HBO & Company

CORBA Based
HL7 Implementation Approaches
CorbaMed RFI 5 Response

November 14, 1997

David A. Schramm
Director, HBOC Application Framework
587 East State Road 434
Longwood, Florida 32750
dschramm@ipworld.com

Introduction

The requirement for interaction between various parts of the healthcare delivery system is intensifying in volume, timeliness, and complexity. Our software solutions must mirror these business requirements. Therefore, HBO & Company believes strongly in moving away from message based interfacing of applications to object based integration of application components. We are participating heavily in the OMG Business Object Facility DTF (BODTF) both evaluating and influencing the Business Object Facility responses and responding to CBO RFIs and RFPs.

While HBO & Company supports the direction to base all business objects on the forthcoming OMG standard meta-models being established in the BODTF, we are using a proprietary business object meta-model and application framework today as the basis for our component development. An introduction to that meta-model can be found in OMG document bom/97-12-15. It is our desire to evolve that meta-model and our application framework to conform to the emerging OMG standards. Until that happens however, all HBO & Company responses will assume the use of our proprietary Application Framework.

HBO & Company recognizes the SIGOBT effort in the HL7 standards efforts. However, we are concerned that this effort is focusing primarily on substituting one message structure and transport protocol for another, while providing little improvement in the ability to deliver component based products. We are therefore concerned about the value of any standards arising from it. However, while moving to true component based solutions, we recognize that interactions with legacy applications still need to be based upon HL7 messages. This response discusses how we are bridging between these technologies.

- Performance experiences in using CORBA based HL7 implementations (what has worked well, what has not?).
- Benefits and drawbacks (pros and cons) to implementation of CORBA based HL7 implementations (why would someone want to build or buy a CORBA based HL7 implementation over a message based 'traditional' implementation).

We have two concerns about a CORBA based HL7 implementation.

1. If the objects are of comparable complexity to current HL7 Version 2 messages, we are concerned about performance ofORB marshalling and unmarshalling of these massive objects.

2. Message based interfacing of applications supports only a relatively simple level of interaction between large grained components. The requirements of the Healthcare industry already greatly exceed the capability of this paradigm and call for more complex interactions between medium grained components with higher performance characteristics.

Overview

The HBO & Company HL7 Facility consists of four major parts which provide end-to-end handling of HL7 messages into and out of our Application Framework based solutions. They are:

- a HL7 message definition compiler,
- a HL7 message type class library,
- an abstract form component type, and
- a HL7 form implementation.

HL7 Message Definition Compiler

- Automated approaches to transformation of HL7 ASCII encoded messages to CORBA 'objects' (for example, use of interface engine technology to manage transformation).

This section describes a HL7 definition compiler. It takes a textual definition of a HL7 V2.x message as input and creates a class library representing that message. Developers can modify / introduce message definitions, segment definitions, composite data type definitions and table definitions in the corresponding text files to accommodate their special needs.

Message definitions are defined in a message definition file which is used by the compiler to generate the tree structure of the messages. A user can either modify a pre-defined message or define her/his own message, as long as the new message is syntactically correct. In practice, we don't include optional groups/segments that are not used by the application. When populating itself with an ASCII HL7 message, the message class is able to pick up only the groups/segments it consists of. This approach avoids the trouble that those optional groups/segments, which are not supported by the receiving application, might cause, as well as improves performance.

The textual definition of a HL7 message is divided into four definition files; message definition, segment definition, composite data type definition and table definition. The following table contains an example of a message definition file.

```
                                        "A01"
MSH EVN PID<NK1>PV1[PV2]<OBX><AL1><DG1><PR1><GT1><IN1[IN2][IN3]>[ACC][UB1][UB2]
//
                                        "A06"
MSH EVN PID[MRG]<NK1>PV1[PV2]<OBX><AL1><DG1><PR1><GT1><IN1[IN2][IN3]>[ACC][UB1][UB2]
//
                                        "RDE"
MSH<NTE>[PID<NTE><AL1>[PV1]]{ORC[RXO<NTE>{RXR}[{RXC}<NTE>]]RXE{RXR}<RXC><[OBX]<NTE>>}
//
                                        "A17"
MSH EVN PID PV1[PV2]<OBX>PID PV1 PV2<OBX>
//
                                        "DFT"
MSH EVN PID[PV1][PV2]<OBX>{FT1}
//
                                        "RRE"
MSH MSA[ERR]<NTE>[[PID<NTE>]{ORC[RXE{RXR}<RXC>]}]
//
```

