Health Level 7 (HL7): A Core Parser

by

Scott Au

A CAPSTONE PROJECT

Presented to the Department of Medical Informatics & Clinical Epidemiology

and the Oregon Health Sciences University

School of Medicine

in partial fulfillment of

the requirements for the degree of

Master of Medical Informatics

May 2003

School of Medicine

Oregon Health & Science University

Master Of Medical Informatics

Certificate of Approval

This is to certify that the Capstone Project of

Scott Au

"Health Level 7 (HL7) A Core Parser"

Has been approved

[redacted]

Professor in charge of capstone project

5/14/03

Date

Table of Contents

Acknowledgements

I would like to thank my advisor, Dr. Christopher Dubay, for his assistance and enthusiasm. His extensive knowledge helped focus rambling brainstorms into what is now this project. I would also like to thank Andrea Ilg for her wonderful support of distance learning students and without whom this project would have not been possible. Thank you also to Dr. William Hersh and the other faculty at OHSU's Department of Medical Informatics & Clinical Epidemiology for providing such an excellent environment in which to learn. Lastly, thank you to Dr. Susan Price for her Java programming classes which were incredibly helpful in writing the code.

Abstract

In 1987 Health Level Seven, Inc. (www.hl7.org) was founded as a non-profit, ANSI-accredited standards developing organization with the mission of developing a comprehensive framework and standards for the exchange of electronic health information. The name references the 7th level of the International Standards Organization's (ISO) Open Systems Interconnection (OSI) model, also known as the application level.

While the organization's work has contributed to several areas in the health information field, its most visible effort is a widely adopted messaging standard that permits disparate healthcare applications to exchange clinical and administrative data. Efforts are now focused on finalizing version 3.0. Version 2.4 was the last version submitted for and approved by the American National Standards Institute (ANSI). The success of this messaging standard has led to international adoption by countries including Argentina, Australia, Canada, China, Czech Republic, Finland, Germany, India, Japan, Korea, Lithuania, The Netherlands, New Zealand, Southern Africa, Switzerland, Taiwan, Turkey and the United Kingdom.

The success of HL7's messaging standard has led many software applications developed for handling health information to base their data transmission protocols on it. Some of the commercial offerings can be quite expensive, anywhere from hundreds to thousands of dollars per individual license. Unfortunately, this can be prohibitive for those health information professionals who have only begun to explore HL7's capabilities or need just

a portion of the software functions. The proprietary nature of commercial packages also makes it difficult to home grow a custom solution due to inaccessibility to source code and legal restraints. Due to these and other problems, open-source projects have become an option to the commercial products. Some of these open projects have flourished with the aid of volunteers and can be quite robust. Some offer a range of utilities with well-designed graphical user interfaces while others have a specific purpose such as linking to mainstream databases. There are drawbacks as well however, such as lack of efficient documentation as well as the constant reworking of software that, while fixing known bugs, also introduces new errors.

The goal of my project therefore, is to create a core HL7 parser that will emphasize simplicity and flexibility. I want to focus on simplicity for the sake of the end users whose only desires are to parse HL7 documents without the need for complicated manuals or extensive training. Flexibility is also important, as I believe it would be of great help for other developers seeking to build more complex HL7 applications to have a basic HL7 parser that is easily modifiable. These reasons are why I have chosen the JAVA programming language as it largely platform independent. The language is quite mature and there are large libraries of tested routines and functions available. Finally, the popularity of the language and the possibility to use JAVA applets online increases the likelihood of its use. As version 3.0 has not yet been submitted for approval by the ANSI, my parser will only be compatible with version 2.4.

Introduction

Background:

Several decades ago, the limitations of computers and computer related technologies, in addition to their prohibitive costs, restricted their possible uses. However, once manufacturing breakthroughs improved performance and subsequent demand reduced expenses, these resources were quickly exploited for their power and utility.

The field of medicine made substantial progress in theory and practice by taking advantage of this growth in technology. Recent accomplishments include DNA micro-arrays, cloning techniques, high throughput DNA sequencers, bio-informatics applications for storing and retrieving sequence annotations, and pharmaceutical databases. Unfortunately, these highlights overshadow the lack of technology use in the healthcare field itself, more specifically information technology. It is ironic that individual areas of medicine have obviously attained "state-of-the-art" status yet the state of the healthcare apparatus itself lags behind its constituent parts.

The article by Brender, et al, illustrated this situation by noting that:

> "Technology has enabled mankind to walk on the moon. We are able to implant artificial organs, to aid artificial hearing and vision, and to enable action by thought. Medical devices supporting/enabling diagnosis, therapy and monitoring (example MRI-techniques) will be more and more software driven, Et cetera, etc. However this is not part of the day-to-day medicine for the compensation of physical handicaps or treatment of diseases."[1]

This "day-to-day" medicine continues to involve paper records, inefficient insurance claims processing, and hospital information modules that cannot routinely communicate with one another. Even today the use of paper charts remains the default means for storing patient information, despite all its limitations. A recent study published by Bates, et al in 2003 reported that only about 5% of US primary care providers use electronic medical records (EMR).[2] Why has the healthcare system not benefited as much from the technology boom as has medical treatment and research? The answer involves a number of factors.

First, healthcare has not historically embraced information technology [IT]. The industry pooled only 2 percent of its total revenue for IT funding in 1998. In contrast, the financial services sector devoted more than 7 percent[3]. Additionally, "only 6 percent of the health care industry uses the Internet for buying supplies today, compared to 25 of companies in other industries."[4]

It is important to note that the IT spending discussed here is distinct from the funding of medical research and products. While the market for information technology software for medicine received a boost during the stock market growth of the 1990s, it is dwarfed by the investment dollars spent on pharmaceutical and new medical devices. Mcintosh reports that, "the most rapidly growing area of healthcare expenditures [is] prescription drugs [which] accounted for 20% of increased healthcare spending."[5]

This imbalance is probably best explained by the direct valuation of the latter. It is much easier to gain research funding for medical products such as devices and pharmaceuticals that can directly save lives or show an immediate return on investment. Healthcare budget planners receive the notion of a streamlined healthcare system that improves efficiency, reduces errors, and cuts back on unnecessary expenditures with more head nodding than action. The journal Medical Economics wrote that "investment funds may be placed in pharmaceutical firms as they are a more profitable area of the health care sector than other areas, such as home health-care firms or health management organizations."[6] Noffsinger also stated "The medical community has been slower to invest in new technology because IT expenses are often not viewed as directly related to better patient care, but instead are seen as a necessary evil required for administration."[7] As a result, Grossman writes that, "We have mistakenly focused on delivering more and better tests and procedures instead of on improving the health, well-being, and satisfaction of patients."[8]

