HL7-Implementation in C++ — ein automatisierter Ansatz

Beitrag zum HL7-Workshop, Göttingen, den 26. Oktober 1996
Gunther Schadow,
Dr. U. Föhring, Prof. Dr. Th. Tolxdorff
Universitätsklinikum Benjamin Franklin, Berlin

In der Klinik für Anaesthesiologie und Operative Intensivmedizin der FU-Belin wurde eine HL7-Implementation (ProtoGen/HL7[1]) erstellt. Mit einem Generator wird der C++-Code der Implementation automatisch aus der Standarddefinition abgeleitet.

Einführung

HL7 zu implementieren heißt, eine EDV-Anwendung aus dem medizinischen Bereich zu befähigen, standardisiert nach HL7 zu kommunizieren. Nun ist eine Anwendung ein Programm, welches in einer bestimmten Programmiersprache verfaßt ist, weshalb die HL7-Implementation die Konzepte von HL7 dem Anwendungsprogrammierer auf der Ebene dieser Programmiersprache zur Verfügung zu stellen hat.

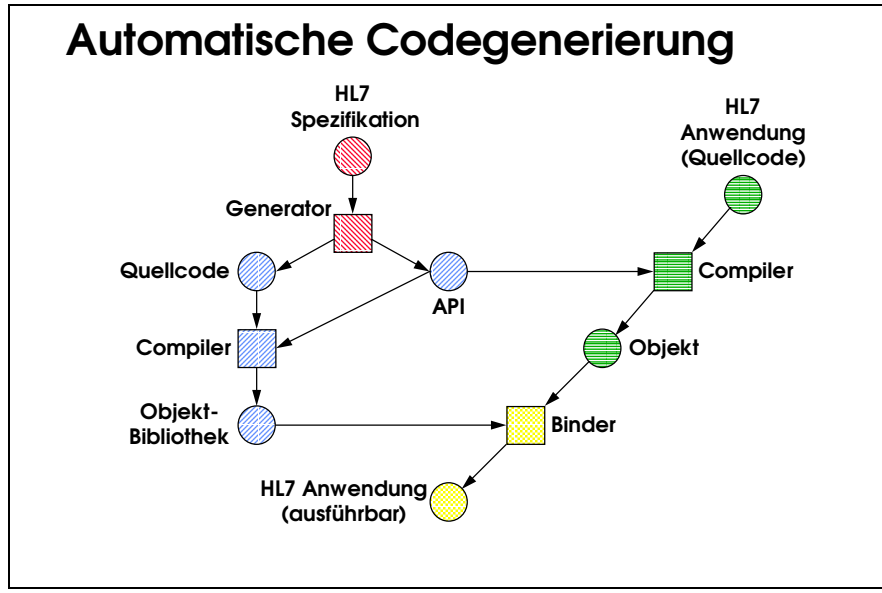

Abbildung 1: Prozeß der Erstellung von HL7-Implementation und HL7-Anwendung

In Abbildung 1 ist dieser Prozeß bildlich dargestellt: Die HL7-Implementation soll als Quellcode einer Compilersprache vorliegen. Durch compilieren wird sie in eine Compilerobjektbibliothek[2] überführt. Das Anwendungsprogramm liegt ebenfalls als Quellcode vor und wird unabhängig von der HL7-Bibliothek compiliert. Die Kopplung zwischen Quellcode der Anwendung und der HL7-Implementation geschieht über das Application programming interface (API). Durch die Bindung des Objektcodes der Anwendung mit dem der HL7-Bibliothek entsteht die ausführbare HL7-Anwendung.

Obwohl hier davon ausgegangen wird, daß das Anwendungsprogramm und die HL7-Implementation in einer (und derselben) Compilersprache verfaßt sind, gilt dieses Verfahren sinngemäß auch für andere Situationen: Liegt die Anwendung in einer Interpretersprache vor, so wird

[1] ProtoGen/HL7 ist freie Software und unterliegt dem GNU Copyright (GPL). Bezug über folgende Adressen: http://fub46.zedat.fu-berlin.de:8080/~gusw, ftp://dumccss.mcc.duke.edu:/standards/HL7/implementations/ProtoGen

[2] Das Wort „Objekt" interferiert hier mit dem Objektbegriff der objektorientierten Methode, mit dem es nichts zu tun hat. Ein Compilerobjekt ist das Produkt eines Compilers, eine Datei, welche u.a. Maschinenprogrammcode enthält. Es kann mit anderen Compilerobjekten zu einer Bibliothek oder zu einem Ausführbaren Programm verbunden werden.

die HL7-Implementation entweder auch in der Interpretersprache verfaßt, wobei der Compilerschritt entfällt, oder sie wird in den Interpreter selbst integriert. Letzteres bietet sich an, um etwa Tcl[3] oder Perl[4] mit HL7-Funktionen auszurüsten.

Mitunter wird ein einzelnes System, welches eine HL7-Schnittstelle besitzt, als eine „HL7-Implementation" bezeichnet. Hier wird so ein System „HL7-Anwendung" genannt, im Gegensatz zu einer von mehreren Systemen wiederverwendbaren HL7-Implementation als Bibliothek. Auch ein Communication Server (oder „Interface engine"), welcher, neben anderen Protokollen, auch über HL7 kommuniziert, ist in diesem Sinne keine HL7-Implementation sondern eine besondere HL7-Anwendung.

Automatische Methode

Weil der Umfang des HL7-Standards sehr groß ist, ist es erforderlich, die Implementation so zu erstellen, daß möglichst wenig manueller Aufwand betrieben wird, um möglichst wenig Fehlerquellen zu haben und so möglichst genau dem HL7-Standard zu entsprechen. Dies kann nur mit automatischen Methoden erreicht werden, wobei API und Quellcode der Implementation von einem Programmgenerator aus der autoritativen HL7-Standarddefinition selbst nach festen Regeln abgeleitet werden. Dieses Verfahren gewährleistet zudem eine hohe Flexibilität, mit der institutsoder landesweite Ergänzungen (Z-Segmente, GSG-Daten) oder Versionswechsel des Standards selbst schnell umgesetzt werden können.

Die Erkenntnis der Notwendigkeit von Automatisierung auf dieser Ebene ist nicht neu: So besitzt die RPC[5]-Implementation von Sun[6] einen Generator namens „rpcgen", der C-Code aus einer Definitionsdatei produziert. Für die Abstract syntax notation One (ASN.1)[7], in welcher die höheren OSI-Protokolle[8] definiert sind, gibt es eine Reihe von Compilern, welche Programme in den verschiedensten Sprachen erzeugen.

