재사용성과 확장성있는 HL7 인코딩/디코딩 프레임워크의 설계 및 구현

(Design and Implementation of a Reusable and Extensible HL7 Encoding/Decoding Framework)

김 정 선 [†] 박 승 훈 [††] 나 연 묵 [†††]
(Jungsun Kim) (Seung-Hun Park) (Yunmook Nah)

요 약 HL7 (Health Level 7)은 Healthcare 환경의 이질적 시스템간에 임상 및 관리정보의 교환을 가능하게 하는 국제 표준 프로토콜로서 표준 인코딩 규칙에 따른 다양한 HL7 메시지 양식을 정의하고 있다.
본 논문에서는 메시지 객체 모델(Message Object Model)과 메시지 정의 저장소(Message Definition Repository)를 이용하여 유연성, 재사용성, 확장성이 탁월한 HL7 인코딩/디코딩 프레임워크의 설계 및 구현을 제시한다. 메시지 객체 모델은 HL7 메시지를 구성하는 객체들과 그들 간의 다양한 관계를 나타내는 추상적 HL7 메시지 양식으로서, 세그먼트, 필드, 컴포넌트 등과 같은 HL7 메시지의 표준 구성요소들 간의 논리적 관계를 반영하는 동시에 표준안에 의해 규정된 구조적 제약사항을 만족하도록 하여 준다. 메시지 객체 모델은 플랫폼 종속적인 데이터 양식과 상관 없이 독립적으로 HL7 인코더와 디코더를 구축할 수 있도록 하여 주기 때문에 최소의 노력으로 임의의 이질적 병원 정보 시스템들을 상호 연결할 수 있도록 한다. 한편, HL7 메시지들을 정의하고 있는 외부 데이터베이스인 메시지 정의 저장소는 표준 HL7 메시지 양식이 수정되더라도 인코더와 디코더의 구현이 영향은 받지 않게 하여 준다. 게다가, 메시지 정의 저장소는 인코더와 디코더 각각의 입력(즉, 메시지 객체 모델로 표현된 HL7 메시지 객체와 인코딩된 HL7 메시지 문자열)에 대하여 합법성 여부를 조사하는 데 유용하게 사용된다.
본 논문에서는 프로토타입 HL7 인코더와 디코더의 구현을 위해 JAVA를 이용하였지만, 제시된 인코딩/디코딩 프레임워크는 인코더와 디코더를 ActiveX, JAVABEAN 또는 CORBA 객체 등과 같이 독립된 표준 컴포넌트로서 쉽게 구현될 수 있도록 하여 준다.
키워드 : HL7, 병원정보시스템, 메시지 객체 모델, 인코더/디코더, 프레임워크

Abstract The Health Level Seven (HL7), an international standard for electronic data exchange in all healthcare environments, enables disparate computer applications to exchange key sets of clinical and administrative information. Above all, it defines the standard HL7 message formats prescribed by the standard encoding rules.
In this paper, we propose a flexible, reusable, and extensible HL7 encoding and decoding framework using a Message Object Model (MOM) and Message Definition Repository (MDR). The MOM provides an abstract HL7 message form represented by a group of objects and their relationships. It reflects logical relationships among the standard HL7 message elements such as segments, fields, and components, while enforcing the key structural constraints imposed by the standard. Since the MOM completely eliminates the dependency of the HL7 encoder and decoder on platform-specific data formats, it makes it possible to build the encoder and decoder as reusable standalone software components, enabling the interconnection of arbitrary heterogeneous hospital information systems (HISs) with little effort. Moreover, the MDR, an external database of key definitions for HL7 messages, helps make the encoder and decoder as resilient as possible to future modifications of the

† 종신회원 : 한양대학교 컴퓨터공학과 교수
jskim@cse.hanyang.ac.kr
†† 비 회 원 : 경희대학교 전자정보학부 교수
parksh@khu.ac.kr
††† 종신회원 : 단국대학교 공학부 컴퓨터공학전공 교수
ymnah@dankook.ac.kr
논문접수 : 2001년 5월 8일
심사완료 : 2001년 10월 13일

standard HL7 message formats. It is also used by the encoder and decoder to perform a well-formedness check for their respective inputs (i.e., HL7 message objects expressed in the MOM and encoded HL7 message strings).

Although we implemented a prototype version of the encoder and decoder using JAVA, they can be easily packaged and delivered as standalone components using the standard component frameworks like ActiveX, JAVABEAN, or CORBA component.

Key words : HL7, Hospital Information Systems, Message Object Model, Encoder/Decoder, Frameworks

1. Introduction

The Health Level Seven (HL7) [1], an international standard for electronic data exchange in all healthcare environments, enables disparate computer applications to exchange key sets of clinical and administrative information. Above all, it defines the standard message formats prescribed by the standard encoding rules. Therefore, the HL7 encoder and decoder are considered to be the integral parts of future hospital information systems (HISs). Specifically, the encoder generates a stream of HL7 messages from specialized data sets of a source HIS, and the decoder is responsible for extracting information from the received HL7 message stream for a target HIS.