Second, administrative and financial events factors have made unification of the system difficult since "communication barriers stand between the various provider units (primary care and specialist physicians, hospitals, laboratories, rehabilitation and nursing home facilities, and home health providers); between the providers, the health plans, and the payers; and between the patient and the health-care system."[8]

Third, the implementations of many medical IT applications in medicine have met with significant resistance by health workers who view them as being too slow or too complex. The learning curve for many of these applications can be high initially, especially for users who have less experience with software applications. It is possible that certain workers may end up with greater workloads after an implementation for the system to benefit as a whole. Logically, their protests create a significant barrier to IT adoption. In a study by Weiner, et al, a comparison was made between the attitudes of nurses and physicians towards a new Provider-Order Entry system. They reported "physicians indicating that POE decreased time spent with patients but nurses indicating increased time."[9]

Fortunately, the drive to streamline the healthcare system has gained greater momentum in recent years. More and more researchers, clinicians, business managers, and even patients have realized the current situation reeks of inefficiencies and lost opportunities. The absence of easily accessible and highly organized patient information diminishes the pool of possible subjects for clinical researchers. Administrative personnel remain encumbered by the costly and erroneous manual collection of operational statistics that could otherwise be automated. The insurance industry likewise can do away with slow and redundant processes that increase overhead and ultimately raise premiums for policyholders. Mcintosh reports that "Administrative costs in hospitals increased considerably in response to the increasingly complex healthcare environment, while physicians' billing expenses likewise have grown substantially in an environment with more than 1200 insurers, each with its own procedures and requirements. Most of these costs are attributable to overhead within the health insurance industry."[5] An example is "hospital staffs that monitor the quality and cost of care now spend much of their time poring through reports to identify incidents that fail to meet specified criteria. With new systems that automatically flag variances, the staff occupy themselves with the more challenging task of analyzing the causes, effects, and solutions."[8]

Attention paid to these problems has culminated in a greater industry push towards remedying the situation. A milestone was recently reached when Congress passed HIPAA legislation. This bill will create a smoother and more efficient healthy system that protects the security and confidentiality of patient information. Buckovich, et al, in their article "Driving Toward Guiding Principles," reported that, "HIPAA mandates the development of a national privacy law, security standards, and electronic transactions standards and provides penalties for standards violations and wrongful disclosures of health information."[10]

Clearly the work to modernize the healthcare system will be more complex than many can imagine. Changes in the economy, insurance schemes, patient rights, technology, and popular opinion will all affect future decisions. Since the task of analyzing such a vast topic is beyond the scope of this capstone project, the focus will instead be on the standardization of healthcare, specifically the messaging subsection. This area will be of key importance in creating an information system within healthcare devoid of obstructions. "'Health care trails almost every other field in information technology application because of a lack of consistent and uniform standards, protocols and language,' says Gene O'Dell, vice president of strategic and business planning for the AHA. 'Standardization will be a major building block in health care IT culture.' Standardization of information and interfaces has long been a Holy Grail of IT, especially in health care. A common frustration, as McDevitt found, is that advanced systems have trouble talking to each other.'"[11]

One format in particular, the Health Level 7 standard (HL7), has provided gained significant adoption in medical IT. With its majority position in the industry, HL7 offers a real opportunity to overcome many long-standing obstacles such as heterogeneity of data, lack of appropriate protocols, and incomplete vocabularies. With ANSI ratification of version 2.3, the continuation of HL7's adoption will hinge on increasing the number of tools available to users. This will be vital when version 3.0 becomes ratified, as the inclusion of XML will increase the complexity of HL7's design, though the resultant benefits are well worth it. Until that time however, those health professionals just beginning to delve into HL7 will find many of the commercial packages off limits due to licensing fees. Unfortunately, this group of users is growing larger and includes a growing number of people who can make significant contributions to the field. Their options are then limited to current open-source projects that do offer attractive functionality. Still, many of these projects proceed at inconsistent rates and many lack proper documentation as the code is constantly reworked. Other projects have been terminated due to lack of resources. The aim of this capstone project is to provide very basic open-source code that provides the kind of functionality beginning users will most likely want. This includes general parsing of HL7 messages for easier reading as well as common search functions. One example is the ability to search and find which HL7 files contains a particular transaction or segment.

Materials and Methods

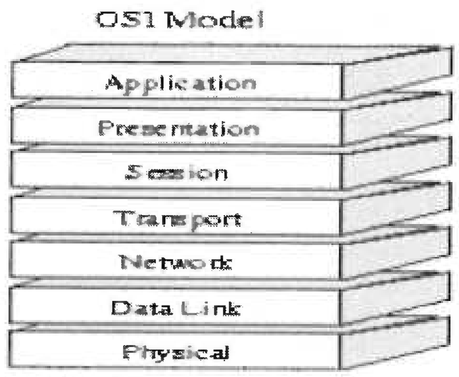
Gathering information regarding HL7 led to the HL7.org website which provided a great deal of history as well as the full standard itself in PDF format. Regarding HL7 as an organization:

HL7:

In 1987 Health Level Seven, Inc. was founded as a non-profit, ANSI-accredited standards developing organization with the mission of developing a comprehensive framework and standards for the exchange of electronic health information. The name references the 7th level of the International Standards Organization's (ISO) Open Systems Interconnection (OSI) model, also known as the application level.

While the organization's work has contributed to several areas in the health information field, its most visible effort is a widely adopted messaging standard that permits disparate healthcare applications to exchange clinical and administrative data. Efforts are now focused on finalizing version 3.0. Version 2.4 was the last version submitted for and approved by the American National Standards Institute (ANSI). The success of this messaging standard has led to international adoption by countries including Argentina, Australia, Canada, China, Czech Republic, Finland, Germany, India, Japan, Korea, Lithuania, The Netherlands, New Zealand, Southern Africa, Switzerland, Taiwan, Turkey and the United Kingdom.