Außer der hier ausgeführten compilierten Methode kann die Protokolldefinition *zur Laufzeit* der HL7-Anwendung *interpretiert* werden. Dabei wird das Senden, Empfangen und alle anderen Zugriffe auf die Nachrichten durch die Protokolldefinition — etwa in Form von Tabellen — gesteuert. Der Interpreter selbst ist relativ kompakt und gegenüber Änderungen der Tabellen invariant. Bei der compilierten Methode muß die Implementation bei jeder Änderung neu generiert und compiliert werden. Andererseits kann eine compilierte Implementation effizienter arbeiten, da der Zugriff auf die Nachrichten ohne ständige Konsultation der Steuertabellen erfolgt. Dagegen könnte man meinen, die interpretierende Methode sei flexibler, da Änderungen in den Tabellen schnell vorgenommen seien. Diese Flexibilität — wenn sie denn nicht nur eine scheinbare sein soll — hat jedoch einen hohen zusätzlichen Preis: Jede Änderung in den Steu-

```
segment(pid, "patient identification", adt).

field(pid, 1, "set id - patient id",       si, '', '', '', 04).
field(pid, 2, "patient id (external id)", ck, '', '', '', 16).
field(pid, 4, "alternate patient id",      st, '', '', '', 12).
field(pid, 3, "patient id (internal id)", ck, '', r,  y,  20).
field(pid, 5, "patient name",              pn, '', r,  '', 48).
...
```

Abbildung 2: Definition des PID-Segmentes und seiner Felder in Prolog

[3] Ousterhout J.K.; Tcl: an embeddable command language.; in Proc. of USENIX Winter Conference 1990, pp 133–146
[4] Wall, Lary; Programming Perl (Practical Extraction and Report Language); O´Reilly and Associates, 1991
[5] RFC 1050: Remote Procedure Call Protocol Specification; Sun Microsystems Inc., USC-ISI
[6] Sun Microsystems Inc.
[7] ISO/IEC 8824:1990: Information Technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)
[8] ISO 7498:1984: Information processing systems – Open Systems Interconnection – Basic Reference Model

ertabellen betrifft unmittelbar auch alle Anwedungen der Implementation, die u.U. Annahmen machen, die mit dem geänderten Zustand der Tabellen nicht mehr übereinstimmen. Dies birgt ein hohes Risiko für evtl. sehr schwer zu entdeckende Fehler.

Voraussetzung für ein automatisches Verfahren — ob compiliert oder interpretierend — ist eine formale Spezifikation des Standards. Der Text des HL7-Standards (Versionen 2.1 und 2.2) definiert die HL7-Nachrichten in der „Abstract message syntax" Notation, einer einfachen formalen Sprache. Aber auch die Tabellen, in welcher die Segmente und Codes präsentiert werden, sind formale Spezifikationselemente. Andere wesentliche Gegenstände des HL7-Standards sind dagegen nicht formalisiert worden. Die primitiven Datentypen lassen sich überhaupt nur schwer formalisieren, da ihre Unterschiede und Spezialitäten zu zahlreich sind. Aus demselben Grunde müssen sie auch manuell implementiert werden. Verbunddatentypen, lassen sich jedoch einfach als Vektoren von primitiven Datentypen in Tabellenform darstellen.

```
ADT message (event A02):
        MSH
        PID
        PV1
        [PV2]
        [{OBX}]

message(adt, a02, "adt messge",
        [msh,evn,pid,pv1,opt(pv2),opt(rep(obx))]).
```

Abbildung 3: Die Nachricht ADT^A02 in der „Abstract syntax definition" und der daraus gebildeten Form in Prolog

Aber auch die formalen Elemente des HL7-Standards müssen zunächst aus dem Text extrahiert werden. Daß dieser mit einem Verbreiteten WYSIWYG-Textprozessor[9] hergestellt wurde, erweist sich bei diesem Vorhaben als sehr nachteilig: Die formalen Elemente sind nämlich nur optisch erkennbar, nicht aber ohne weiteres maschinenlesbar. Nur mit Einschränkungen und erheblichem Aufwand war es möglich die Elemente zu extrahieren, und dieser Aufwand droht sich mit jeder neuen Edition zu wiederholen. Ein Ausweg wäre die Verwendung von SGML[10] (HTML[11]) oder TEX[12]. Dies würde nicht nur das automatische „lesen" des Textes erleichtern. Auch die Erstellung könnte automatisch aus entsprechenden Datenbanken erfolgen. Diese wären auf Konsistenz geprüft, was schließlich nicht nur der konsistenten Darstellung, sondern auch der Richtigkeit des Inhalts förderlich wäre. Für die Implementation der HL7 Version 2.2 konnte zum Großteil auf die von Herrn Oemig[13] und dem technischen Komitee der deutschen HL7-Benutzergruppe zusammengestellte Datenbank zurückgegriffen werden. Allein die Nachrichtendefinitionen, die dort bisher nicht berücksichtigt sind, wurden noch mit einem AWK[14]-Programm aus dem Text des Standards extrahiert.

Es ist naheliegend, die Gegenstände von HL7, welche in Form von Tabellen definiert wurden, nach dem relationalen Modell weiterzuverarbeiten. Da aber mit den Nachrichtendefinitionen ein wesentlicher Teil von HL7 aus nicht-relationalen Strukturen — darstellbar durch Bäume oder andere Graphen — besteht, ist die relationale Datenbank zur Erfassung der Information über HL7 und deren Weiterverarbeitung nicht ausreichend. Statt dessen wurde hierfür die Programmiersprache Prolog gewählt, von welcher sowohl Relationen, als auch Bäume unterstützt werden. Eine Segmentdefinition wird damit wie in Abbildung 2 ausgedrückt.Es handelt sich hierbei um ein Prädikat namens **segment**, welches für die gegebenen Argumente (**pid** usw.) als zutreffend festgestellt wird, oder anders ausgedrückt: **pid** oder **"patient identification"** ist ein Segment des **adt**-Bereiches. Die dazu gehörenden Felder werden dann in analoger Weise angege-

[9] „What You See Is What You Get", (z.B. Microsoft Word für Windows).
[10] ISO 8879: Standard Generalized Markup Language
[11] RFC 1866: Hypertext Markup Language - 2.0
[12] Knuth, Donald E.: The TEXbook, 1986, Addison-Wesley
[13] Oemig, Frank: Informationsstruktur von HL7, Aufbau einer Access-Datenbank; Beitrag zum HL7-Workshop, Göttingen, den 26. Oktober 1996
[14] Aho, Kernighan, Weinberger: The AWK Programming Language; 1988; Addison-Wesley

ben. Man erkennt leicht die Form der Relation wieder: jeder Satz gibt ein 7-Tupel („Zeile“) der Relation („Tabelle“) **`field`** an.

Die Nachrichtendefinitionen in Prolog sind eine einfache lexikalische Umformung der „Abstract message syntax“-Schreibweise des HL7-Standards (Abbildung 3). Auch die Nachrichten werden also in der Form des Prädikats definiert, welche der Form der Relation äquivalent ist. Der Unterschied ist nur, daß das vierte Feld eine Baumstruktur zum Inhalt hat. Dabei wird eine Folge von Segmenten als Liste abgebilet, und die Klammerausdrücke **`[...]`** und **`{...}`** in die Funktorenschreibweise **`opt(...)`** bzw. **`rep(...)`** überführt, deren Argumentliste wiederum Segmente und Funktoren enthält.