Since the standard defines only the formats and legitimate encoding rules for the HL7 message types, there exist many ways to implement the encoder and decoder. However, as shown in Figure 1, a typical implementation of the encoder and decoder may cause a tremendous waste of time and effort since they cannot be reused across heterogeneous HISs. The reason for this is that despite their inherent functions are invariant, they must be implemented for each HIS system that uses its own specific data formats. If data formats on either side are changed or replaced, the corresponding encoder and/or decoder must also be properly adjusted. This is considered as a serious maintenance problem. The problem lies in the fact that the encoder and decoder are tightly coupled with site-specific data formats that may be different from one HIS to another. To overcome this problem, the encoder and decoder must be designed to be independent of platform-specific data formats.

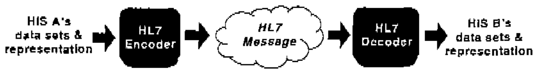

HIS A's data sets & representation
HL7 Encoder
HL7 Message
HL7 Decoder
HIS B's data sets & representation

Fig. 1 A typical HL7 encoding and decoding scheme

In this paper, we propose a novel HL7 encoding/decoding framework that makes it possible to build a flexible, reusable, and extensible HL7 encoder and decoder. We also present a JAVA [2] implementation of a prototype encoder and decoder using this framework.

The framework is characterized by two salient features, called a Message Object Model (MOM) and Message Definition Repository (MDR).

First, the MOM provides an abstract HL7 message form represented by a group of objects and their relationships. It reflects logical relationships among the standard HL7 message elements such as segments, fields, and components, while enforcing the key structural constraints imposed by the standard. Since the MOM completely eliminates the dependency of the HL7 encoder and decoder on platform-specific data formats, it makes it possible to build the encoder and decoder as reusable standalone software components, enabling the interconnection of arbitrary heterogeneous HISs with little effort. The idea of representing HL7 messages as objects is not new [3]. However, we focus our attention on the simplicity of the model which nevertheless can not only reflect the structure of HL7 messages correctly, but also enforce the key structural constraints imposed by the standard.

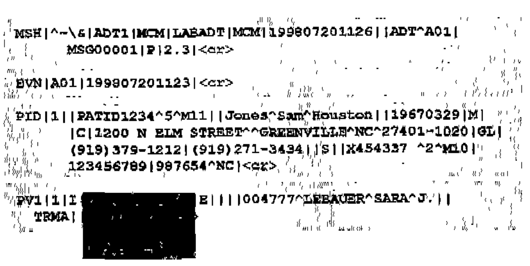
Second, the MDR, an external database of the key definitions for HL7 messages, helps make the encoder and decoder as resilient as possible to future modifications of the standard HL7 message formats. It is also used by the encoder and decoder to perform a well-formedness check for their respective inputs (i.e., HL7 message objects and encoded HL7 message strings).

This paper is organized as follows. In the following section, we briefly review the standard HL7 message formats. In section 3, the MOM is described. In section 4, the MOM-based HL7 encoding/decoding framework is presented. A JAVA implementation of the HL7 encoder and decoder is briefly explained in section 5. Finally, concluding remarks are given in section 6.

2. Brief Overview of the HL7 Message Formats

A *message* is the smallest unit of HL7 data transferred between two communicating parties. It is comprised of a group of *segments* in a defined sequence. A *segment* is a logical grouping of data *fields*. Segments of a message may be required or optional. They may occur only once in a message or they may be allowed to repeat. A *field* is a string of characters which may be required or optional. Like segments, fields of a segment may occur only once in a segment or they may be allowed to repeat. A field may consist of a group of *components* in a defined sequence which may be required or optional. Finally, a *component* may consist of a group of *subcomponents* in a defined sequence which may be required or optional. This hierarchical message structure can represent very complex information very effectively. Figure 2 shows an example of HL7 message.

As shown in Figure 2, certain special characters are used in constructing a message. A segment is terminated by a segment terminator which is always a carriage return character. Fields, components, and subcomponents are delimited by a field separator ('|'), a component separator ('^'), and a subcomponent separator ('&'), respectively. For repeated fields in a segment or repeated components in a field, a repetition separator ('~') is used to separate them. All of the separator characters, except the segment terminator, are negotiable and configurable. The characters shown inside parentheses in the above sentences denote the default ones. Using these separators, a hierarchically structured message can be represented as an encoded character stream.



```
MSH|^~\&|ADT1|MCM|LABADT|MCM|199807201126||ADT^A01|
     MSG00001|P|2.3|<cr>

EVN|A01|199807201123|<cr>

PID|1||PATID1234^5^M11||Jones^Sam^Houston||19670329|M|
     |C|1200 N ELM STREET^^GREENVILLE^NC^27401-1020|GL|
     (919)379-1212|(919)271-3434||S||X454337 ^2^M10|
     123456789|987654^NC|<cr>

PV1|1|I[redacted]E||||004777^LEBAUER^SARA^J.||
  TRMA|[redacted]
```