A segment definition file describes the segment's field length, data type, optionality, repetition number, and what HL7 tables are used and where they are used. The following table contains an example. A usage flag is used for turning off unused fields in the C++ class library.

```
   Y;  15; ST;  ;    ;                    ;sending application
   Y;  20; ST;  ;    ;                    ;sending facility
   Y;  30; ST;  ;    ;                    ;receiving application
   Y;  30; ST;  ;    ;                    ;receiving facility
   Y;  26; TS;  ;    ;                    ;date/time of message
   N;  40; ST;  ;    ;                    ;security
   Y;   7; ZM; R;    ;0076(1,1)|0003(0,2);message type
   Y;  20; ST; R;    ;                    ;message control id
   Y;   1; ID; R;    ;0103(1,1)           ;processing id
   Y;   8; ID; R;    ;0104(1,1)           ;version id
   Y;  15; NM;  ;    ;                    ;sequence number
   N; 180; ST;  ;    ;                    ;continuation pointer
   Y;   2; ID;  ;    ;0155(1,1)           ;accept acknowledgment type
   Y;   2; ID;  ;    ;0155(1,1)           ;application acknowledgment type
   Y;   2; ID;  ;    ;                    ;country code
```

A composite data type file describes each component data type and its name. Generic data type CM can be avoided by introducing new composite data types. Following is the text file for AD type.

```
ST _street;                             //street address
ST _other;                              //other destination
ST _city;                               //city
ST _state;                              //state or province
ST _zip;                                //zip
ST _count;                              //country
ID _type;                               //type
ST _geogr;                              //other geographic designation
```

There is a text file for each HL7 table. Only those tables that have HL7 suggested entries are pre-edited. They could be modified and new tables could be added.

The following is an example of the text file for HL7 table 0004:

```
  "B", // obstetrics
  "E", // emergency
  "I", // inpatient
  "O", // outpatient
  "P", // pre-admit
  "R", // recurring patient
```

Our implementation of the C++ classes parallels the CORBA IDL definition developed by HL7 Object Brokering SIG. One of the major differences is that we use a list for each field, each segment and each group, no matter what its requirement and repetition are. This implementation yields the freedom to specify its requirement and repetition using two integers. More importantly, it gives application programmers the convenience to create an empty object and fill in data later on.

HL7 Message Type Class Library

- Mappings which have been used to implement CORBA based HL7 solutions (could be in the form of IDL with textual explanation).

This section describes the lightweight C++ class library built by the HL7 Message Definition Compiler. The C++ class library serves as a C/C++ API to ASCII HL7 messages. It could be used alone as an API, or used as intermediate data structure for converting between ASCII HL7 messages and the CboForms. The system automatically generates the appropriate complex CboForm structure to contain the message.

We take a little different approach than that of the HL7 Object Brokering SIG in dealing with the data slots. We use a dynamic array for each data slot, no matter if it is optional, repeating or required. This approach simplifies the user interface of the library, and it treats data members uniformly.

This library is automatically generated. Different versions of HL7 and other HL7 variants, such as HBO & Company's HBOCHI standard, as well as customized messages and segments (Z message/Z segment) can be easily supported simultaneously. The functionality it offers includes:

- Parsing a stream message
- Building a message
- Validating the segment pattern of an incoming/outgoing message
- Validating the format of each field against HL7 encoding rule
- Validating those fields that use the tabular values for both incoming/outgoing messages

In HL7, a message is comprised of a group of segments in a defined sequence. The C++ class of a message in terms of segments is implemented as a tree structure with segments as leaves and groups as internal nodes. Figure 1 below shows the tree structure (without leaves) of a Pharmacy Encoded Order RDE message.

