ProtoGen/HL7 — eine C++ Klassenbibliothek für HL7

Gunther Schadow*

3. Juli 1995

Zusammenfassung

Mit dem Programmgenerator ProtoGen/HL7[1] wurde der HL7 Standard in Form einer C++ Klassenbibliothek vollständig implementiert. Dies erlaubt die schnelle Ausrüstung von Anwendungen mit einer HL7 Schnittstelle. Ergänzungen zum HL7 Standard können schnell und effizient integriert werden. Weiter lassen sich einerseits andere HL7 Schnittstellen auf Korrektheit untersuchen und andererseits der HL7 Standard selbst auf Vollständigkeit und Widerspruchsfreiheit.

1 Der Kontext

Mein Projektziel im Rahmen meiner Doktorarbeit, die ich im Herbst 1993 am Universitätsklinikum Benjamin Franklin begann, lautet, einen HL7 basierten klinikweiten Kommunikationsdienst zu entwickeln.

Die EDV Umgebung bietet, was die Hardware- und Betriebsystem-Ebene betrifft, mit leistungsfähigen Unix Workstations an einem schnellen LAN, auf welchem die Internet-Protokolle gefahren werden, sehr gute Voraussetzungen. Die Ausstattung mit hochwertiger klinischer Software, die den Anforderungen an offene Systeme entspricht, ist dagegen eher beschränkt: Ein proprietäres KIS und eine selbst entwickelte DOS/dBase Patientendatenbank auf der anaesthesiologischen Abteilung, ein proprietäres RIS in der radiologischen Abteilung, ein proprietäres Stammdatensystem und einige PC Inselapplikationen. Einzig das KIS der Anaesthesiologie weist eine bescheidene HL7 Schnittstelle auf.

Jedenfalls wurde zunächst eine möglichst vollständige HL7 Implementation gebraucht. Da keine geeignete fertige Software zur Verfügung stand, mußte schließlich selbst eine erstellt werden.

2 Die Methode

Der große Umfang des HL7 Standards erfordert eine weitgehend automatisierte Methode, um in angemessener Zeit eine zuverlässige Implementation zu erstellen, die sowohl an lokale Gegenbenheiten anpassungsfähig, als auch für spätere Versionen von HL7 weiter verwendbar bleiben soll.

Mir lag die deutsche Edition der HL7 Version 2.1 in Form von Word 5.0 Dateien vor, die abgesehen von binären Formatköpfen und -Anhängen, ASCII texte sind. Damit konnte ich mit den üblichen Unix-Werkzeugen „sed“ und „awk“ die formalisierten Informationen, wie Nachrichten- und Segment-Definitionen, sowie Code-Tabellen extrahieren. Zur Weiterverarbeitung wurden die gewonnenen Informationen in Prolog Syntax gebracht. Die programmiersprache Prolog eignet sich besonders gut, um Annahmen an der Wissensbasis zu beweisen. Dies waren zunächst Konsistenzprüfungen.

Dabei stellte sich heraus, daß der HL7 Standard in jener Version viele Mängel hinsichtlich der Konsistenz und Vollständigkeit aufwies. Dies reichte von undefiniertteten Nachrichten und Segmenten, über fehlende, als „HL7-Standard“ deklarierte Code-Tabellen, bis zu Syntax-Fehlern in den Nachrichtendefinitionen. Nicht alle dieser Fehler ließen sich mit Hilfe des narrativen, den automatischen Methoden zunächst unzugänglichen Teils des Textes korrigieren. Manches mußte improvisiert werden, und nicht alle dieser Improvisationen erwiesen sich später als richtig.

Nachdem die HL7 Wissensbasis manuell in einen weitgehend konsistenten Zustand gebracht wurde, konnte in einem zweiten Schritt ein Programmgenerator geschrieben werden, welcher den eigentlichen Quellcode der HL7 Implementation produzieren sollte. Vorbilder zu diesem Vorhaben waren

*schadow@ukbf.fu-berlin.de

[1]ProtoGen/HL7 ist freie Software. Bezug über WWW: http://fub46.zedat.fu-berlin.de:8080/~gusw/ oder FTP: dumccss.mc.duke.edu:standards/HL7/implementations

ähnliche Generatoren, wie sie in Sun-RPC („rpcgen“) und ISO/OSI Umgebungen (ASN.1 Compiler) benutzt werden.