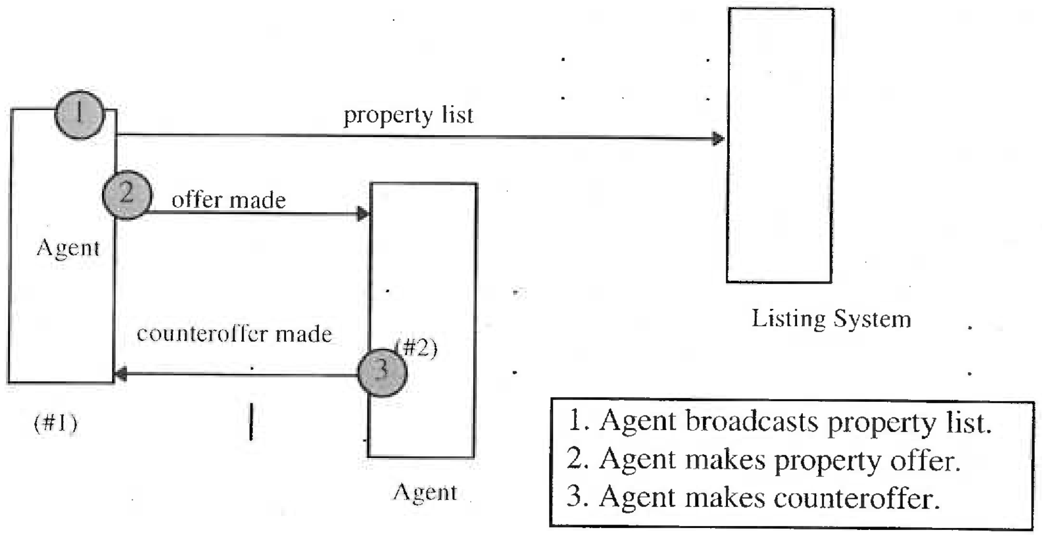
The name Health Level 7 (HL7) as the messaging standard refers to the Application Level of the OSI model:



Graphic courtesy Freesoft.org

The model was created to divide the work of interconnecting hosts into seven layers that stack upon each other. This makes the task more manageable since problems that occur in a layer can be isolated and focused on rather than examining the system as a whole. It is also extensible, since new protocols or technologies can be readily implemented. In HL7's case, the standard developers only needed to focus on the top level, the application level and could afford to ignore the rest. Established methods from the bottom six layers already provide the necessary means for supporting HL7.

In HL7 messaging, real world events trigger messages. Such events could be an admission of a patient that then triggers a Laboratory message. The following is a generalized diagram of messaging (courtesy of the HL7 Message Development Framework Report, HL7 Quality Assurance Committee, February 28, 1996):


property list
offer made
Agent
counteroffer made
(#2)
(#1)
Agent
Listing System
1. Agent broadcasts property list.
2. Agent makes property offer.
3. Agent makes counteroffer.

In a clinical or hospital setting, the Agent entities can be replaced with Admissions, Laboratory, Medical Records, etc. The Property List can be seen as the initial admissions message which is sent to be processed by the Listing System and the counteroffer made is data returned by the processing.

In regards to HL7 *messages*, they are constructed as a list of *segments* followed by *data fields*, which contain respective *data elements* also known as *components*. These data elements themselves may also have components also known as *sub-components*. Fields may repeat and the maximum number of repetitions can also be repeated. Data types can be compound, such as addresses, phone numbers, and patient names. Data types can also be simple, such as a name, percentage, and percentage. Delimiter characters are used to separate the parts of the message:

Segment terminator	0x0d
Field separator	|
Component separator	^
Sub-Component separator	&
Repetition Character	~
Escape Character	\

All segments are prefaced by a three letter code such as:

MSH: **Information about the message itself.**

PID:
AL1: **All 3 convey patient information.**
PV1:

ORC:
OBX: **Information regarding the order.**
BLG:

There are more than 120 segments currently defined as well as a customizable Z-segment. The MSH segment is always found as it contains necessary information regarding the message. Within in different type of segment, any number of data fields maybe required. The full standard can be found at HL7.org. A sample HL7 message follows:

```
MSH|^~\&|EPIC|EPICADT|SMS|SMSADT|199912271408|CHARRIS|ADT^A04|18174
57|D|2.3|
EVN|A04|199912271408|||CHARRIS
PID||0493575^^^2^ID 1|454721||DOE^JOHN^^^^|DOE^JOHN^^^^|19480203|M||B|254
E238ST^^EUCLID^OH^44123^USA||(216)731-
4359|||M|NON|400003403~1129086|999-|
NK1||CONROY^MARI^^^^|SPO||(216)731-4359||EC|||||||||||||||||||||||||
RXA|0|1|19900607|19900607|08^HEPB-
PEDIATRIC/ADOLESCENT^CVX^90744^HEPB-PEDATRIC
/ADOLESCENT^CPT|.5|ML^^ISO+||03^HISTORICAL INFORMATION - FROM
PARENT=S WRITTEN RECORD^NIP0001|^JONES^LISA|^^^CHILDREN=S
HOSPITAL||5|MCG^^ISO+|MRK12345|199206|MSD ^MERCK^MVX
PV1||O|168   ~219~C~PMA^^^^^^^^^^||||277^ALLEN FADZL^BONNIE^^^^|||||||||
||2688684|||||||||||||||||||||||||199912271408|||||002376853
```

Due to the constraints of paper width, the message lines are truncated here. Real messages segments however flow horizontally until complete and are only terminated by a carriage return character (0x0d). Note the proliferation of sequential |'s signify empty fields. Note also that the HL7 message standard utilizes flat text files which should be human readable, but given the compacting, reading such messages manually is too time consuming.

When a message is received, software allocated for parsing the message retrieves the necessary data which is then utilized however desired. The flexibility of the standard allows software developers to create specific applications that are not required to handle the standard in its entirety. An example is the CDC Immunization Record EXchange (iREX) Project, found at http://dt7.com/cdc/. As its focus is on transmission of immunization and VAERS information, the messages used in this project only include specific segments.