Fig. 2 An example of HL7 message

3. The Object Model for the HL7 Messages

In this section, we propose an object model for the HL7 messages which we call the Message Object Model (MOM). In MOM, an HL7 message is represented as an object [4] (called *an HL7 message object*). An HL7 message object is an abstract form of HL7 message, which is used as a source for the encoder and at the same time as a target for the decoder, in the HL7 encoding/ decoding framework to be explained in the next section.

Figure 3 shows the class diagram for the MOM

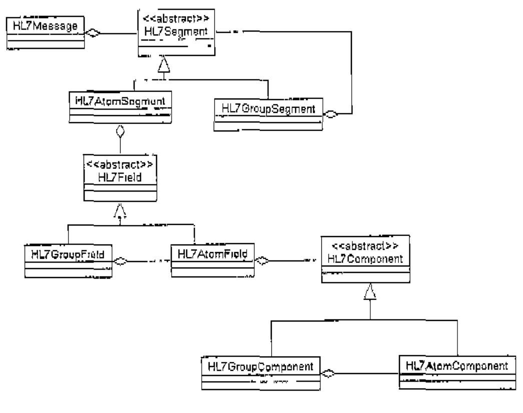


Fig. 3 A Message Object Model (MOM)

using the Unified Modeling Language (UML) [5], illustrating the classes that compose an HL7 message object and their various relationships. The organization of the classes reflects the logical relationships among the constituent elements of an HL7 message in a simple and elegant way, while enforcing certain structural constraints imposed by the HL7 standard. Therefore, any HL7 messages can be effectively represented using the MOM.

In the MOM, an HL7 message is modeled as an instance of the *HL7Message* class that provides an API for creating and setting values of various fields contained in it. Instances of the *HL7Message* class maintain a set of top-level *HL7Segment* objects. By top-level segments, we mean the segments that are not contained in another segment. Segment objects are either instances of the class *HL7AtomSegment* (i.e., atomic segments) or instances of the class *HL7GroupSegment* (i.e., group segments). The *HL7AtomSegment* is a logical segment which corresponds to a standard HL7 segment such as Message Header (MSH), Event Type (EVN), Patient ID (PID), Patient Visit (PV1) etc. The *HL7GroupSegment* is a structural segment which can contain a set of atomic segments and other group segments. A group of segments that must appear together in a defined sequence must always be contained in a group segment. Any repeatable segments must also be contained in a group segment. As you may have noticed, the standard GOF Composite Pattern [6] is applied here to model the relationship among the classes for segments.

Instances of the *HL7AtomSegment* class consist of a set of *HL7Field* objects. The field objects are either instances of the class *HL7AtomField* (i.e., atomic fields) or instances of the class *HL7Group Field* (i.e., group fields). The *HL7AtomField* is a logical field that corresponds to a standard HL7 field defined in its enclosing segment. Like *HL7GroupSegment*, the *HL7GroupField* is a structural element that can contain a group of field objects. However, unlike the *HL7GroupSegment*, the *HL7GroupField* can contain only the atomic field objects that are repeatable. It cannot contain any group field objects. This prohibits a recursive nesting of fields, as is specified by the standard. The relationship between a field and its components can be modeled the same way as the one between a segment and its fields.

Notice that there exist no classes defined in the MOM for the HL7 subcomponents. The reason for this is that subcomponents for a component can be safely modeled as a group of primitive components that are contained in a composite component. A primitive component is represented as an instance of the *HL7AtomComponent* class. A composite component is represented as an instance of the *HL7GroupComponent* class.

Beware that although the *HL7GroupField* class and *HL7GroupComponent* class have a similar naming pattern, their usage is different. While the former is used as a container for repeatable field objects, the latter is used a container for subcomponents.

Conceptually, the MOM is similar to the DOM (Document Object Model) [7] which represents the organization of an XML (eXtensible Markup Language) [8,9] document in an object-oriented way. Just as the DOM greatly facilitates the manipulation of XML documents, the MOM allows us to access clinical information more flexibly through a high-level API in various end-user applications.

4. A Flexible and Robust HL7 Encoding/ Decoding Framework

In this section, we describe a novel HL7 encoding and decoding framework that is flexible and robust. As shown in Figure 4, the framework is characterized by two salient features: the Message Object Model (MOM) and the Message Definitions Repository (MDR).