Ist die Information über HL7 in dieser Weise erfaßt, so kann sie vielfältig weiterverarbeitet werden. Zunächst muß die Information auf Konsistenz geprüft werden, wobei Fehler der Erfassung und auch fehlerhafte Definitionen des Standards selbst zu tage treten. Diese Inkonsistenzen müssen manuell korrigiert werden, dann erst kann die Information zur Erzeugung von Programmcode verwendet werden. Aber auch andere Schreibweisen derselben Information können gebildet werden, um etwa HTML-Dokumente zu erzeugen, welche die HL7-Struktur in gut lesbarer Weise darstellen. So wird die Dokumentation der Implementation immer auf dem Aktuellen Stand gehalten.

```
konsistenz_1 :- forall(field(S, _,_,_,_,_,_), segment(S, _,_)).
konsistenz_2 :- forall(segment(S, _,_), field(S, F, _,_,_,_,_)).
```

Abbildung 4: Konsistenzprüfung: Jedes Feld muß zu einem Segment gehören (1) und jedes Segment muß Felder enthalten (2).

Das Verfahren der Programmgenerierung darzustellen würde den gegebenen Rahmen sprengen, doch sei hier am Beispiel einer Konsistenzprüfung gezeigt, wie intuitiv die HL7-Information in Prolog verarbeitet werden kann (Abbildung 4). Die erste Konsistenzbedingung besagt, daß alle Segmentbezeichner, für welche Felder definiert sind, auch als Segment bekannt sein müssen, und die zweite, daß umgekehrt jedes Segment auch Felder beinhalten soll. Entsprechend wird dann auch untersucht, ob jedes in den Nachrichten verwendete Segment auch tatsächlich definiert ist.

Konzepte von HL7 und ihre Implementation

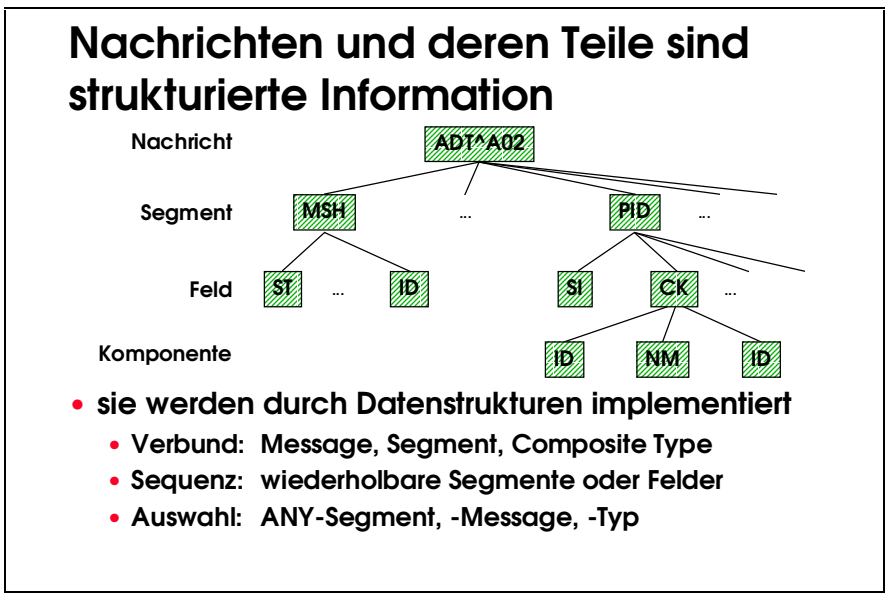

Abbildung 5: Ausschnitt aus dem Strukturbaum der Nachricht ADT^A02

Der wesentliche Bestandteil des HL7-Standards sind die Nachrichten. Nachrichten sind strukturierte Informationen, d.h. mehrere Informationselemente werden in bestimmter weise angeordnet gemeinsam übermittelt. Abbildung 5 zeigt einen vereinfachenden Ausschnitt aus dem Strukturbaum der ADT^A02-Nachricht. Es werden die vier bekannten Ebenen dargestellt: Eine Nachricht besteht aus bestimmten Segmenten, diese wiederum aus Feldern, welche evtl. mehrere Komponenten enthalten können. Letzteres ist als rekursiver Vorgang intendiert.

Es ist naheliegend, die struktu-

rierte Information in Datenstrukturen der Zielprogrammiersprache abzubilden. Dazu können Verbünde (Pascal: **Record**, C: **struct**) verwendet werden. Sequenzen, welche wiederholbare Felder oder Segmente fassen, werden durch verkettete Listen dargestellt, da in der Regel die Anzahl der Wiederholungen unbestimmt ist. Schließlich benötigt man auch Varianzen, (Pascal: **Record** mit variablem Teil, C: **union**), da vor allem im Kapitel 4 (Order entry), mehrere syntaktisch verschiedene Nachrichten mit dem selben Bezeichner und für das selbe Ereignis (Event) definiert werden. Es ist hier keine Lösung, den Nachrichten unterschiedliche Bezeichner zu vergeben, denn dies würde den Standard an empfindlicher Stelle verletzen. Daher müssen die Abweichungen in der Syntax herausgearbeitet und in variablen Datenstrukturen abgebildet werden.

Die Datenstrukturen ermöglichen es jede in HL7 erlaubte Nachricht im Anwendungsprogramm darzustellen. Zum Zweck der Übermittlung zwischen verschiedenen Systemen muß diese Nachricht in eine externe Repräsentation umgewandelt werden. Dies wird auch als „Serialisierung" der Datenstrukturen, oder nach ISO/OSI als „Präsentation" bezeichnet. Hierfür werden zwei Funktionen benötigt: der *Builder* und der *Parser*. Der *Builder* erstellt den serialisierten Zeichenstrom aus der Struktur, und der *Parser* ist die dazu inverse Funktion. Dies ist in Abbildung 6 dargestellt.