MSH<NTE>[PID<NTE><AL1>[PV1]]{ORC[RXO<NTE>{RXR}[{RXC}<NTE>]]RXE{RXR}<RXC><[OBX]<NTE>>}

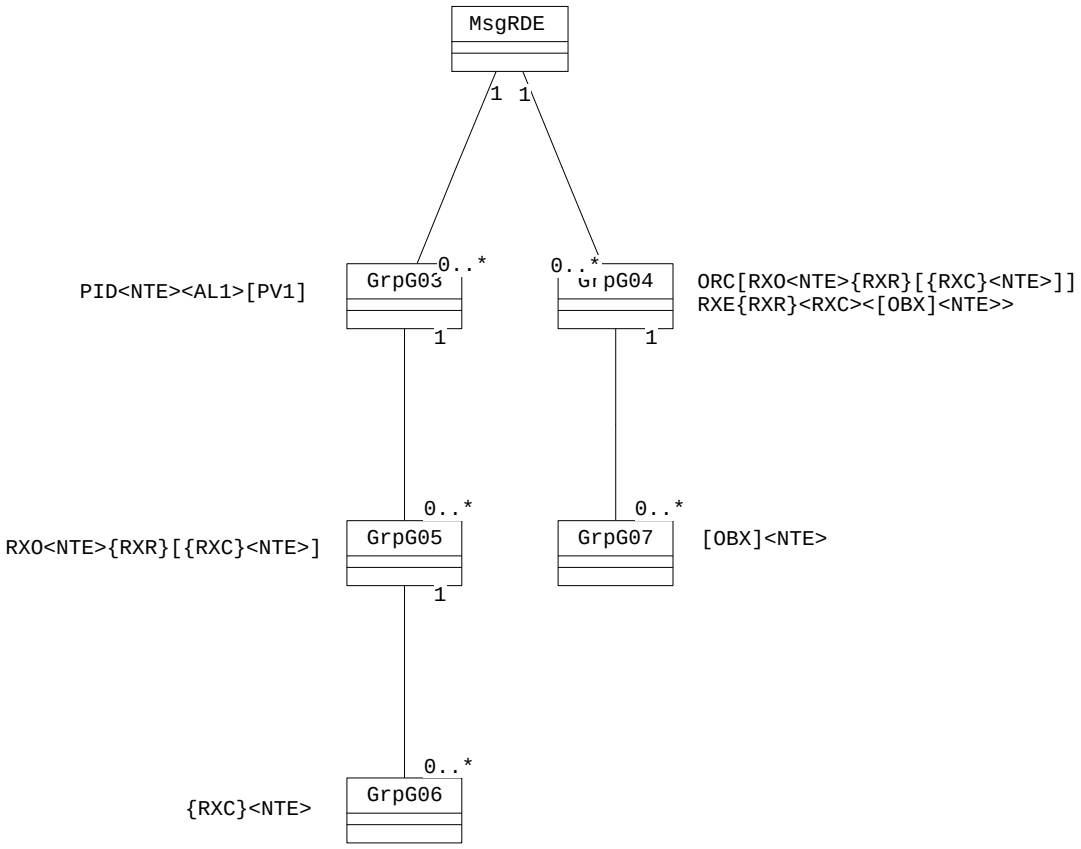

Figure 11

Each HL7 table is implemented as a C++ class that is associated with the HL7 message on the segment level; i.e., its instance is used in the segment node.

HL7 Forms

An overview of the abstract framework component types is presented in OMG document bom/97-12-15. One of the types outlined in that document is a CboForm. Forms are responsible for exchanging data into and out of the HBO & Company Application Framework. CboForms can only contain CboAttributes and CboForms. This abstract type provides all the interfaces used by other component types to send and receive HL7 messages.