In this framework, the encoder and decoder use HL7 message objects as the source for encoding and as the target for decoding, respectively. Specifically, the encoder generates an encoded HL7 message string from a given HL7 message object, and the decoder constructs an HL7 message object

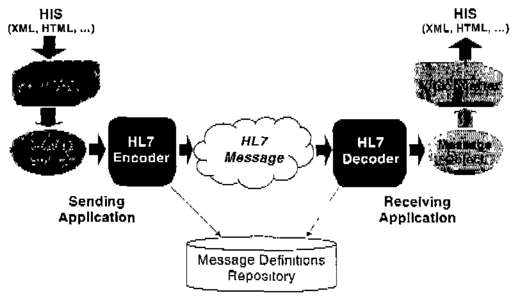


Fig. 4 A flexible and robust HL7 encoding/decoding framework

from a given encoded HL7 message string. This scheme completely eliminates the dependency of the encoder and decoder on the platform-specific data formats and representation. Now, since the encoder and decoder purely depend on abstract, data format-neutral HL7 message objects, their operations can be completely encapsulated and robustly implemented. This fact makes them very good candidates for reusable software components which may be implemented using the standard component frameworks such as ActiveX [10], JAVABEAN [11], or CORBA component [12].

In this framework, the encoder and decoder also make use of an external Message Definitions Repository (MDR). The MDR maintains key information about all HL7 messages like message type definitions, segment type definitions, field attributes, HL7 data type definitions, and etc. During an encoding process, the encoder consults this repository not only to interpret a given HL7 message object, but also to check the validity (i.e., well-formedness) of it. An HL7 message object is considered to be well-formed if and only if it does not violate any of the definitions prescribed in MDR. For example, an HL7 message object is considered to be ill-formed if any of the required elements are missing, or if it violates any of the structural constraints imposed by the MOM. Any invalid form of message objects are rejected. The repository is also used by the decoder for similar purposes. Consequently, the MDR makes the operations of the encoder and decoder be as resilient as possible to future modifications of the standard HL7 message formats.

Notice that, in our encoding/decoding framework, we may need converters to transform HL7 message objects to and from data sets represented in HIS specific data formats (e.g., XML-to-MOM/MOM-to-XML). What kind of converters to use is a decision that must be made per application basis. However, since the encoder and decoder are shielded from these gory details, it is now possible to plug-and-play any converters without affecting the encoder and decoder.

As compared with the Figure 2 showing a typical, non-reusable, and error-prone encoding/decoding scheme, our framework makes it possible to build the encoder and decoder that are not only data format independent and robust, but also flexible enough to accomodate plausible evolution of the HL7 message definitions in the future.

5. A Prototype Implementation of the HL7 Encoder and Decoder

In this section, we present a JAVA implementation of a prototype HL7 encoder and decoder system based on the HL7 standard 2.3.1 [1].

5.1 The Message Definitions Repository (MDR)

The Message Definition Repository (MDR) is implemented as simple text files each of which contains message type definitions, segment type definitions, field attribute definitions, and data type definitions, respectively. Listing 1 and 2 show the message definitions and data type definitions, respectively. For the sake of simplicity, the other two definitions are not presented here.

As shown in Listing 1, a message type, say ADT_A05, is defined as a sequence of segment type definitions that are separated by a bar character (|). The message type ADT_A05 corresponds to the *ADT-pre admit a patient (event A05)* in HL7. Message type definitions are terminated by a carriage return. In the ADT_A05, "MSH", "EVN", "PID", and "PV1" are all mandatory non-repeatable atomic segments. Optional non-

```
ADT_A01¦MSH¦EVN¦PID¦NK1OptGrp¦PV1¦PV2_O¦OBXOptGrp¦AL
1OptGrp¦DG1OptGrp¦PR1OptGrp¦GT1OptGrp¦InsOptGrp¦ACC_
O¦UB1_O¦UB2_O<CR>
ADT_A02¦MSH¦EVN¦PidGrp<CR>
ADT_A03¦MSH¦EVN¦PidGrp<CR>
ADT_A04¦MSH¦EVN¦PID¦NK1OptGrp¦PV1¦PV2_O¦OBXOptGrp¦AL
1OptGrp¦DG1OptGrp¦PR1OptGrp¦GT1OptGrp¦InsOptGrp¦ACC_
O¦UB1_O¦UB2_O<CR>
ADT_A05¦MSH¦EVN¦PID¦NK1OptGrp¦PV1¦PV2_O¦OBXOptGrp¦AL
1OptGrp¦DG1OptGrp¦PR1OptGrp¦GT1OptGrp¦InsOptGrp¦ACC_
O¦UB1_O¦UB2_O<CR>
...
```