Since my aim was to create a core parser, meaning a generalized program that could be modified for more specific intents, I left out handling of escape sequences. These include:

&	\T\
^	\S\
|	\F\
~	\R\
\	\E\

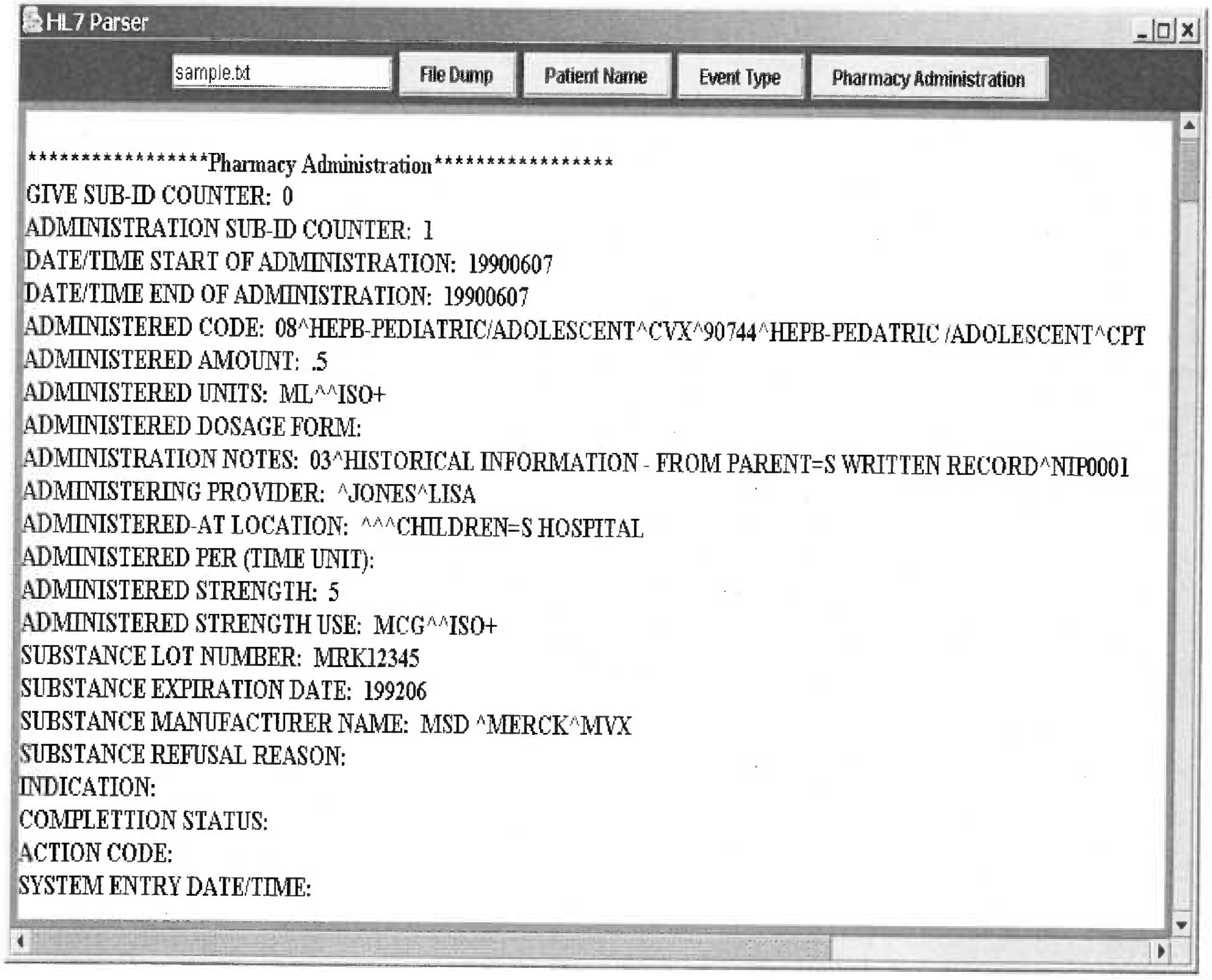
The last symbol itself is the escape character which also has an escape sequence. Since not all message will contain delimiter characters, it made little sense to incorporate the escape sequences. They can be added by anyone wishing to tailor their own program.

JAVA:

The JAVA programming language was written in the idea that a platform-independent language would lead to more efficient and powerful software development. This would make obsolete the practice of manually porting applications from platform to platform, such as Windows to Macintosh to the various flavors of Unix. Unlike other programming languages, where the source code is either compiled or interpreted, Java source code is first translated into bytecode. This bytecode is platform independent code that is then interpreted by interpreters specific to each Java platform. As long as there is an interpreter for a platform, such as a Windows-Intel machine, Java applications can be run. The benefit is that compilation is only needed once. JAVA has been chosen by other medical informatics projects such as reported in Schadow, et al's, article entitled, "Units of Measure in Clinical Information Systems" where it provided an open-source implementation of a unit conversion model for clinical measurements.[12] Sun Microsystems maintains the JAVA language and additional information can be found at http://java.sun.com.

Results

The following is a screenshot of the parser:



```
****************Pharmacy Administration****************
GIVE SUB-ID COUNTER: 0
ADMINISTRATION SUB-ID COUNTER: 1
DATE/TIME START OF ADMINISTRATION: 19900607
DATE/TIME END OF ADMINISTRATION: 19900607
ADMINISTERED CODE: 08^HEPB-PEDIATRIC/ADOLESCENT^CVX^90744^HEPB-PEDATRIC /ADOLESCENT^CPT
ADMINISTERED AMOUNT: .5
ADMINISTERED UNITS: ML^^ISO+
ADMINISTERED DOSAGE FORM:
ADMINISTRATION NOTES: 03^HISTORICAL INFORMATION - FROM PARENT=S WRITTEN RECORD^NIP0001
ADMINISTERING PROVIDER: ^JONES^LISA
ADMINISTERED-AT LOCATION: ^^^CHILDREN=S HOSPITAL
ADMINISTERED PER (TIME UNIT):
ADMINISTERED STRENGTH: 5
ADMINISTERED STRENGTH USE: MCG^^ISO+
SUBSTANCE LOT NUMBER: MRK12345
SUBSTANCE EXPIRATION DATE: 199206
SUBSTANCE MANUFACTURER NAME: MSD ^MERCK^MVX
SUBSTANCE REFUSAL REASON:
INDICATION:
COMPLETTION STATUS:
ACTION CODE:
SYSTEM ENTRY DATE/TIME:
```

As seen above, I chose a very simple layout so as to reduce the complexity of the JAVA code needed. There is a small text entry area, enabled by a JTextField, for inputting the filename which is followed by four buttons. Each button is associated with an ActionEvent which represents the type of functions which can be coded. The JTextField and all four buttons are added to the top JPanel which itself is placed in the northern layout of the JFrame. The entire bottom area is a JTextArea which is maximized for screen output. A JScrollPane is used to enable vertical and horizontal scrollbars to handle exceptionally long data segments. The JScrollPane is focused on the JTextArea and placed in the center of the JFrame. My intent is for users to customize their own buttons to represent functions. The four chosen by me were picked randomly and have different duties to illustrate the potential.