HL7 Form Implementations

When a business transaction or process uses the form's Get() or Put() methods, the abstract form methods ask a form implementation factory for a HL7 form implementation based upon execution context information. This section briefly describes the architecture of the HBOC Application Framework HL7 form implementation. The concentration is on each step that the data format changes.

We shall discuss this issue in three categories:

- Inbound
- Outbound
- Real Time Query / Response

In all three cases, generated Abstract Form provides the programming interface.

Inbound

The HL7 listening socket is implemented in a separate executable which publishes business event (CboEvent) objects. Business transactions subscribe to and process these events. The following figure shows the design:

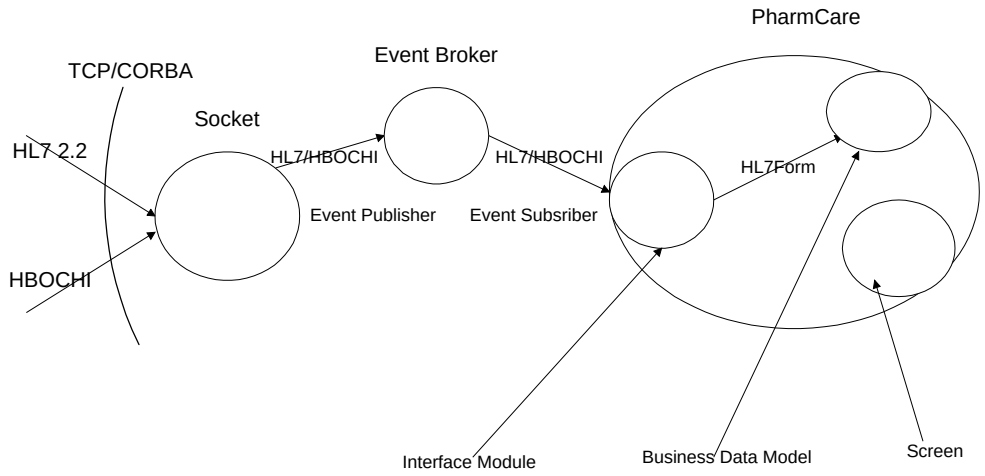

Figure 22

The listening application is responsible for:

- accepting messages from other systems,
- doing high level validation and sending ACK messages, and
- publishing the valid messages as business events.

Each HL7 message type is published as a separate business event subject. The business transaction must:

1. Subscribe to the event of interest.

2. Instanciate a generated HL7 Form for the published message. The type of the form depends only on the type of message, i.e., create a patient, and not on protocol standard, HL7 2.2 or HBOCHI 0.5.

3. Call the form's Get() method. The implementation of Get() accommodates different protocol standards. The generated HL7 Form performs the following operations.

    a) Instanciates a proper HL7 message structure (different versions or different mapping of the same version are handled here) and parses the message into its C++ structure (the message class in HL7lib) including lower level validation.

    b) Converts the message structure into the corresponding form structure, including converting between different encoding schemes and mapping between different versions of tables, i.e., one is used by HL7 and the other is used by the application.

    c) Validates the form's contents.

Outbound

Outbound messages are created by the business transaction. Following is the description of the architecture