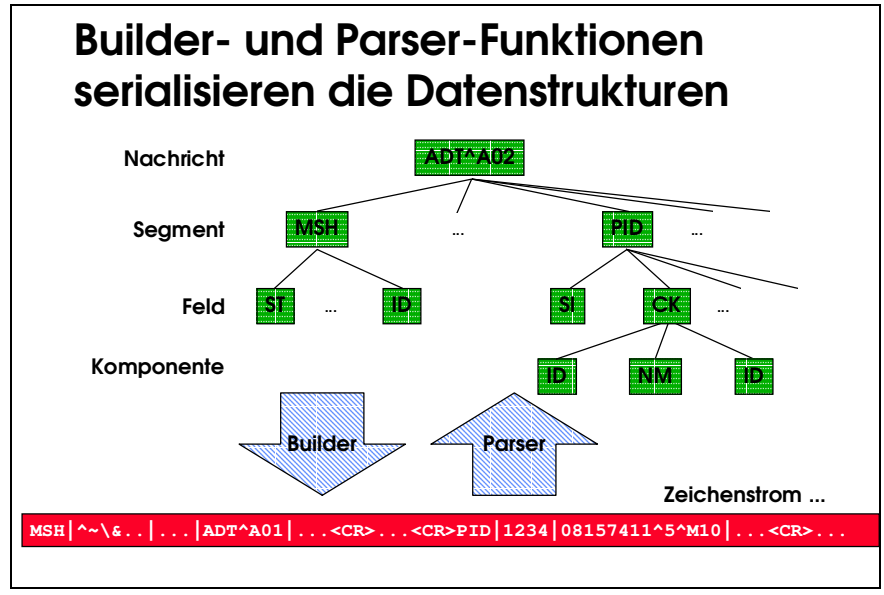

Abbildung 6: Der Strukturbaum der Nachricht ADT^A02 wird zu einem Zeichenstrom „serialisiert".

Zur Serialisierung können verschiedene Kodierregeln angewandt werden. HL7-Nachrichten werden meist in den HL7-Kodierregeln ausgetauscht, die denen von ASTM[15] und EDIFACT[16] ähnlich sind. Sie sind textorientiert und haben den Vorteil, daß die Nachrichten — wenn auch eingeschränkt — lesbar sind. Sie verwenden Trennzeichen, wodurch bei den HL7-Kodierregeln das Problem auftritt, daß Verbunddatentypen nur bis zu zweimal geschachtelt werden können, weil darüber hinaus keine Trennzeichen mehr definiert sind. HL7 Version 3.0 wird aller Voraussicht nach eine Erweiterung der klassischen HL7-Kodierregeln aufnehmen. Diese werden den Namen „ER7"[17] tragen. Ein Fortschritt gegenüber den herkömmlichen Regeln ist, daß mit zwei neuen klammernden Trennzeichen v.a. die echte Rekursion möglich wird.

Es ist durch den HL7-Standard ausdrücklich erlaubt auch andere Kodierregeln zu verwenden. Alternativ könnten daher zum Beispiel die „eXternal Data Representation" (XDR)[18] wie von RPC benutzt werden. Das Übertragungsformat ist nicht mehr textorientiert sondern binär und dadurch sehr effizient. Datenstrukturen können unbegrenzt geschachtelt werden. Ferner könnte

[15] ASTM E1394: A Standard Specification for Transferring Information Between Clinical Instruments and Computer Systems

[16] ISO 9735:1988: Electronic data interchange for administration commerce and transport (EDIFACT) – Application level syntax rules

[17] Ein m.E. etwas unglücklich gewählter Name. Es ist wohl schon vergessen worden, daß die „7" in „HL7" einst auf die 7. Schicht des OSI-Modells hinwies. Kodierregeln, und damit auch „ER7", gehören nach OSI aber in die 6. Schicht!

[18] RFC 1014: External Data Representation Standard; Sun Microsystems Inc., USC-ISI

man auf die von der ISO definierten „Basic Encoding Rules“ (BER)[19] zurückgreifen, welche im Zusammenhang mit ASN.1 benutzt werden. Diese arbeiten ebenfalls binär, sind aber viel weniger effizient als XDR. Sowohl XDR als auch BER sind im HL7-Bereich jedoch bisher nicht gebräuchlich. Daher wurden in ProtoGen/HL7 zunächst nur die HL7-Kodierregeln implementiert. Vorkehrungen für die künftige Unterstützung anderer Kodierregeln wurden aber getroffen.

Als Zielsprache dieser HL7-Implementation wurde C++ gewählt. Diese Wahl hat verschiedene Vorteile, welche alle in den einen münden, Konzepte statt Details zu behandeln. Die meisten Gegenstände von HL7 werden dabei als *Klasse* implementiert. Eine Klasse ist eine Zusammenfassung aus einer Datenstruktur, und den für sie sinnvoll verwendbaren Funktionen. So hat z.B. jede Nachricht, jedes Segment und jeder Typ die *Parser*- und *Builder*-Funktion (**`input()`** bzw. **`output()`**). Der Objektbegriff bezeichnet konkrete Instanzen einer Klasse. Klassen können voneinander abgeleitet sein, so ist z.B. jede HL7-Klasse von einer Grundklasse abgeleitet, welche die „Anlage“ zur *Builder*- und *Parser*-Funktion enthält und diese auf die abgeleiteten Klassen „vererbt“. Sehr nützlich ist, daß Funktionsnamen mehrfach vergeben werden, und daher auch Operatoren neu definiert werden können. Dies alles erlaubt eine sehr intuitive Anwendungsprogrammierung, wodurch sich vor allem auch Programmierfehler vermeiden lassen. Dennoch kann in C++ durch seine Kompatibilität zu C sehr effizient auf Betriebssystemresourcen zugegriffen werden, was für eine Kommunikationsanwendung unerläßlich ist.

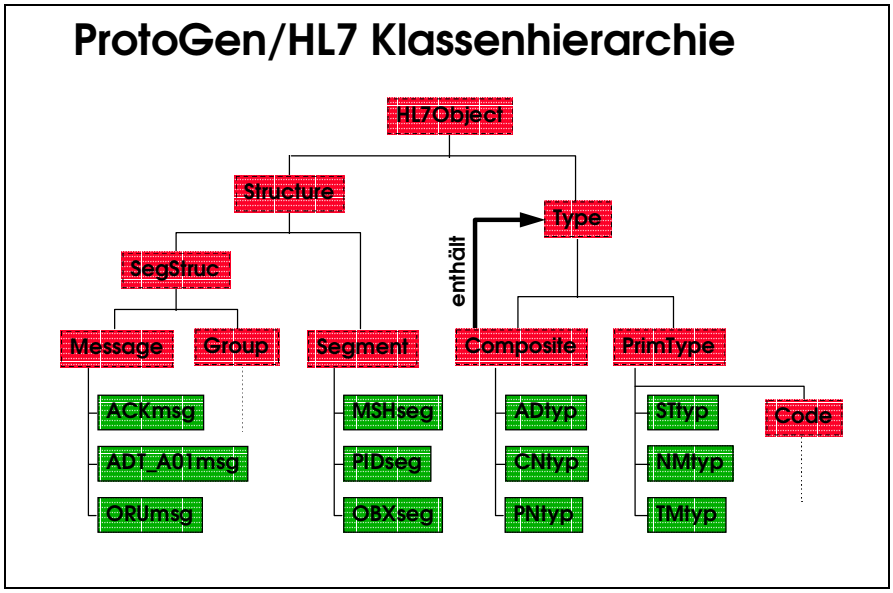

Abbildung 7: Die Gegenstände von HL7 nach Gattungen und Arten. Die Rekursivität der Beinhaltung von Typen im Verbund (Composite) ist durch den dicken Pfeil ausgedrückt.