The File Dump button does exactly that; it parses through the HL7 message and attempts to represent the information as cleanly as possible. Blank fields are not represented with their own blank lines in the display as the numerous amounts of blank fields can make viewing quite difficult. Any sub-components are indented 5 spaces and any sub-sub components are indented 10 spaces.

The Patient Name button searches through the HL7 message for the PID segment and locates the 5th data field which, according to the HL7 standard, contains the patient's name. I wanted this to be an example of very specific data retrieval. I imagine that many encoding functions will utilize this framework for seeking specific data components.

The Event Type and Pharmacy Administration both seek out their representative segments, EVN and RXA, and displays all the data fields for these segments. The code here is an example for those users who wish to print out blocks of data. While this is nothing more than a glorified file dump routine, the current code should save time for those users who wish to it. The routine for preventing overrunning the carriage return character required some extra coding which can be a waste of time.

I ran numerous sample HL7 messages gleaned from the Internet through the Google search engine and found the parser had no problem parsing them. If any of the segments I chose, PID, EVN, and RXA, were not found, the program merely states so. I would have liked to code a more complex display to better represent sub-components but decided many users would rather customize their own display to highlight certain information. Thus I merely added tabs to indent sub-components and left repeated data fields alone.

Discussion

I began the coding project by first designing the GUI. To be as simple as possible, I only wanted a small space for the filename, a large area for the results display, and some room for the function buttons. Using JTextArea, JTextField, JScrollPane, JPanel, and JFrame classes, I was able to put together the GUI shown above.

The parsing code was a little trickier to write as the variety of situations that may occur with the delimiter characters presented a challenge. Adding the various data fields cost nothing more than the labor of typing but I did have to write a separate function to return the data field values. This was done to reduce the amount of duplicate code in the program which was beginning to affect the startup time. The need to evaluate strings to find the segment headers also complicated things as the parser itself runs through the messages character by character. The methods for reading the file, through the BufferedReader class, also required extra thought as reading frame can be shifted by skip() methods but the character will not be read.

All in all, the basic components of the parser should be enough to serve as a framework for other users who wish to tailor it. The only real coding they would need to do is to create or modify their own function buttons. This should be quite easy as the code structure I have written separates each button's ActionEvent into its own class. After coding their function, they only need to add the button to the top Jpanel. I also left the information display commands inside the button's ActionEvents so these too can be customized to be specific to each function. Whereas it might be easier to display

Pharmacy information if there data fields there contain mostly primitive data types, it might be more meaningful to display Patient information in a tiered method, to better highlight those compound data types such as first and last name.

Creating basic function buttons such as the ones I wrote should not be difficult considering all the necessary data field names can be found in the HL7 standard at the HL7 website. For those buttons which may entail more coding, such as a function that retrieves specific data and re-encodes it into a file for database addition, the complexity will certainly go up. Still, it should not be hard given the clear and simple setup of the HL7 messages and the available JAVA routines.

An item I believe other users might find valuable but did not code include a collapsible window to search for the HL7 messages on the file system instead of having to type them out. Another routine would be to add the ability to process an entire directory of messages so that bulk amounts of HL7 messages can be parsed. In terms of output, a printer function would be convenient for those users who want a hard copy and for other data purposes, such as database entry, a means for formatting parser output into specific formats such as Oracle. Another idea, one that I believe many users would opt for, is to transform the parser into an applet that could be used online. The applet could be hosted and accessed from a specific webpage and with the proper permissions, produce local output. The servlet would contain all the necessary code with the applet only contained the aesthetic portions to transmit the HL7 message.

Another event which will undoubtedly make this parser obsolete is the eventual ANSI ratification of HL7 version 3.0. This version, currently in the works, implements the HL7 standard in XML, the current popular platform for data encoding. With the number of XML documents, DTDs, and Schemas being created, it makes sense to add HL7 to that growing list. The multitude of XML programs that could harvest data from XML enabled HL7 messages would result in a jump from a newly minted standard to a mature one that retains a large library of applications. Even so, the flat text based nature of XML (XML too can be written in a simple text processor such as Microsoft Notepad), will not change HL7 syntax so much that current techniques for parsing, encoding, and manipulating HL7 messages could easily be modified for the new standard.

Conclusion

In writing this program, I have found that the HL7 message standard is remarkably powerful for such a limited design. I believe this simple parser can easily be built up is so desired and expanded to include encoding HL7 data as well as parsing it. The fact that flat text could hold such a large amount of data certainly points out that medicine should not be hindered by any technical concerns. All the technology needed to solve many of the information problems in the medical field is currently accessible. What is necessary is for more health professionals, those who conduct their daily business in medicine, to add their input to the further development of applications. Since they are so aware of the needs and special considerations of their work, it is imperative that their voices be heard in the development cycle. It is my hope that my program contributes to this, even if only to spark the interest of some person in HL7, medical messaging, and the health information field.

Bibliography

1. Brender, et al. "Research Needs and Priorities in Health Informatics." International Journal of Medical Informatics 58–59 (2000): 257–289.

2. Bates, et al. "A Proposal for Electronic Medical Records in U.S. Primary Care." JAMIA 10 (2000): 1-10.

3. Computer Economics, 1998 Information Systems Spending Report

4. Coile, et al. "The Digital Transformation of Health Care." Physician Executive 26 (2000): 8-14.

5. McIntosh, M. "The Cost of Healthcare to Americans." JONA's Healthcare Law, Ethics and Regulations (2003) 4(3).

6. Medical Economics, March 11, 1996 v73 n5 p63(4), Health care: the sector that just keeps on booming. (Interview).

7. Noffsinger, Richard. "Harnessing Technology for Better Patient Care." Health Management Technology January (2000): 63-64.

8. Grossman, Jerome H. "Plugged-In Medicine." Technology Review (1997) Jan94 97(1): 22-30.

9. Weiner, Michael, et al., "Contrasting Views of Physicians and Nurses about an Inpatient Computer-based Provider Order-entry System." JAMIA (1999) 6: 234-244.