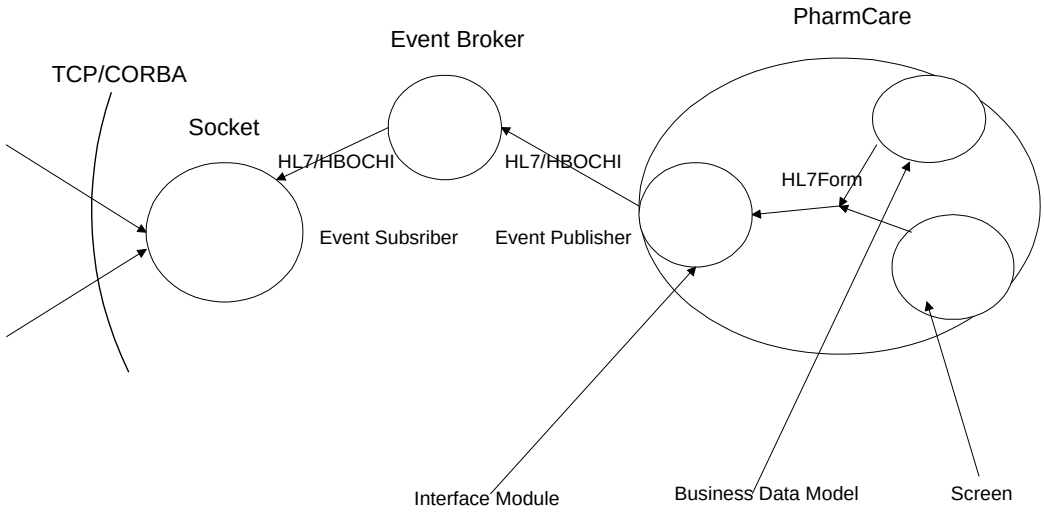

Figure 33

The form's Put() method is designed to format and send outbound messages. As in the inbound case, Put() is the function which deals with different protocols. The following is performed by the IMhl7Form implementation.

- Check the interface configuration table to see which version of message should be sent (HL7, HBOCHI, or both).

- Create corresponding HL7/HBOCHI class instance(s) and convert the form into the HL7/HBOCHI structure(s).

- Convert the HL7/HBOCHI structure(s) into character string format.

- Send the string message(s) to the proper system.

Real Time Query / Response

This interface is real time. The put method blocks until the response is received. The following is the description of the architecture:

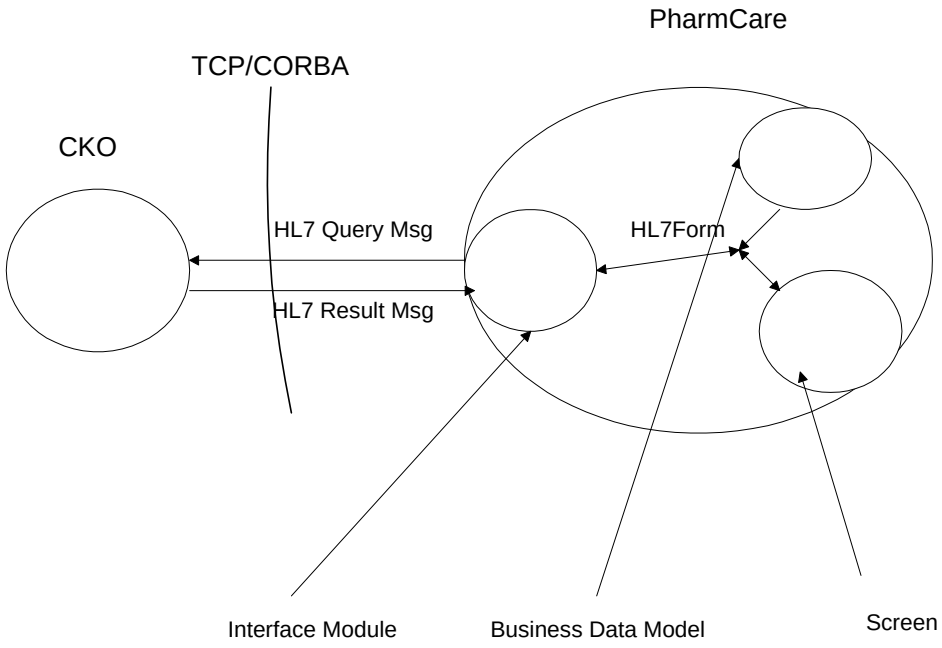

Figure 45

The CboForm::Put(Cboform &resultForm) method is designed for this purpose. The following is performed by the IMhl7Form implementation.

- Create the corresponding HL7 class instance and convert the form into the HL7 structure.

- Convert the HL7 structure into character string format.

