intelliware.ca
business application development

Always lead.™
www.intelliware.ca

HL7 Experience Report

by BC Holmes, Intelliware Development

Created on: **November 15, 2007**

Last Updated: **November 15, 2007**

Purpose

Recently, during a Standards Collaborative we heard several people bemoan the lack of experience reports from the vendor community. As vendors working on a number of eHealth applications, we thought that we could contribute back to this community by producing an experience report about a recent eHealth implementation.

What follows is a recap of our most recent eHealth project.

Background: The Project

We became involved in the project in the summer of 2007. We had initial conversations with the client mid-summer, and kicked off the project in earnest in mid-August. Our first code assets were created and checked in to our code repository on August 22nd.

Our client was especially interested in rapid delivery of first capability. Our initial target delivery date for release 1.0 was September 30th.

Release 1.1 has recently been dropped to a staging environment, with an intended production date of December 10th.

Coding has begun on Release 2.0 of the application.

The Application

We were asked to build a simple web-based application that could gather information about a care referral. The application would then format that referral data into an HL7 message which would be sent to a central (existing) HL7 messaging infrastructure (a HIAL server).

The application could additionally receive referrals from the HIAL server, and present that referral information users of the application. (Email notifications also ensured that the users would notice the new referral arriving).

Release 1.0 implemented the following features:

- Userid and password management
- Database encryption of sensitive information
- Create referral, edit and submit
  - data: referring practitioner, referred client/patient, patient contact, family physician, diagnosis, concern and consent

- Search for referrals
- HL7 version 2.4 (using XML) marshalling of one message type ("Create new referral")
- Additional message tier infrastructure including digital signing of the HL7 message, and wrapping the message in a custom XML envelope
- transmit of the HL7 via SOAP/web services
- receive an HL7 message via SOAP/web services

Release 1.1 implemented the following features:

- HL7 version 3.0 marshalling of two message types ("Create new referral" and "Acknowledge message")
- additional referral data (prognosis, additional contacts, special diet, prognosis, etc.)
- defaulting of data, and reuse of data (such as “refer the same client I referred yesterday”)

What Worked Well

1. Consistency of HL7 Concepts Across Different Applications

We were only able to accomplish the delivery of the application in the short time frames because we were already familiar with HL7 and had code assets and frameworks that could handle the construction and marshalling of HL7 messages. In a sense, we incurred the learning curve of HL7 once, in another project, and could bring that knowledge to bear on a completely different project.

Certain concepts -- common HL7 data types, code sets, and even the structure of Transport Wrapper around Control Act around Payload -- were immediately recognizable to us, even though we were dealing with them in a completely different problem space. The breadth of the standardization process is therefore useful.

While we may have reservations about the complexity of HL7, the idea of this kind of standard for health care is clearly a benefit.

We also found that, with few exceptions, the message specification contained a very complete set of concepts relating to our problem domain. Physicians, concerns, diagnoses, prognoses, etc., were already described in the message specification. Clearly the exercise in defining the message elements was sufficiently thorough.

2. HL7 Code Framework

As mentioned, we developed an HL7 messaging framework in previous projects and were able to reuse that framework in this new problem domain. We found that the framework worked surprisingly well with only minor enhancement required.

We were able to use:

- pre-defined HL7 types (Addresses, Telecoms, Instance Identifiers, etc.)
- domain value type information, and code resolution strategies
- code generation tools
- marshalling and unmarshalling tools

Because this framework allowed us to do bidirectional parsing and formatting of the XML messages, we employed a good round-trip testing strategy whereby we could compose and format a message, then turn around and parse and analyze the results. This kind of round-tripping for testing purposes helped build our confidence that the message processing was thorough.

3. Code Generation

In previous projects, we adopted a strategy of generating code for various messages and message elements from MIFs, although for the purposes of this exercise we needed to generate code from XSDs. Code generation is, to our mind, the most effective way of dealing with the richness of the messages. We were pleased to see that the strategy still seemed like the right choice.

4. Oids

We really buy in to the idea of Oids, and the global namespace that Oids enable.

5. Descriptiveness of HL7 Version 3

This project gave us the opportunity to use both HL7 version 2.4/XML and version 3.0 on the same project. It quickly became clear during integration testing and debugging that the version 3.0 messages were far more comprehensible. The longer names for the elements make it easier to understand what different parts of individual messages contain.