Listing 1 HL7 Message Definitions

repeatable atomic segment types have names with a suffix "_O" attached. For example, "PV2_O", "ACC_O", "UB1_O", and "UB2_O" represent optional non-repeatable segments. The segment types that have names ending with "Grp" such as "NK1OptGrp" and "InsOptGrp" denote repeatable optional segments.

```
EVN¦R¦A¦=<CR>
IN1¦R¦A¦=<CR>
IN2_O¦O¦A¦IN2<CR>
IN3_O¦O¦A¦IN3<CR>
MSH¦R¦A¦=<CR>
NK1¦R¦A¦=<CR>
PID¦R¦A¦=<CR>
PV1¦R¦A¦=<CR>
PV2_O¦O¦A¦PV2<CR>
NK1OptGrp¦O¦G¦NK1<CR>
InsOptGrp¦O¦G¦IN1¦IN2_O¦IN3_O<CR>
...
```

Listing 2 HL7 Segment Definitions

As shown in Listing 2, segment type definitions are also terminated by a carriage return. Each segment type is defined as below.

Segment Type Name | Optionality | Atomicity | { Base Segment Type Name }

The *"Segment Type Name"* is a name that appears in the message type definitions. The *"Optionality"* field is marked either 'O' (optional segment) or 'R' (required segment). The *"Atomicity"* field is marked either 'A' (atomic segment) or 'G' (group segment that contains repeatable segments). For mandatory atomic segments, the last field is marked by '='. For an optional atomic segment, the last field must denote the name of its base segment type. For example, the base segment type for "PV2_O" is just "PV2". For a group segment, a sequence of constituent segment type definitions must appear in the last field.

As mentioned before, in addition to the message and segment type definitions, our system also maintains two definition tables as part of the MDR: one for the attributes of HL7 fields that compose HL7 segments, the other for the HL7 data type definitions. All these definitions are collectively used by the encoder (decoder) to check the validity the HL7 message object (encoded message string) and subsequently to generate well-formed encoded HL7 message string (HL7 message object).

5.2 Operations of the HL7 Encoder/Decoder

An encoder, an instance of the *HL7Encoder* class, accepts as its input an HL7 message object (i.e., an *HL7Message* object) and generates an encoded HL7 message string. When the encoder's *encode()* method is invoked, it first creates a message definition object (i.e, an instance of the *HL7MsgDefinition* class), using the Message Definition Repository (MDR), that corresponds to the triggering event type of the message, say ADT A05, and then subsequently constructs an encoded message string. During the encoding process, the encoder strictly checks the validity of the HL7 message object by consulting the *HL7Msg Definition*. Therefore, any invalid HL7 message objects are automatically rejected.

An decoder, an instance of the *HL7Decoder*, accepts as its input an encoded HL7 message string and constructs a corresponding HL7 message object. Like the encoder, the decoder also creates a message definition object first and then uses it during its decoding process to construct a well-formed HL7 message object, checking the validity of the encoded message string. Therefore, any invalid input strings are rejected.

The encoder can generate an encoded HL7 message string either in a compact or verbose form. Both are valid forms that are permitted in the HL7 standard. In an encoded message string, while all the trailing empty elements are suppressed in a compact form, all constituent elements appear in the verbose form regardless of whether they are empty or not. Listing 3 and 4 show the output of the encoder in verbose form and in compact form, respectively. The decoder can handle both forms of encoded HL7 message strings. The encoder and decoder also provide methods for configuring various separators.

```
MSH¦^~\&¦REGADT^^¦MCM^^¦IFENG^^¦¦199601061000^¦¦ADT^A05
¦000001¦P^¦2.3¦¦¦¦¦¦¦
EVN¦A05¦199601061000^¦199601101400^¦01¦¦199601061000^
PID¦¦¦191919^^^GENHOSP&&^^¦253763^^^^^¦MASSIE^JAMES^A^^
^^¦¦19560129^¦M¦¦¦171
ZOBERLEIN^^ISHPEMING^MI^49849^""^^^^¦¦(900)485-5344^^^^
^^^^¦(900)485-5344^^^^^^^^¦¦S¦C¦10199925^^^^^¦371-66-92
56¦¦¦¦¦¦¦¦¦¦¦
NK1¦1¦MASSIE^ELLEN^^^^^¦SPOUSE^^^^^¦171
ZOBERLEIN^^ISHPEMING^MI^49849^""^^^^¦(900)485-5344^^^^^
^^^¦(900)545-1234^^^^^^^^~(900)545-1200^^^^^^^^¦EC^EMER
GENCY CONTACT^^^^¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
NK1¦2¦MASSIE^MARYLOU^^^^^¦MOTHER^^^^^¦300
ZOBERLEIN^^ISHPEMING^MI^49849^""^^^^¦(900)485-5344^^^^^
^^^¦(900)545-1234^^^^^^^^~(900)545-1200^^^^^^^^¦EC^EMER
GENCY CONTACT^^^^¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
NK1¦3¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
PV1¦¦O¦¦¦¦¦0148^ADDISON,
JAMES^^^^^^^^^^^^¦0148^ADDISON,
JAMES^^^^^^^^^^^^¦0148^ADDISON,JAMES^^^^^^^^^^^^¦AMB¦¦¦
¦¦¦¦¦0148^ADDISON,JAMES¦S^^^^^¦1400^¦A¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
ENHOP¦¦¦¦¦¦¦¦¦¦¦¦¦¦
```