3 Anforderungen

Die HL7 Implementation sollte folgenden allgemeinen Anforderungen genügen:

- in einer Objektbibliothek für bestehende und zukünftige Anwendungen verfügbar sein
- in einer verbreiteten, unabhängigen Programmiersprache geschrieben sein
- die HL7-Konzepte (Message, Segment etc.) möglichst klar und direkt wiedergeben
- in der Anwendung so einfach wie möglich und dennoch ausdrucksstark sein
- das OSI-Referenzmodell beachten, insbesondere die Schichtung nicht verletzen, bzw. dort zu kompensieren wo HL7 diese Schichtung verletzt.

Nachdem ich die ersten Ansätze in der Programmiersprache C geplant hatte wurde bald deutlich, daß insbesondere das Ziel der Einfachheit und Klarheit nur mit den objekt-orientierten Erweiterungen zu erreichen waren, wie sie C++ bietet. Hiermit ist es möglich, selbst komplizierte Aspekte der Implementation einiger Module, vom Anwendungsprogrammierer fern zu halten. Dieser kann beispielsweise ein wiederholbares Feld wie ein Array behandeln, obwohl dem eigentlich eine verkettete Liste zugrunde liegt. Er braucht sich nicht um dynamischen Speicher zu kümmern. Zeichenketten, selbst ganze Segmente und Nachrichten können wie atomare Werte behandelt werden.

Der Anwendungsprogrammierer darf sich selbstverständlich nicht darum sorgen müssen, daß gewisse Zeichen, die er in einem Textdatum verwendet, mit den Trenn- oder Steuerzeichen irgendwelcher tieferer Protokollschichten interferieren könnten. Diese Dinge bleiben vor ihm verborgen, wie es das OSI Schichtenmodell fordert.

4 Grundfunktionen

Realisiert wurden zunächst die folgenden Dienste, die vor allem in der Präsentations-Schicht (6) nach dem OSI-Referenzmodell liegen:

1. Datenstrukturen als Schnittstelle zur Anwendungs-Schicht (7)
2. eine Funktion, die die Datenstrukturen mittels der HL7 Codierregeln serialisiert („builder“).
3. eine Funktion, die einen serialisierten HL7 Zeichenstrom analysiert und die Datenstrukturen entsprechend aufbaut („parser“).

Abbildung 1 gibt einen Überblick über die Struktur der HL7 Klassen. Aus Platzgründen sind nur die abstrakten Klassen aufgeführt, während sich die konkreten jeweils von ihrer abstrakten Klasse ableiten. So ist zum Beispiel das MSH-Segment von der Klasse „Segment“ und der DT-Datentyp von der Klasse „Primtype“ abgeleitet.

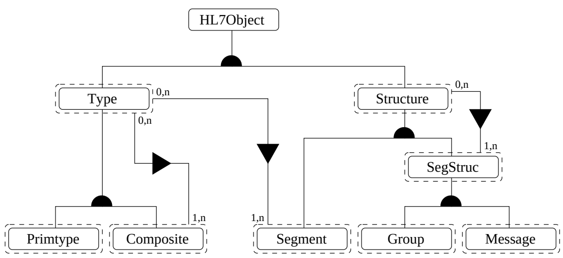

Abb. 1: ProtoGen/HL7 Klassenhierarchie

Die Datenstrukturen (C++ Klassen) gestatten wahlfreier Zugriff auf die Inhalte der Nachrichten, wobei die Reihenfolge der Felder nicht beachtet werden muß. Die Bezeichner werden mit einem einfachen Algorithmus aus den standard Datenelementnamen gebildet, wodurch sie kurz und dennoch lesbar bleiben. Man kann z.B. auf den Patiennamen in einer ADT-Nachricht `msg` symbolisch einfach so zugreifen:

```
msg.PatIde.PatName.FamName = "Meier";
```

Jede Klasse besitzt eine Reihe von Funktionen (Methoden), von denen ich hier nur die Ein- und Ausgabe-Funktionen nennen will. Diese lesen von, bzw. schreiben auf eine Zeichenstrom-Abstraktion, wie sie mit der standard „IOstream“ Bibliothek zur Verfügung stehen. Diese Ströme können beliebig mit einem konkreten Ein-/Ausgabegerät verbunden werden, wie einer Datei, einem TCP/IP Socket, einer serielle Schnittstelle u.dgl. Die anderen Methoden sind vor allem am Management der