10. Buckovich, Suzy, et al., "Driving Towards Guiding Principles." JAMIA (1999) 6: 122-133.

11. Bilchik, Gloria Shur. "The IT Culture: How's Yours Coming?" Hospitals & Health Networks January (2003) 77(1): 32-36.

12. Schadow, Gunther. "Units of Measure in Clinical Information Systems." JAMIA (1999) 6: 151-162.

Appendix A: Source Code

To run this source code, copy every line and paste into a simple text editor, such as Microsoft Notepad. Save as Parser.java and with the appropriate PATHs set, compile and run. Make sure to maximize the window for full visibility.

```
// Scott Au
// Capstone Project:  HL7 Core Parser
// Spring 2003

import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Parser extends JFrame
{
  private Box mainbox;

  public Parser (String title) /*Setup the main Frame screen (full screen)*/
  {
    /* Set up the Main Window (JFrame) */
    setTitle(title);
    mainbox = new Box();
    getContentPane().add (mainbox);  /* mainbox provides all the interior components */
    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(mainbox.top, BorderLayout.NORTH); /*Add the top buttons +
text box*/
    getContentPane().add(mainbox.sp, BorderLayout.CENTER); /*Add the scrollable text
box*/
  }

  public static void main (String args[])
  {
    Parser parser = new Parser ("HL7 Parser");

    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize(); /*get the
maxscreen size*/
    parser.setBounds(0,0,screenSize.width, screenSize.height); /*start frame at upperleft*/
    parser.setVisible(true);

    parser.addWindowListener(new WindowAdapter() /*Action: User closes window*/
 {
  public void windowClosing (WindowEvent e)
    {
     System.exit(0);
    }
  } );
}

public class Box extends JPanel
implements ActionListener, ItemListener
 {
  /* The contents of the main JFrame include the following: */
  /* (1) JPanel at the NORTH to hold the filename box + buttons */
  /* (1) JPanel at the CENTER to hold the JScrollPane */
  /* The JScrollPane focuses on a JTextField */

  /* The function buttons */
  private JButton dump;
  private JButton patient;
  private JButton event;
  private JButton pharmacy;

  private JTextArea txt; /* Text area for results displaying */
  private JTextField input; /* Text area for entering file name */
  public JPanel top; /* Top components include buttons + filename area */
  private String query; /* holds filename */
  public JScrollPane sp; /* Scroller contains the JTextArea */

public Box()
{
 /* Setup the top area */
 top = new JPanel (); /* Initialize the top Jpanel */
 top.setBackground (Color.blue);
 input = new JTextField(15); /* Initialize the filename text area */
 input.setEditable(true);
 input.addActionListener (this); /* Add an ActionListener to filename text area */
 top.add(input); /* Add filename text area to top Jpanel */

 /* Create the fuction buttons */
 dump = new JButton ("File Dump"); /* File Dump button */
 dump.addActionListener (this);
 patient = new JButton ("Patient Name"); /* Patient Name button */
 patient.addActionListener (new PID());
 event = new JButton ("Event Type"); /* Event Type button */
 event.addActionListener (new EVN());
```

```
   pharmacy = new JButton ("Pharmacy Administration"); /* Pharmacy Administration
button */
   pharmacy.addActionListener (new RXA());

   /* Add the buttons to the top Jpanel */
   top.add(dump);
   top.add(patient);
   top.add(event);
   top.add(pharmacy);

   /* Create the reporting text area and scroller */
   txt = new JTextArea(300, 100);
   txt.setEditable(false);
   txt.setFont(new Font("Times New Roman", Font.BOLD, 16));
   sp = new JScrollPane();
   sp.setViewportView(txt);
   sp.setViewportBorder(BorderFactory.createLineBorder(Color.gray, 5));
   sp.setPreferredSize(new Dimension(300,100));
  }

  public void actionPerformed (ActionEvent e)
  {
   // Entering the filename
   query = input.getText();
   InputStreamReader input = new InputStreamReader(System.in);
   BufferedReader reader = new BufferedReader(input);

   // Reading the file
   try
   {
    if (query.equals("quit"))
    {
     System.exit(0);
    }

    FileReader fr = new FileReader ( query );
    BufferedReader fr2 = new BufferedReader(fr);

    /* First char for holding read chars */
    /* Second char for comparison to first */
    char ch = 'a';
    char ch2 = 'b';

    /* Need int due to BufferedReader returning int */
    /* Int is converted to char in reading steps */
    int inputchar = '1';
```

```
/* Denote new results display */
txt.append("\n");
txt.append("****************File Dump****************");
txt.append("\n");

// Read segments until END OF FILE
while ((inputchar = fr2.read()) != -1)
{
 ch = (char)inputchar;

 if (ch != '|')
    {
     if (ch != '^')
     {
      if (ch != '&')
      {
       String str = "" + ch;
             txt.append(str);
             System.out.print(ch);
    }
     }
    }


 /* If char = | delimiter, leave blank and carriage return */
 if (ch == '|')
 {

        /* A check is made for sequent | delimiters to prevent numerous blank lines,
aesthetic

issue */
  if (ch != ch2)
  {
   txt.append("\n");
       System.out.print("\n");
  }
 }

/* If char = ^ delimiter, add 5 blank spaces and carriage return */
if (ch == '^')
{
 txt.append("\n");
 txt.append("     ");
     System.out.print("\n");
```

```
          System.out.print("     ");
      }

      /* If char = & delimiter, add 10 blank spaces and carriage return */
      if (ch == '&')
      {
        txt.append("\n");
          txt.append("          ");
          System.out.print("\n");
          System.out.print("          ");
      }

      /* ch2 keeps track of the previous character */
      /* this allows comparison to prevent numerous */
      /* blank lines from being shown.  Aesthetic concern */
      ch2 = ch;

    } //End while
  } //End try

  catch(Exception e2){}

} //End ActionPerformed


class PID implements ActionListener {

  public void actionPerformed(ActionEvent e) {

  // Entering the filename
  query = input.getText();
  InputStreamReader input = new InputStreamReader(System.in);
  BufferedReader reader = new BufferedReader(input);

  // Reading the file
  try
  {
   if (query.equals("quit"))
   {
    System.exit(0);
   }

   FileReader fr = new FileReader ( query );
   BufferedReader fr2 = new BufferedReader(fr);

   /* Three chars hold the segment headers */
```

```
char ch = 'a';
char ch2 = 'b';
char ch3 = 'c';

/* Need int due to BufferedReader returning int */
/* Int is converted to char in reading steps */
int inputchar = '1';

/* Found variable determines if EVN segment present */
int found = 0;

/* Denote new results display */
txt.append("\n");
txt.append("****************Patient Name*****************");
txt.append("\n");

/* Search for PID segment */