Mit den Konzepten der Objektorientierung wurde ein Informationsmodell von HL7 entworfen (Abbildung 7). Die Baumstruktur zeigt die Beziehung der einzelnen Gegenstände von HL7 nach ihrem Wesen, d.h. dem Art-Gattungs-Verhältnis. Im Gegensatz also, zur Struktur einer einzelnen Nachricht, wie sie in Abbildung 5 vorgestellt wurde, wird hier die Struktur des ganzen Standards aufgeführt. Man findet wieder die Klassen **`Message`**, **`Segment`**, **`Composite`** und **`PrimType`**, diesmal jedoch sind sie Horizontal auf derselben Ebene angeordnet. Alle HL7-Klassen stammen von der abstrakten Klasse **`HL7Object`** ab, welche z.B. die allgemeine Definition der *Parser*- und *Builder*-Funktion enthält.

Alle nicht-terminalen Knoten des Klassen-Baumes sind abstrakte Klassen, d.h. diese können keine unmittelbaren Instanzen haben. Die Blätter des Klassen-Baumes bilden die Konkreten Klassen. So kann eine Variable vom Typ **`ACKmsg`** definiert werden, was ein ACK-Nachrichtenobjekt erzeugt.

Die Klassen **`Type`** und **`Structure`** unterteilen die HL7-Klassen in das was der HL7-Standard „Data element“ nennt und Strukturen aus solchen Datenelementen. Der Unterschied ist vor al-

[19] ISO 8825:1990: Information technology – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

lem, daß der NULL-Wert für Typen definiert ist, nicht aber für Strukturen. Die NULL ist von dem bloßen Fehlen eines Wertes zu unterscheiden. Der Unterschied wird deutlich wenn aufgrund einer HL7-Nachricht ein Datenbank-Update vorgenommen werden soll. Ein fehlender Wert bedeutet, daß keine Aussage über diesen Wert gemacht wird und ein in der Datenbank evtl. vorbestehender Wert weiterhin gültig bleibt. Dagegen besagt der NULL-Wert, daß der vorbestehende Wert nicht mehr gültig, ein bestimmter neuer Wert aber auch nicht einzusetzen ist.

Datentypen

Typen können sowohl primitive Datentypen (**PrimType**) als auch Verbunddatentypen (**Composite**) sein. Verbunddatentypen enthalten nun wiederum Typen. Diese sind in der Regel Primitive, können aber auch ihrerseits wieder Verbundtypen sein. Diese Schachtelung ist rekursiv und die Tiefe ist prinzipiell nicht begrenzt — allein die HL7-Kodierregeln geben hier zur Mäßigung Anlaß. Eine rekursive Schachtelung von Datentypen ist nicht ohne weiteres in ein relationales Modell abbildbar. In C++ (und den meisten anderen Programmiersprachen) ist Rekursion dagegen ein natürliches Ausdrucksmittel. Es obliegt der Kopplung zwischen HL7 und einer Datenbank (also einer HL7-Anwendung) diese Schachtelung aufzulösen.

Daß aber in HL7 der anonyme Verbunddatentyp CM verwendet wird, ist nun sehr wohl ein Problem. Es wurde bereits darauf hingewiesen, daß eine abstrakte Klasse nicht unmittelbar ein Datentyp sein kann. CM ist aber eine abstrakte Klasse und wird im HL7-Standard dennoch vielfach als Datentyp gebraucht. Dabei wird bestenfalls im Text kurz erwähnt aus welchen Komponenten er an einer bestimmten Stelle bestehen sollte. Dieses Problem wurde in ProtoGen/HL7 auf zweierlei Art gelöst: Erstens wurde eine konkrete **Composite**-Klasse namens **CMtyp** definiert, welche eine feste Anzahl Komponenten vom ST-Typ enthält. Dadurch wird gewährleistet, daß die per HL7 übermittelte Information zunächst wenigstens nicht verloren geht. Will man jedoch sinnvoll in einer Anwendung mit einem solchen Verbundtyp umgehen, so ist es ratsam, einen speziellen Verbundtyp (wie es etwa PN oder AD ist) in der formalen Spezifikation zu definieren und die Standardspezifikation an den Stellen zu ändern, wo dieser neue Typ gemeint ist.

Beispielsweise verwendet HL7 immer wieder einen CM-Typ, welcher für einen Patienten-Aufenthaltsort („Patient location“) steht. Dieser enthält die Komponenten: nurse-unit^room^bed. Hier wurde daher der Typ ZL definiert (Abbildung 8). Damit kann der neue ZL-Typ statt des CM-Typs in die entsprechenden Felddefinitionen (v.a. im PV1-Segment) eingesetzt werden. Obwohl in Version 2.2 angekündigt wird, daß künftig von dem unspezifizierten CM-Typ abgegangen werden soll, wurde die Zahl der Verwendungen des CM-Typs seit der Version 2.1 mehr als verdoppelt (von ca. 20 auf über 50). Hinzu kommt, daß sich bei genauerer Betrachtung hinter manchem ST- oder ID-Typ ebenfalls ein CM-Typ verbirgt. Die eigentlich vom HL7-Standard zu leistende Arbeit kann durch den hier gegebene doppelten Lösungsansatz schrittweise bei Bedarf ausgeführt werden.

```
composite(zl,[
       [id, "nurse unit"],
       [id, "room"],
       [id, "bed"]]).
```

Abbildung 8: Die Definition eines neuen Verbunddatentyps „ZL“, der für den Aufenthaltsort des Patienten in der Klinik steht („patient location“).

Es scheint der CM-Typ vielfach als eine Art „String“ aufgefaßt zu werden, weshalb man davon ausgeht, daß man an jeden Typ grundsätzlich neue Komponenten „anhängen“ könne. Dies führte dazu, daß in Version 2.2 der vormals primitive Datentyp TS (Time stamp) solcherart erweitert wurde. Damit ist er aber nun ein Verbundtyp geworden, was nicht unerhebliche Probleme mit sich bringt. Wollte man TS nun als Verbundtyp definieren, so bräuchte man immer noch einen primitiven Typ der dem TS-Typ aus Version 2.1 identisch ist. Da „TS“ nun aber einen neuen Verbundtyp bezeichnet, muß für den alten ein neuer Name gefunden werden. Da der HL7-Standard sich des Problems offenbar nicht bewußt ist, wird von der Seite auch kein Standardname

definiert. Hier wurde entschieden, das Wesen des TS-Typs zu erhalten, und die Behandlung der zusätzlichen Komponente intern zu Regeln. Es ist jedoch zu hoffen, daß der HL7-Standard künftig nicht so grundlegend verändert wird.

