruflicher Werdegang

09.1988 - 12.1990	Ausbildung zum Datenverarbeitungskaufmann bei der Energieversorgung Oberhausen (EVO)
12.1990 - 09.1994	Organisationsprogrammierer und Projektleiter ebd.
10.1994 - 11.1998	Selbstständiger Berater und Programmierer
10.1995 - 11.1998	Studentische Hilfskraft im Fachgebiet Wirtschaftsinformatik und Softwaretechnik der UGH Essen
04.1998 - 07.2002	Sänger bei Starlight Express, Bochum
12.1998 - 09.1999	Systemplaner bei der Mannesmann Datenverarb. GmbH
10.1999 - 02.2000	Informationsmanager bei der Plexus Media Komm. GmbH
03.2000 -	Freiberuflicher IT-Berater
07.2002 - 12.2002	Wissenschaftliche Hilfskraft am Lehrstuhl Spezifikation von Softwaresystemen der Universität Duisburg-Essen
04.2003 - 09.2003	Wissenschaftlicher Mitarbeiter ebd.

Sonstiges

09.1992	Ausbildereignungsprüfung vor der IHK zu Essen
03.2003 - 12.2003	Ausbildung und Zertifizierung zum Führungskräfte-Coach an der NWA Osnabrück