while ((inputchar = fr2.read()) != -1)
{
    ch = (char)inputchar;
    if (ch == (char)0x0d)
    {
      fr2.skip(1);
  inputchar = fr2.read();
  ch = (char)inputchar;
      inputchar = fr2.read();
      ch2 = (char)inputchar;
      inputchar = fr2.read();
      ch3 = (char)inputchar;
      String str = "" + ch + ch2 + ch3;

  /* Search for 5th field which contains patient name */
      if (str.equals("PID"))
  {
   found = 1;
      txt.append("Patient Name: ");
      int dcount = 0;
      while (dcount < 5)
   {
        inputchar = fr2.read();
        ch = (char)inputchar;
        if (ch == '|')
     {
          dcount = dcount + 1;
        }
      }
```

```
        /* Read the contents of 5th field until | delimiter */
        inputchar = fr2.read();
           ch = (char)inputchar;
           while (ch != '|')
        {
             str = "" + ch;
             txt.append(str);
             ch = (char)(inputchar = fr2.read());
           }
      }
     }
    } //End while

    if (found == 0)
    {
     txt.append("No Patient Information found");
    }

  } //End Try

  catch(Exception e2){ }

  } //End ActionPerformed
 } //End PID

class EVN implements ActionListener
{
  public void actionPerformed(ActionEvent e)
  {

   // Entering the filename
   query = input.getText();
   InputStreamReader input = new InputStreamReader(System.in);
   BufferedReader reader = new BufferedReader(input);

   // Reading the file
   try
   {
    if (query.equals("quit"))
       {
     System.exit(0);
    }

    FileReader fr = new FileReader ( query );
    BufferedReader fr2 = new BufferedReader(fr);
```

```
/* Three chars hold the segment headers */
char ch = 'a';
char ch2 = 'b';
char ch3 = 'c';

/* Need int due to BufferedReader returning int */
/* Int is converted to char in reading steps */
int inputchar = 1;

/* Found variable determines if EVN segment present */
int found = 0;

/* Denote new results display */
txt.append("\n");
txt.append("****************Event Information*****************");
txt.append("\n");

while ((inputchar = fr2.read()) != -1)
{
    ch = (char)inputchar;
  if (ch == (char)0x0d)
    {
     fr2.skip(1);
     inputchar = fr2.read();
     ch = (char)inputchar;
     inputchar = fr2.read();
     ch2 = (char)inputchar;
     inputchar = fr2.read();
     ch3 = (char)inputchar;
     String str = "" + ch + ch2 + ch3;

   /* Report all data fields of the Event segment */
     /* Note that the Fieldprint function returns a */
   /* char.  This is prevent the function from reading */
   /* past the end of the segment (denoted by 0x0d) */
     if (str.equals("EVN"))
     {
      found = 1;
      fr2.skip(1); /* This skips the first | delimiter after the segment header */

    txt.append("EVENT TYPE CODE:  ");
      ch = Fieldprint(fr2, ch);

      txt.append("DATE/TIME OF EVENT:  ");
      ch = Fieldprint(fr2, ch);
```

```
            txt.append("DATE/TIME PLANNED EVENT:  ");
            ch = Fieldprint(fr2, ch);

          txt.append("EVENT REASON CODE:  ");
            ch = Fieldprint(fr2, ch);

            txt.append("OPERATOR ID:  ");
            ch = Fieldprint(fr2, ch);

            txt.append("EVENT OCCURRED:  ");
            ch = Fieldprint(fr2, ch);
           }
          }
      } //End while

      if (found == 0)
      {
          txt.append("No Event information found");
      }

    } //End try

    catch(Exception e2){ }

  } //End ActionPerformed
} //End EVN

class RXA implements ActionListener
{
 public void actionPerformed(ActionEvent e)
 {

   // Entering the filename
   query = input.getText();
   InputStreamReader input = new InputStreamReader(System.in);
   BufferedReader reader = new BufferedReader(input);

   // Reading the file
   try
   {
    if (query.equals("quit"))
    {
     System.exit(0);
      }
   FileReader fr = new FileReader ( query );
   BufferedReader fr2 = new BufferedReader(fr);

/* Three chars hold the segment headers */
   char ch = 'a';
   char ch2 = 'b';
   char ch3 = 'c';

/* Need int due to BufferedReader returning int */
/* Int is converted to char in reading steps */
   int inputchar = 1;

/* Found variable determines if EVN segment present */
int found = 0;

/* Denote new results display */
txt.append("\n");
txt.append("****************Pharmacy Administration****************");
txt.append("\n");

   while ((inputchar = fr2.read()) != -1)
{
    ch = (char)inputchar;

    if (ch == (char)0x0d)
  {
      fr2.skip(1);
      inputchar = fr2.read();
      ch = (char)inputchar;
      inputchar = fr2.read();
      ch2 = (char)inputchar;
      inputchar = fr2.read();
      ch3 = (char)inputchar;
      String str = "" + ch + ch2 + ch3;

      /* Report all data fields of Pharmacy Administration segment */
      /* Note that the Fieldprint function returns a */
   /* char.  This is prevent the function from reading */
   /* past the end of the segment (denoted by 0x0d) */
      if (str.equals("RXA"))
   {
    found = 1;
       fr2.skip(1); /* This skips the first | delimiter after the segment header */

       txt.append("GIVE SUB-ID COUNTER:  ");
       ch = Fieldprint(fr2, ch);
```

```
            txt.append("SUBSTANCE MANUFACTURER NAME:  ");
            ch = Fieldprint(fr2, ch);

            txt.append("SUBSTANCE REFUSAL REASON:  ");
            ch = Fieldprint(fr2, ch);

            txt.append("INDICATION:  ");
            ch = Fieldprint(fr2, ch);

            txt.append("COMPLETTION STATUS:  ");
            ch = Fieldprint(fr2, ch);

            txt.append("ACTION CODE:  ");
            ch = Fieldprint(fr2, ch);

            txt.append("SYSTEM ENTRY DATE/TIME:  ");
            ch = Fieldprint(fr2, ch);
          }
        }
      } //End while

    if (found == 0)
    {
        txt.append("No Event information found");
    }
      } //End try

      catch(Exception e2){ }

  } //End ActionPerformed
} //End RXA


public void itemStateChanged (ItemEvent e)
{
  dump = (JButton)e.getItem();
  patient = (JButton)e.getItem();
}


/* This function was made to simply the process of */
/* reading the data fields of selected segments */
public char Fieldprint (BufferedReader fr3, char ch)
{
 int inputchar = 1;
 String str = "";
```

```
      if (ch != (char)0x0d)
      {
       try
       {
            ch = (char)(inputchar = fr3.read());

         if (ch != '|')
            {
             while (ch != '|' && ch != (char)0x0d)
           {
              str = "" + ch;
              txt.append(str);
              ch = (char)(inputchar = fr3.read());
             }

         txt.append("\n");
            }

            else
         {
             txt.append("\n");
            }

       } //End try

       catch(Exception e2){ }
      }

     else
     {
      txt.append("\n");
     }

     return ch;

    } //End Fieldprint


   } //End Box
   } //End MainFrame
```