Listing 3 A verbose form of an encoded message string

```
MSH¦^~\&¦REGADT¦MCM¦IFENG¦¦199601061000¦¦ADT^A05¦000001¦
P¦2.3
EVN¦A05¦199601061000¦199601101400¦01¦¦199601061000
PID¦¦¦191919^^^GENHOSP¦253763¦MASSIE^JAMES^A¦¦19560129¦M
¦¦¦171 ZOBERLEIN^^ISHPEMING^MI^49849^""¦¦(900)485-5344¦
(900)485-5344¦¦S¦C¦10199925¦371-66-9256
NK1¦1¦MASSIE^ELLEN¦SPOUSE¦171
ZOBERLEIN^^ISHPEMING^MI^49849^""¦(900)485-5344¦
(900)545-1234~(900)545-1200¦EC^EMERGENCY CONTACT
NK1¦2¦MASSIE^MARYLOU¦MOTHER¦300
ZOBERLEIN^^ISHPEMING^MI^49849^""¦(900)485-5344¦
(900)545-1234~(900)545-1200¦EC^EMERGENCY CONTACT
NK1¦3
PV1¦¦O¦¦¦¦¦0148^ADDISON, JAMES¦0148^ADDISON,
JAMES¦0148^ADDISON,JAMES¦AMB¦¦¦¦¦¦¦¦0148^ADDISON,JAMES¦S
¦1400¦A¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ENHOP
```

Listing 4 A compact form of an encoded message string

5.3 Creating HL7 Message Objects

In order to use the HL7 encoder, an HL7 message object (i.e. an *HL7Message* object) must be created first. An HL7 message object can be created using the MOM class library that is comprised of various classes shown in Figure 3. There are two different ways to create an HL7 message object using the MOM library. Whereas the first method explicitly makes use of the internal structure of the message object, the second method makes use of the high-level API provided by the MOM library. Thus the second method is more abstract and easier to use.

We will describe both methods briefly. In both methods, we will create an HL7 message object that maps to the following HL7 message string.

```
MSH|^~\&|||||||ADT^A05|000001|P|2.3<CR>
EVN|A05|199601061000|199601101400|||199601061000
<CR>
PID|||191919^^^GENHOSP||MASSIE^JAMES^A<CR>
NK1|1|MASSIE^ELLEN||||(900)545-1234~(900)545-
1200<CR>
NK1|2|MASSIE^MARYLOU|||(900)545-1234<CR>
PV1||O|||||0148^ADDISON, JAMES<CR>
```

The above message string is an example of the *ADT-preadmit a patient (event A05)*. Here, we will show how the PID and NK1 segments are created and added to an HL7 message object. All the segments except the NK1 are of the same nature in that they are all non-repeatable atomic segments. However, the NK1 segment is a repeatable segment in ADT A05 message type.

5.3.1 Method 1

In method 1, all the constituent elements of an HL7 message object are explicitly created using a factory object and then they are composed one by one using the low-level API of appropriate MOM classes. Listing 5 shows how an HL7 message object is created.