- Call the function string HelloCKO (string queryMessage), which returns result message. The HelloCKO() function is responsible for communication to CKO, i.e., sending the query message and getting result message.

- Create the corresponding HL7 class instance for the result message and parsing the result message.

- Convert the HL7 structure into the result Form

Discussion

The following is a discussion of some additional topics requested in this RFI.

- Automated approaches to transformation of HL7 ASCII encoded messages to CORBA 'objects' (for example, use of interface engine technology to manage transformation).

We use an abstraction called Forms which perform all communications to and from non-object technologies including WWW Browsers, EDI protocols and HL7. Form objects have Get() and Put() methods to convert between themselves and external stream protocols such as HL7 messages. This provides the application programmer with only one object type to perform any communications of this type.

- Any object model examples which make use of the HL7 2.X specification or V3 model

The HL7 library is the implementation of a HL7 2.X specification model. The HL7Form implementation implements a sort of V3 model.

- Performance experiences in using CORBA based HL7 implementations (what has worked well, what has not?).

Since entire ASCII HL7 messages are exchanged as a blob at the communication level, this approach yields very good communication performance.

- Discussion of use of CORBA services in relation to the versioning concepts of HL7.

In order to support multiple versions/multiple mappings of the same version of HL7, we use two data models to bridge the ASCII HL7 messages and their corresponding object components. One is the HL7 Class Library, which represents the structure of HL7 message. The other is the Form, which is a subset of the message data relevant to a transaction. While only one Form is used for each HL7 message, various HL7 Class Library message structures are used to accommodate multiple versions and multiple mappings. Both of the structures and the C++ code to convert between them are automatically generated. This approach isolates the application from the interface issues, and offers much more manageable coding.

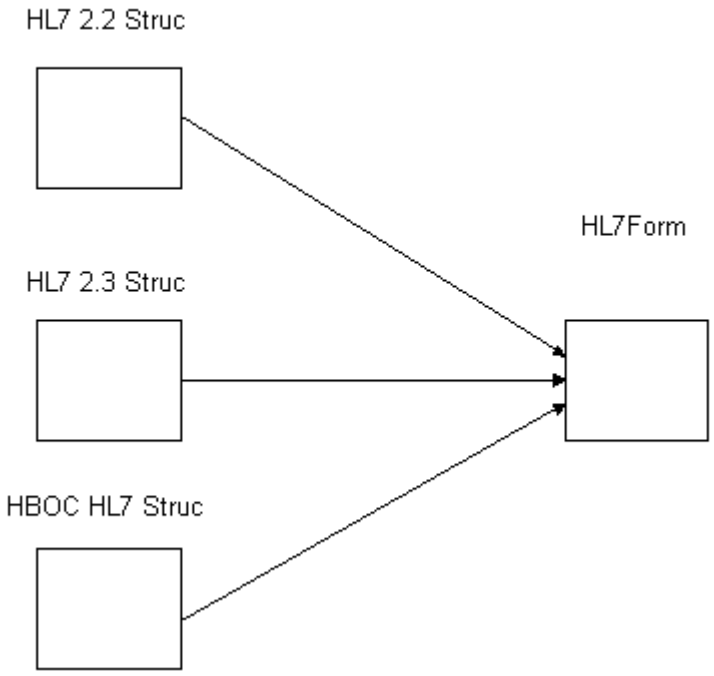

Figure 56

- Discussion of how greater interoperability was achieved through the approaches described.

1. Since ASCII HL7 messages are exchanged, this approach could be used to integrate with any application that supports HL7.

2. Since different versions/flavors of HL7 could be supported at the same time, it can talk to applications speaking different 'dialects' of HL7.

3. The overall framework supports the flexibility for dynamically change the execution environment enabling the interface to be changed quickly and conveniently.

Appendix A

Information Being Requested

This response does not address the following topics.

- Solutions developed in the healthcare arena that take advantage of general CORBA services such as naming, security, etc.
- Descriptions of relationships between HL7 trigger events and other CORBA services.
-