Objekte beteiligt. Abbildung 2 zeigt beispielhaft die Deklaration des NK1-Segmentes.

```
class NK1seg : public Segment
{
 public:
  SItyp           SetIdNextOfKin;
  PNtyp           NextOfKinName;
  STtyp           NextOfKinRel;
  ADtyp           NextOfKinAdd;
  repfield<TNtyp> NextOfKinPhoneNum;

  NK1seg();

  result input(istream&);
  void   output(ostream&) const;
};
```

Abb. 2: Deklaration der NK1 Segment-Klasse

Das Konzept von ProtoGen/HL7 ist nicht auf HL7 beschränkt. Mit relativ geringem Aufwand können auch HL7 ähnliche Standards (EDIFACT, ANSI X12, ASTM) integriert werden. Durch den objekt-orientierten Aufbau können auch ganz andere Codierungsregeln verwendet werden, wie sie zum Beispiel bei DICOM Verwendung finden.

5 Praktische Anwendung

Mit der so erstellten Bibliothek waren HL7 Testanwendungen in sehr kurzer Zeit zu erstellen. Dies sind: eine Schnittstelle zu einer Oracle Patientendatenbank und ein Gateway zu dem oben genannten proprietären KIS. Eine Schnittstelle zu einem Labor-Roboter wird in nächster Zeit erstellt und weitere Schnittstellen, wie zur Radiologie oder Mikrobiologie, sind geplant. Besonders bei der Anbindung des KIS, dessen HL7 Schnittstelle Z-Segmente gebraucht, machte sich die flexibilität meiner Methode bezahlt: Die Abweichungen vom HL7 Standard waren binnen einer halben Stunde gebrauchsfertig implementiert.

Bei jedem der Entwicklungsschritte treten je eigene Probleme auf, die mit HL7 zusammenhängen. Waren es bei der Implementaion des Standards selbst nur die formalen Inkonsistenzen, so wird bei der Entwicklung von Anwendungen die Unvollständigkeit und nur vage Bestimmtheit von HL7 zum überragenden Problem. Fragen wie die nach dem Unterschied zwischen „patient type“, „patient class“ und „admission type“ sind anhand des offiziellen Standards ebenso wenig zu klären wie die, wie man die Information über den Hausarzt konsequent und vollständig unterbringen kann. Fehlende Codetabellen tun ihr übriges, um Interoperabilität mit HL7 zu erschweren.

6 Ähnliche Arbeiten

Seit dem Beginn der Arbeiten an ProtoGen/HL7 wurden auch andere HL7 Implementationen bekannt. „HL7ImExa“[2] ist eine vergleichsweise einfache Implementation, bestehend aus einigen C-Funktionen, die desto schwieriger in der Anwendung ist. Sie bietet keine Datenstrukturen, und ist nicht typensicher[3], geschweige denn objektorientiert.

Eine andere Implementation in C++[4] deckt nur eine Untermenge von HL7 ab, nämlich die Schnittmenge mit DICOM. Der Code ist manuell erzeugt und daher wenig flexibel. Obwohl C++ benutzt wird, ist die Programmierschittstelle weniger Komfortabel und nicht typensicher, da ebenfalls alle Daten als Strings behandelt werden.

7 Ausblick

Noch ausstehende Arbeiten an der HL7 Bibliothek sind neben der Aufnahme von Version 2.2 vor allem Unterstützung in den Netzwerkfunktionen, um auf alle möglichen Übertragungsmedien in gleicher Weise zugreifen zu können (`connect()`, `accept()` etc.).

Des weiteren arbeite ich daran, die Bibliothek um eine Remote procedure call (RPC) Semantik zu erweitern. Damit soll es möglich sein, von den Nachrichten selbst zu abstrahieren, und nur noch mit Informationen und Aktionen zu handeln. So soll der Anwendungsprogrammierer von der Aufgabe befreit werden, die richtige ACK Nachricht auf eine Message zu empfangen (bzw. zu senden), zu prüfen, und auf eventuelle Fehler zu reagieren. Um Antwortnachrichten sollte er sich nur kümmern müssen, sofern sie angeforderte Informationen enthalten. Damit ist auch ein Weg vorgezeichnet, wie über die Implementation von statischen Datenmodellen hinaus auch dynamische, operationale Aspekte des HL7 Standards formal erfaßt und zuverlässig implementiert werden können.

[2] Allen Rueter <allen@mir.wustl.edu>

[3] d.h. man kann versehentlich eine Aufnahmenummer als Geburtsdatum interpretieren.

[4] S. Munjal, et.al.: A C++Class Library for DICOM and HL-7, Proceedings of CAR '95, Berlin 1995, Springer-Verlag