The first step to build an HL7 message object is to create a factory object (i.e., an instance of the *HL7Factory* class) that is responsible for allocating necessary MOM objects shown in Figure 4. At line 10, an empty HL7 message object (i.e. an instance

```
HL7AtomSegment aseg;
HL7GroupSegment          gseg, gingseg;
HL7AtomField  afield;
HL7GroupField gfield;
HL7AtomComponent         acomp;
HL7GroupComponent gcomp;

HL7Factory factory = new HL7Factory();

HL7Message msg = factory.createMessage("ADT", "A05");
...
aseg = factory.createSegment("PID");
  gfield = factory.createGroupField("Patient ID (Internal ID)");
     afield = factory.createField("Patient ID (Internal ID)");
         acomp = factory.createComponent("ID");
         acomp.setValue("191919");
         afield.add(acomp);
         gcomp = factory.createGroupComponent ("assigning authority");
            acomp = factory.createComponent ("namespace ID");
               acomp.setValue("GENHOSP");
            gcomp.add(acomp);
         afield.add(gcomp);
      gfield.add(afield);
   aseg.add(gfield);
  afield = factory.createField("Patient Name");
      acomp = factory.createComponent("family name");
         acomp.setValue("MASSIE");
      afield.add(acomp);
      acomp = factory.createComponent("given name");
         acomp.setValue("JAMES");
      afield.add(acomp);
      acomp = factory.createComponent("middle initial or name");
         acomp.setValue("A");
      afield.add(acomp);
   aseg.add(afield);
msg.add(aseg);
...
gseg = factory.createGroupSegment("NK1OptGrp");
   gingseg = factory.createGroupSegment();
       aseg = factory.createSegment("NK1");
          afield = factory.createField("Set ID - NK1");
             acomp = factory.createComponent();
                   acomp.setValue("1");
                afield.add(acomp),
          aseg.add(afield);
         gfield = factory.createGroupField("Name");
                afield = factory.createField();
                   acomp = factory.createComponent("family name");
                      acomp.setValue("MASSIE");
                   afield.add(acomp);
                   acomp = factory.createComponent("given name");
                      acomp.setValue("ELLEN");
                   afield.add(acomp);
                 gfield.add(afield);
              aseg.add(gfield),
              gfield = factory.createGroupField("Business Phone Number");
                 afield = factory.createField();
                    acomp = factory.createComponent("text");
                          acomp.setValue("(900)545-1234");
                    afield.add(acomp);
      gfield.add(afield);
                 afield = factory.createField();
                    acomp = factory.createComponent("text");
                       acomp.setValue("(900)545-1200");
                          afield.add(acomp);
                    gfield.add(afield);
          aseg.add(gfield);
       gingseg.add(aseg);
   gseg.add(gingseg);
    ...    // Creation of the second NK1 segment is omitted here
msg.add(gseg);

HL7Encoder encoder = new HL7Encoder();

String encString = encoder.encode(msg, true); // encode in vervose mode
...
```

Listing 5 (Continued)

of the *HL7Message* class) for a message type of "ADT" with a triggering event "A05" is created. Line 12 through 36 show how an atomic segment, named "PID", is created and then added to the HL7 message object created at line 10. In our example, the "PID" segment contains only two mandatory fields: a repeatable field (*"Patient ID (Internal ID)"*) and an atomic field (*"Patient Name"*). While line 13 through 24 show how a repeatable field is created and then added to a segment, line 25 through 35 show how an atomic field is created and then added to a segment.

Since the *"Patient ID (Internal ID)"* field is repeatable, it must be contained in an *HL7GroupField* object. Therefore, an *HL7Group Field* object is first created at line 13 and then an atomic *HL7AtomField* object is created at line 14. This *HL7AtomField* will eventually be added to the *HL7GroupField* object at line 23. Line 15 through 17 show how the primitive component named *"ID"* is created and then added to the *"Patient ID (Internal ID)"* field. Line 18 through 22 show how the composite component *"assigning authority"* is created and then added to the *"Patient ID (Internal ID)"* field. Unlike a primitive component, a composite component must first be contained in the *HL7GroupComponent* and then the *HL7Group Component* must be added to its enclosing field.

Line 38 through 71 show how a repeatable segment is created as a group and then added to the HL7 message object created at line 10. As you can see from Listing 5, any repeatable segments like "NK1" cannot be directly contained in an HL7 message object. It must first be contained in an *HL7Group Segment* object and the *HL7Group Segment* object must be added to an HL7 message object.

Even if the construction procedure for an HL7 message object using method 1 is intuitive, it is relatively complex and error-prone. Therefore we provide a more convenient mechanism to facilitate the creation of an HL7 message object, which is described next.

5.3.2 Method 2

In method 2, we do not use a factory object to create the MOM objects composing HL7 message objects. They are implicitly and automatically created, whenever necessary. Listing 6 shows how an HL7 object is created using this method.

At line 1, an empty HL7 message object (i.e. an instance of the *HL7Message* class) for the message type of "ADT" with a triggering event "A05" is created. Line 3 through 7 show how the

```
 1: HL7Message msg = new HL7Message("ADT", "A05");
 2: ...
 3. msg.setValue("PID:Patient ID (Internal ID):ID", "191919", 0);
 4: msg.setValue(":Patient ID (Internal ID):assigning authority:namespace ID", "GENHOSP", 0);
 5: msg.setValue(":Patient Name:family name", "MASSIE");
 6: msg.setValue("::given name", "JAMES");
 7: msg.setValue(":Patient Name:middle initial or name", "A");
 8:
 9: HL7GroupSegment group = msg.newGroup("NK1OptGrp");
10: HL7GroupSegment grp = group.newGroup();
11: grp.setValue("NK1:Set ID - NK1", "1");
12: grp.setValue("NK1:Name:family name", "MASSIE", 0);
13: grp.setValue("NK1:Name:given name", "ELLEN", 0);
14: grp.setValue("NK1:Business Phone Number:text", "(900)545-1234", 0);
15: grp.setValue("NK1:Business Phone Number:text", "(900)545-1200", 1).
16: grp = group.newGroup();
17: grp.setValue("NK1:Set ID - NK1", "2");
18: grp.setValue("NK1:Name:family name", "MASSIE", 0);
19: grp.setValue("NK1:Name:given name", "MARYLOU", 0);
20: grp.setValue("NK1:Phone Number:text", "(900)545-1234", 0);
21: ...
22: HL7Encoder encoder = new HL7Encoder();
23:
24: String encString = encoder.encode(msg, false); // cncode in compact mode
25:
...
```