What Didn't Work Well

1. HL7 Version 3 Has a Lot of Unnecessary Complexity

There are several ways in which HL7 version 3 is, in our opinion, overly complex. This complexity serves to make applications harder and more expensive to implement. Now, we're vendors: we make money off of hard things to implement, so one would think we'd be happy with complexity. Nonetheless, we think that the health care industry, in general, would be better served by simpler standards.

Among the complicating factors are:

1. HL7's insistence on implementation independence, to the extent that it defines its own notions of "String" and "boolean". The HL7 definition of these types don't map well to common types in most modern languages such as Java or C#. Since XML is such an important component of HL7 version 3, it would have made much more sense to adopt the XML types (including string and boolean); most languages have already done their own language to XML mappings.

2. The various null flavours add very little business value, in our opinion, and make perfect implementations more complicated than necessary

3. Several of the types have strange implementations. For example, the Address type allows for fine-grained specification of individual address parts, such as "street direction" and "unit designator", whereas the Telecom type is surprisingly coarse-grained in comparison.

4. There's something about the message definition process that produces messages that includes about three times as many "types" as seems to really be necessary. Meaningful message elements are often wrapped in relatively useless types (often called "Component3" or equally unhelpful names). Our code generation process has done some work to simplify the code representations of these messages (while maintaining fidelity to the XML representation) because otherwise the classes are stunningly complicated. In our opinion, this complication only leads to developer confusion, and therefore longer implementation cycles.

5. The existing HL7 documentation is not terribly helpful. That it values abstractness over concreteness makes it more difficult to follow. More specificity, more brevity and more examples are called for, in our opinions.

6. Finally, there are many "levels" to the standards. It was hard to tease out which parts were HL7 standards, versus Canadian Standards Collaborative standards versus client-specific requirements. An easier mechanism for articulating which players had responsibility for which parts of the overall message specifications would be greatly appreciated. What's more, some choices looked to us to be deviations from the standards as we understood them, and it was hard to validate those suspicions.

Going forward, we believe, strongly, in the value of, say, an HL7 version 3.1 with a simpler set of data types, a more concise XML representation, and a clearer set of names.

2. XSD versus MIF

In past projects, we've been able to rely on the existence of MIF files to define the messages. We used the MIFs to generate our code assets, and we used a lot of information in the MIFs to optimize the code we ended up producing. Things like references to the domain of a coded value or even business names had a great deal of use to us.

This client had abandoned the MIFs, and relied solely on XSDs. Changes to the message definitions were manually applied to the XSDs. We were able to adapt our framework to use the XSDs in place of the MIFs, with some loss of information.

During the Standards Collaborative conference, we heard that there was a push to declare MIFs as the authoritative source for message definitions, and we agree with that push.

3. There Were Some Business Concept Misalignments

These are relatively small in the grand scheme of the project, but some of the business concepts that were familiar to our clients didn't correlate nicely with the message definitions. We're not certain if this is a glitch in the message server implementation, the Standards Collaborative process, or an HL7 process.

The main concept that didn't align well was the concept of a precaution. SARS was a motivating example: if a referral was made for a SARS-infected person, a precaution (gloves, mask) should accompany the referral.

4. Message Transport

Again, HL7's "generic" approach to discussing message transport requirements has led us into an environment that caused us some grief. We wanted to believe that certain features could be expected of a HIAL implementation -- for example, store and forward. The absence of this capability has tremendous implications for an applications such as ours.

Similarly, there were a lot of custom requirements for the SOAP transport that were fussy and hard to test, and consumed more time than seemed necessary, especially given how straight-forward most SOAP environments are. We would welcome more guidelines from the Standards Collaborative group in this area.

Things That Would Have Made Our Lives Easier

1. More Examples

Our primary reference was a 340-page message specification document that, despite its thoroughness, was still difficult to digest and make sense of. In addition to that specification document, we had to consult XSDs, HL7 documents about type definitions, domain values, and message element rendering as XML.

We would have found our jobs easier if there were more examples, including XML messages (or XML representations of individual message elements) linked to sample scenarios. These kinds of assets better support unit testing, and would have been easier for us to realize value from.

2. An Accessible Integration Server

Integration with the messaging server was easily the hardest part of the exercise. This integration was orders of magnitude harder than integration with, say, the PEI Test Harness. It would have been helpful to have an accessible messaging server that supported integration testing and tracking down integration problems. If message transport could be standardized, then it would be possible to have an integration environment as a shared resource for multiple eHealth projects.

3. Clearer Documentation

We think that the various existing documents would benefit from:

- brevity over thoroughness
- concreteness over abstractness
- examples in place of concepts