In welche Dimensionen die Komplexität von Verbunddatentypen gesteigert werden kann zeigt der neue TQ-Typ (Timing/quantity). Nicht genug, daß dieser in der ersten Ebene zehn, und in der zweiten Ebene noch einmal fünfzehn Komponenten beinhaltet. Da dies dem verantwortlichen Komitee noch nicht reichte, die HL7-Kodierregeln jedoch weitere Ebenen verbieten, wurden kurzerhand strukturierte Codes definiert. Beispielsweise bedeutet nun **`Q30S`**, daß ein Ereignis alle 30 Sekunden stattfinden soll, und mit **`Q50M`** alle 50 Minuten. Diese strukturierten Codes sprengen den Rahmen, den sich HL7 für seine Struktur einmal gesetzt hatte. Denn sie konstituieren Substrukturen, die von dem übrigen HL7-Standard losgelöst sind. Praktisch heißt das, daß die für diese Codes nicht unkomplizierte *Parser*- und *Builder*-Funktionen nun von Hand geschrieben werden müssen.

Ein solches Vorgehen wäre mit den bestehenden Ausdruckselementen von HL7 eigentlich ganz unnötig gewesen, da man die gleiche Information ebensogut in Feldern und Komponenten unterbringen kann. Grundsätzlich sind strukturierte Codes allerdings nicht zu vermeiden. Sie sind dann sinnvoll, wenn Sie außerhalb von HL7 benutzt werden und standardisiert sind. ICD-10 und ICPM sind Beispiele für die sinnvolle Verwendung strukturierter Codes.

Aus der Sicht von HL7 ist ein Code ein ID-Typ, für den die möglichen Werte definiert sind, meist in Form einer Tabelle aber auch in Form von Regeln, wie bei strukturierten Codes. Andererseits wird in vielen Fällen auch der ID-Typ verwendet, ohne daß eine Wertetabelle angegeben wird. Daher wurde für die Implementation wieder eine doppelte Lösung gefunden: Vorkommen des ID-Typs ohne weitere Definition werden als Zeichenketten behandelt. Wenn aber eine Wertetabelle definiert wurde, so sind nur noch die darin enthaltenen Werte erlaubt.

Kleine Tabellen und solche, deren Umfang relativ unveränderlich ist, wie etwa der das Geschlecht bezeichnende Code, werden dabei direkt in die Bibliothek compiliert. Große Tabellen und solche, die häufigeren Änderungen unterworfen sind, können extern in Datenbanken gehalten werden. Dabei wurde aus Gründen der Effizienz und Portierbarkeit nicht auf kommerzielle Relationale Datenbanksysteme zurückgegriffen, sondern ein eigenes Modul entwickelt, welches einen schnelleren und vereinfachten Zugriff unter C++ ermöglicht.

Strukturierte Codes können schließlich durch spezielle Klassen manuell implementiert werden und so die Eigenarten des jeweiligen Codes wiedergeben. Auf diese Weise wurden bisher der LOINC-Code[20] implementiert, welcher zur Bezeichnung von Laborparametern im Result report verwendet wird. Hier werden zum Beispiel mehrere Zugriffswege auf einen Eintrag des Codes ermöglicht: Einerseits über den Schlüssel, und andererseits über die Kombination mehrerer Attribute, welche den Inhalt beschreiben. Ganz ähnlich würde man auch ICD und ICPM Codes erschließen.

```
Unit   kPa  ("kN/m2");
Unit   mmHg ("mm(Hg)");
double P_mmHg, P_kPa = 13.6;
if(dim(kPa) == dim(Pa))
  P_mmHg = convert(P_kPa, kPa, mmHg);
```

Abbildung 9: Verwendung der Klasse **`Unit`**: Einheiten-Objekte für *kPa* und *mm(Hg)* werden erstellt, auf Kommensurabilität getestet und ein Druckwert umgerechnet.

Weiter wurde der ISO Code für Maßeinheiten[21] implementiert, welcher ebenfalls im Result report Verwendung findet. An der Verbindung zwischen verschiedenen Anwendungen aus dem Laborbereich (oder jedem anderen klinischen Bereich) müssen immer wieder Meßgrößen in ihre verschiedenen Einheiten umgerechnet werden (z.B. *kPa* in *mm(Hg)*). Die Klasse **`Unit`** ermöglicht dazu nicht nur das Erkennen von Präfixen und Einheiten und das Zerlegen von ganzen Einheiten-

[20] LOINC: Laboratory Observation Identifier Names and Codes; Regenstrief Institute Indianapolis, 1995

[21] ISO 2925:1983: Information processing – Representation of SI and other units in systems with limited character sets

termen wie etwa $kg \cdot m/s^2$. Die Einheiten werden vielmehr vollständig semantisch erfaßt, so daß sie auf Kommensurabilität geprüft und ggf. ineinander umgerechnet werden können. Abbildung 9 spricht für sich selbst, und für die Eleganz der Methode.

Segmente — Gruppen — Nachrichten

Nachrichten und Segmente werden gemeinsam als Strukturen (**`Structure`** in Abbildung 7) den Datenelementen (**`Type`**) gegenübergestellt. Darunter befindet sich eine weitere Klasse, mit der im HL7-Standard zwar umgegangen wird, die aber nicht explizit definiert wird. Gemeint ist die Segment-Gruppe (**`Group`**). Gruppen sind Einheiten von Segmenten oder wieder anderen Gruppen, welche hinsichtlich ihrer Optionalität oder Widerholbarkeit zusammengehören. Dieser bloß formal-syntaktischen Feststellung liegt allerdings auch eine inhaltlich-konzeptionelle Bedeutung zu Grunde. Am Beispiel der ORU-Nachricht soll dies in Abbildung 10 verdeutlicht werden.

Daß zum Beispiel die PID- und PV1-Segmente gemeinsam optional sind, hat einen bestimmten Grund, nämlich den, daß eine Ergebnisnachricht sowohl patientenbezogene, als auch patientenunabhängige Informationen enthalten kann. Man kann in der ORU-Nachricht die Gruppen „Patient", „Auftrag" und „Ergebnis" identifizieren. Ferner gibt es die äußerste Gruppe „Satz", die fast den gesamten Nachrichtenkörper umfaßt und wiederholbar ist.

Da der HL7- Standard die Existenz der Gruppen weitgehend unerwähnt läßt, muß der Codegenerator die Gruppen durch eine Strukturanalyse der Nachrichtendefinitionen identifizieren. Dabei werden die strukturgleichen Gruppen aus allen Nachrichtenarten zusammengefaßt. Dieses Verfahren kann jedoch nur mit Einschränkungen die *konzeptionelle* Bedeutung der Gruppen rekonstruieren, und allzu oft verhindern feine syntaktische Unterschiede der Definition, daß konzeptionell identische Gruppen nicht als solche erkannt werden. Es muß daher gefordert werden, daß in künftigen Versionen von HL7 die Gruppen explizit als solche definiert werden.

Ein weiteres Problem, welches jedoch v.a. mit den HL7-Kodierregeln zusammenhängt, kann ebenfalls an der ORU-Nachricht veranschaulicht werden. Die folgende Sequenz von Segmenten beschreibt eine gültige ORU-Nachricht:

MSH, OBR, OBX, PID, PV1, OBR, OBX, OBR, OBX.