Appendix B: Sample HL7 message

Copy the following lines into a simple text editor such as Microsoft Notepad and turn off word wrapping. Each line should terminate immediately before the next segment header (shown in bold here to clarify). Note that this information is a composite of other HL7 sample messages and includes no real clinical information.

MSH|^~\&|EPIC|EPICADT|SMS|SMSADT|199912271408|CHARRIS|ADT^A04|18174
57|D|2.3|
EVN|A04|199912271408|||CHARRIS
PID||0493575^^^2^ID 1|454721||DOE^JOHN^^^^|DOE^JOHN^^^^|19480203|M||B|254
E238ST^^EUCLID^OH^44123^USA||(216)731-
4359|||M|NON|400003403~1129086|999-|
NK1||CONROY^MARI^^^^|SPO||(216)731-4359||EC|||||||||||||||||||||||||||
RXA|0|1|19900607|19900607|08^HEPB-
PEDIATRIC/ADOLESCENT^CVX^90744^HEPB-PEDATRIC
/ADOLESCENT^CPT|.5|ML^^ISO+||03^HISTORICAL INFORMATION - FROM
PARENT=S WRITTEN RECORD^NIP0001|^JONES^LISA|^^^CHILDREN=S
HOSPITAL||5|MCG^^ISO+|MRK12345|199206|MSD ^MERCK^MVX
PV1||O|168   ~219~C~PMA^^^^^^^^^^||||277^ALLEN FADZL^BONNIE^^^^||||||||||
||2688684|||||||||||||||||||||||||199912271408|||||002376853

Appendix C: Requirements document

The following is a basic requirements document that helped me to elucidate the final details of my program.

Project scope:

The Health Level 7 messaging standard has gained wide spread adoption in the health care information field. Many vendors have incorporated it as their means for allowing disparate healthcare applications to exchange clinical and administrative data. The creation of software for HL7 manipulation will further its adoption which will ultimately benefit medicine and research.

Objective:

To write an HL7 parser along with basic functions that would be needed by those investigators beginning to test the waters. I want to create an alternative to the commercial packages which are often too expensive and complex for amateurs. The same complexity is found many current and terminated open-source projects. Some require compiling source code while others require settings paths and running MakeFiles. Revisions to these projects can occur at inconsistent rates and documentation is not always comprehensive or accurate due to the multitude of volunteers.

Resources:

SourceForge.Net and other sites on the Internet allow me to see the projects of other users. Their work as well as the HL7 v2.4 standard itself (available from www.hl7.org) should provide ample examples and information. I also refer to the book Just Java 2 by Peter van der Linden and Sun Microsystem's online Java resources.

Preliminary plan:

Week 1: Review HL7 standard (470+ pages!) and the project sites of other HL7 developers.

Week 2: Make initial designs for the interface (GUI, command line, or both). Start to re-familiarize myself with Java since it has been a year since my last classes with Susan Price.

Week 3: Start thinking about the basic algorithm approach I will need to parse HL7 documents. This may get hairy due to the possible ways HL7 can be customized. May need to sacrifice some compatibility for greater general purpose.

Week 4: Start coding the parser and begin coding the basic GUI layout.

Week 5: Continue week 4's work and if progress is adequate, test the output of the parser. Start to consider what useful routines could be added (such as search functions).

Week 6: Continue adding routines. Debug any problems with the parser code, the GUI code, and the linkage between the two.

Week 7: Use this week to finalize the Parser code given the risk that I may not have progressed as far as desired. I am quite sure I will have problems with escape characters and repeatability of data fields.

Week 8: Use this week to finalize the GUI code since I am unsure of how to graphically present the as of yet undecided additional functions.

Week 9: Troubleshoot any last minute problems, clean up the code, and document everything. This includes source code as well as the program itself.

Week 10: Present my work.

User Parameters:

This project targets individuals with exposure to healthcare and information technology. The most likely users will be those just delving into HL7 as they may be turned off by the cost of commercial packages or the necessary technical skills to access the larger open-source projects. Java compilers are easily accessible and it would take very little work to make my basic program run. The number of these individuals is difficult to count as many may investigate on their own time as a personal pursuit.

Users would most likely use the program either at work, home, or school. Systems will probably be Wintel platforms given the predominance of these machines. System requirements will be no different from having a Java capable machine.

The lifetime of this project could be quite short given the push by the HL7 committee to ratify v3.0. This version incorporates XML technology and is seen as another milestone in HL7 technology. Movement to v3.0 would seriously undercut the usage of my project but it is currently inappropriate to try and write a parser in v3.0 due to the length of this class as well as the incomplete v3.0 standard.

I do not currently plan to support this program mostly due to the basic coding scheme that any persons wishing to add or modify should have no serious problems with.

I plan to make all code and documentation available online, either through SourceForge.Net or through a personal Web page.

Feature list:

A core parser for HL7 messages conforming to version 2.4. Additional functions include routines such as specific searching. The software components will be Java source code. My previous work in Java include the Java programming classes with Susan Price, MD at the Oregon Health & Science University as well as a general programming background in C, C++, and Pascal.