Die erste Auftragsgruppe (*OBR, OBX*) gleich nach dem Nachrichtenkopf (*MSH*) enthält offensichtlich patientenunabhängige Ergebnisse. Diese ist gefolgt von einer Patient-/Fall-Bestimmung (*PID, PV1*) welcher nun eine oder zwei Auftragsgruppen folgen. Es ist aber nicht klar, ob die letzte Ergebnisgruppe noch zum genannten Patient/Fall gehört, oder ob es sich wieder um unabhängige Ergebnisse handelt. Folgende Interpretationen sind also möglich:

MSH, (OBR, OBX), ((PID, PV1), (OBR, OBX), (OBR, OBX)).

und

MSH, (OBR, OBX), ((PID, PV1), (OBR, OBX)), (OBR, OBX).

```
ORU Observation Result Message
MSH
{                          Satz
   [                          Patient
      PID
      [{NTE}]
      [PV1]
   ]
   {                          Auftrag
      [ORC]
      OBR
      [{NTE}]
      [{                        Ergebnis
         OBX
         [{NTE}]
      }]
   }
}
[DSC]
```

Abbildung 10: Die Nachricht ORU enthält 4 Gruppen.

Es wird deutlich, daß HL7-Kommunikation unter bestimmten Umständen nicht eindeutig ist. Man kann auch Regeln finden, mit denen sich die Möglichkeit solcher Uneindeutigkeiten ausschließen läßt. In der Anwendung der derzeitigen Version (2.2 und 2.3) muß aber auf solche Fälle

im Einzelfall geachtet werden. Eine Korrekturmaßnahme im vorliegenden Fall besteht darin, die äußere Wiederholbarkeit zu verbieten.[22]

Wie einleitend gesagt wurde, sollte die HL7-Implementation dem Anwendungsprogrammierer die HL7-Konzepte auf der ebene seiner Programmiersprache verfügbar machen. Die Details sollen dabei möglichst verborgen bleiben. Das entlastet nicht nur, es vermeidet auch Fehler. Wichtig ist daher nicht allein die Funktionalität einer Bibliothek sondern auch das API. Ein API definiert aber wesentlich Bezeichner (Namen) durch deren Gebrauch im Anwendungsprogramm die Bibliotheksfunktionalität referenziert und aktiviert wird. Namen sollen daher die Bedeutung sichtbar machen und daneben praktikabel in der Handhabung, also nicht zu lang, sein. Mit ProtoGen/HL7 werden die Namen über einen einfachen Algorithmus aus den Standardbezeichnern generiert, wodurch eine sinnvolle Kürze bei erhaltener Aussagekraft erreicht wird.

```
class PIDseg: public Segment      // patient identification
{
    SItyp             SetIdPatId;  // set id - patient id
    CKtyp             PatIdExtId;  // patient id (external id)
    repfield<CKtyp>   PatIdIntId   // patient id (internal id)
    CKtyp             AltPatId;    // alternate patient id
    PNtyp             PatName;     // patient name
...
};

...

PIDseg pid;
pid.PatIdIntId[0].IdNum   = "08/15-4711";
pid.PatName.FamName       = "Musterman";
pid.PatName.GivenName     = "Hans";
```

Abbildung 11: Definition der Klasse für das PID-Segment, wie sie von ProtoGen/HL7 erzeugt wird. Die interne Patientenkennzahl ist als wiederholbares Feld definiert. Unten werden einige Felder gesetzt. Man erkennt, wie auf das wiederholbare Feld wie auf ein Array zugegriffen wird, obwohl es als verkettete Liste implementiert ist.

Abbildung 11 zeigt zuerst die Klassendeklaration in C++, welche aus der in Abbildung 2 vorgestellten Definition des PID-Segmentes generiert wurde. Daran schließt ein Beispiel seiner Verwendung an: Die Variable **pid** wird als ein PID-Segment wird deklariert und anschließend der Patientenname auf „Hans Mustermann“ gesetzt.

Sitzungs- und Transportschicht

Zu einer brauchbaren HL7-Implementation gehören natürlich Vorrichtungen, welche die tatsächliche Übermittlung der Nachrichten ermöglichen. Diese Aufgabe fällt in den Bereich der Sitzungs- und Transportschicht (4) nach dem OSI-Schichtenmodell. Von ProtoGen/HL7 wird derzeit TCP/IP[23] als Transportmedium unterstützt. Ein separater Umsetzer für die Unterstützung serieller Verbindungen mit dem im HL7-Standard benannten ANSI X3.28[24] Protokoll wurde entwickelt. Das Minimal LLP und Hybrid LLP, welche im HL7-Standard bzw. „Implementation Guide“ beschrieben werden, sind damit ebenfalls Verwendbar. Ferner können Dateien (etwa über NFS[25]), Pipes, Sockets und FIFOs benutzt werden. TCP/IP hat den Vorteil, daß es ebenso einfach wie verbreitet in der Anwendung ist.

Schließlich wurde auf der Grundlage von TCP/IP eine Abstraktion einer HL7-Sitzung (OSI-Schicht 5) implementiert, wobei eine Sitzung als ein Nachrichtenpaar — bestehend aus Anforderung (Request) und Antwort (Reply) — angesehen wird. Damit kann die Erstellung eines HL7-Servers sehr vereinfacht werden. Abbildung 12 skizziert einen einfachen ADT-Server. Die Ereig-

[22] Es besteht m.E. auch kein Nachteil darin, statt der Wiederholung des ganzen Nachrichtenkörpers einfach mehrere Nachrichten zu senden.

[23] RFC 791: Internet Protocol und RFC793: Transmission Control Protocol

[24] ANSI X3.28-1976: Procedures for the use of the Communications Control Characters of American National Standard Code for Information Interchange in Specified Data Communication Links

[25] RFC 1094: NFS: Network File System Protocol Specification; Sun Microsystems Inc.

nisse (Events) Aufnahme (A01), Verlegung (A02) und Entlassung (A03) sollen bedient werden. Die Funktionen **admit_patient()**, **tarnsfer_patient()** und **discharge_patient()** nehmen als ihr einziges Argument die jeweilige Anforderungsnachricht (ADT^A01, ADT^A02 bzw. ADT^A03) und geben eine ACK-Nachricht zurück. Diese Funktionen werden nun als Behandlungsroutinen (Event handler) für ihr jeweiliges Event registriert. Nachdem der Server gestartet wurde wird eine bestimmte TCP/IP-Adresse (**port**) Bedient. Anstehende Anforderungsnachrichten werden gelesen und der Event handler aufgerufen, der für die entsprechende Nachricht registriert wurde. Liegt ein nicht unterstütztes Event vor, dann wird mit einer negativen ACK-Nachricht (AR) geantwortet. Wie man sieht, beschränkt sich der Programmieraufwand für einen solchen Server nun ganz auf die Programmierung der Event handler Funktionen.

```
ACKmsg *admit_patient(ADT_A02msg       *request);
ACKmsg *transfer_patient(ADT_A02msg    *request);
ACKmsg *discharge_patient(ADT_A02msg   *request);

...

ANYmsg srv;
srv.register_event(UniMesIdCode::ADT_A01val, admit_patient);
srv.register_event(UniMesIdCode::ADT_A02val, transfer_patient);
srv.register_event(UniMesIdCode::ADT_A03val, discharge_patient);
srv.serve(port);
```

Abbildung 12: Eine einfacher ADT-Server, der die Events A01, A02 und A03 unterstützt. Die Behandlungsfunktionen werden im Server-Objekt registriert, bevor es gestartet wird. Die auf dem TCP/IP Port eingehenden Nachrichten rufen dann ihre jeweilige Behandlungsroutine auf.

Natürlich muß bei Netzwerkverbindungen auf die Sicherheit geachtet werden. Dazu wird mit ProtoGen/HL7 derzeit die Adresse des Clients in einer Hosttabelle aufgesucht, in welcher festgelegt ist, ob und welche Nachrichten von dem entsprechenden Client akzeptiert werden. Andere Sicherheitsmechanismen (wie etwa Kerberos) sind Alternativen, die allerdings mehr Anforderungen an die kommunizierenden Systeme richten.

Praktische Erfahrungen

Abbildung 13 zeigt schließlich die bisherige Anwendung von ProtoGen/HL7 im Universitätsklinikum Benjamin Franklin. Auf der operativen Intensivstation (IOP) sind derzeit zwei Dokumentationssysteme im Einsatz: Eines mit mehr administrativer und ein anderes mit eher klinischer Ausrichtung. Das administrative System ist eine auf DOS/dBase selbst entwickelte Datenbankanwendung.

Das klinische System ist ein kommerzielles Produkt, welches einen sehr eingeschränkten HL7-Dialekt als Server unterstützt: ADT- und ORU-Nachrichten werden akzeptiert. Die Kommunikation kann nur über eine serielle ANSI X3.28 getriebene Schnittstelle erfolgen. Ein Gateway, welches eine ProtoGen/HL7 Anwendung ist, wandelt Standard-HL7 Nachrichten in den Dialekt um und bedient einen TCP/IP Port wie oben beschrieben. Zu diesem Port können nun verschiedene Systeme Verbindung aufnehmen. Das Administrative System tut dies über ein weiteres Gateway, welches HL7-Nachrichten aus Dateien von einem NFS liest und an den HL7-Server schickt. Zwei Blutgasanalyse-Geräte, welche ihre Druckerausgabe über eine serielle Leitung senden können, übermitteln die Meßwerte per HL7 — ebenfalls über ein Gateway — an das Klinische System. Der klinikweite Patientendatenserver wird auf diese Weise schließlich ebenfalls mit einer HL7-Schnittstelle ausgerüstet.

Man erkennt, daß jede Anwendung, die nicht als C++ Quellcode vorliegt — derzeit sind das alle verfügbaren — ihr eigenes Gateway hat. Jedes Gateway ist eine selbständige ProtoGen/HL7-Anwendung. Dieser Ansatz unterscheidet sich grundsätzlich von dem, der einen zentralen „Interface engine“ benutzt. Hier wird jede Anwendung zusammen mir ihrem Gateway als selb-

ständige HL7-Instanz aufgefaßt. Jedes Gateway tritt mit ganz spezifische Funktionen als HL7-Schnittstelle — als Server, Client oder beides — in den Dienst seiner Anwendung, so daß mit dem Austausch oder der Änderung der Anwendung nur dieses eine Gateway angepaßt werden muß. Durch ihre Gateways kommunizieren alle Anwendungen mit einem gemeinsamen HL7-Standardprotokoll.

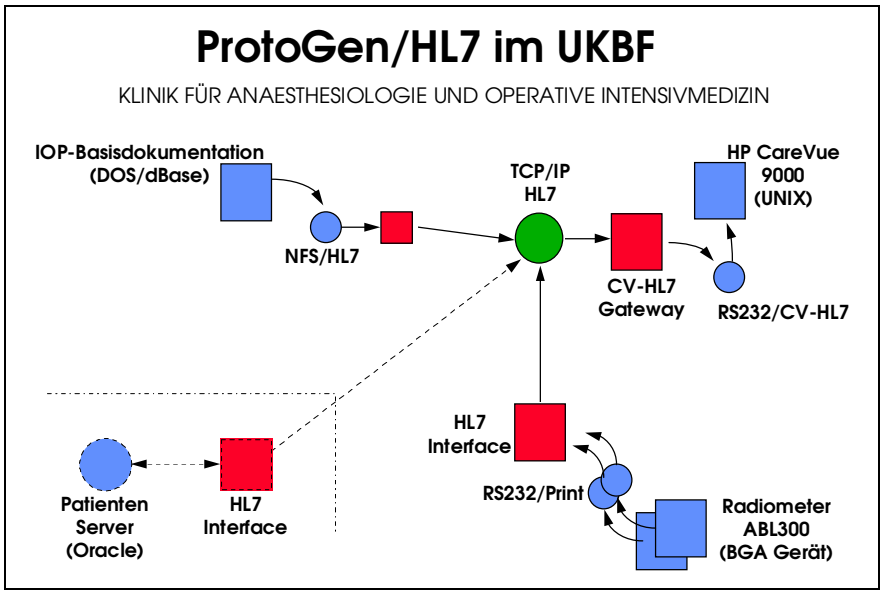

Abbildung 13: Derzeitiger Einsatz von ProtoGen/HL7: Im Zentrum steht das bettseitige Patientenmonitoring-System (CareVue). Alle HL7-Nachrichten an die Adresse „TCP/IP HL7“ werden vom CV-HL7 Gateway weitergeleitet. So die Blutgasanalysen von den Stationen (ORU). ADT-Nachrichten werden von der IOP-Basisdokumentation gesendet und künftig auch von der klinikweiten Stammdatenbank kommen.

Die Entwicklung der HL7-Bibliothek ist nicht als abgeschlossen anzusehen. Es zeigt sich nämlich, daß jeder funktionellen Bereich von HL7 (ADT, Result report, u. dgl.) durch spezifische weitergehende Abstraktionen, wie dies etwa die Klassen **`LOINC`** oder **`Unit`** darstellen, erschlossen werden muß. Durch den modularen und objektorientierten Aufbau ist aber gewährleistet, daß sich die Mühe der Erweiterung der Bibliothek lohnt sobald weitere Anwendungen auf dem neu erschlossenen Gebiet erstellt werden sollen. Schon jetzt zeigte sich nämlich, daß durch die Bibliothek, welche die Konzepte von HL7 auf hohem Abstraktionsniveau verfügbar macht, die Erstellung der einzelnen HL7 Client/Server-Anwendungen zu einer Arbeit von wenigen Tagen geworden ist.