Listing 6

values of fields for the non-repeatable segment "PID" is assigned. Line 9 through 20 show how the values of fields for the repeatable segment named "NK1" is assigned. Notice the difference between the ways in which values are assigned to the fields. The values of fields for a non-repeatable segment are assigned using the *setValue* function defined in the *HL7Message* class. On the other hand, those for a repeatable segment are assigned using the *setValue* function defined in the *HL7GroupSegment* class.

As shown in Listing 6, the key contributors in this method are the following overloaded functions.

public void setValue(String comp, String value);

public void setValue(String comp, String value, int index);

The *"setValue"* sets the value of the specified field in an HL7 message object. While the first function must be called to allocate or modify the value of a non-repeating field of an atomic segment object, the second function must be called to allocate or modify the value of each of the repeated field of an atomic segment object. In these functions, the first parameter *"comp"* denotes a path name designating a target primitive component object, and the second parameter *"value"* is a value to be assigned to the specified primitive component. In the second function, the third parameter *"index"* specifies a sequence number for the repeated field objects. The sequence number that starts from number 0, uniquely identifies those repeated fields in their enclosing segment. Note that, in any case. the values of fields can be set even before the corresponding fields (and, if necessary, their enclosing segments) exist. If you try to set the value of a non-existent, but valid field of a segment, the field (and, if necessary, its enclosing segment) is automatically created before the value is assigned.

The *"comp"* parameter needs a further explanation. Since an HL7 message object is comprised of hierarchically nested elements, we can specify a target component using a sequence of element names reflecting the nested hierarchy. For example, at line 3, the path name "PID:Patient ID (Internal ID):ID" denotes a target component in the sequence of segment name (PID), field name (Patient ID (Internal ID)), and component name (ID). While interpreting the path name, if any of the constituent elements in the path name appear for the first time, their corresponding MOM objects are automatically and implicitly created. At line 4, a segment name is omitted in the first parameter. In this case, the most recently used segment name (i.e. PID) is implicitly used. Likewise, at line 6, the path name "::given name" is the same as "PID:Patient Name:given name".

5.4 Accessing HL7 Message Objects

Just as there are two different ways to create an HL7 message object using the MOM library, there also exist two different ways for accessing an HL7 message object. The first method explicitly makes use of the internal structure of the message object and relies on the primitive API provided by the MOM classes. The second method makes use of the high-level API provided by the MOM library such as the overloaded *"getValue"* functions defined in the *HL7Message* and *HL7GroupSegment* classes. Since their usage pattern is similar to the *"setValue"*, we will not explain it here.

6. Conclusion

In this paper, we proposed a flexible, reusable, and robust HL7 encoding and decoding framework using a Message Object Model (MOM) and Message Definition Repository (MDR). As an object-oriented model reflecting the logical relationships among the standard HL7 message elements, the MOM completely eliminates the dependency of the HL7 encoder and decoder on platform-specific data formats. Therefore, it makes it possible to build the encoder and decoder as reusable standalone software components. Moreover, the MOM also provides a high-level API for creating and manipulating the HL7 message objects. In addition, the MDR, an external database of key definitions for HL7 messages, helps make the encoder and decoder resilient to future modifications of the standard HL7 message formats. It is also used by the encoder and decoder to perform a validity check for their respective inputs.

Although we implemented a prototype version of the encoder and decoder using JAVA, they can be easily packaged and delivered as standalone components using the standard component frameworks like ActiveX, JAVABEAN, or CORBA component.

We strongly believe that the proposed HL7 encoding/decoding framework can be effectively used as a critical architectural component to construct an integrated medical information system.

References

[1] Health Level Seven Standard version 2.3.1., www.hl7.org

[2] Arnold, K. and Gosling, J., The Java Programming Language, Sun Microsystems, Inc., 1996.

[3] Robert Seliger, SIGOBT Mapping HL7 Messages to Objects, Version 1.0, Revision B, August 14, 1996.

[4] Cox, B., Object-Oriented Programming: An Evolutionary Approach, Addison-Wesley, 1986.

[5] Booch, G., Rumbaugh, J., and Jacobson, I., The Unified Modeling Language User Guide, Addison-Wesley, 1999.

[6] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design Patterns: Elements of Object-Oriented Software. Addison-Wesley, 1995.

[7] Document Object Model (DOM), a working draft of the W3C, http://www.w3.org/DOM/

[8] Chang, D., and Harkey, D., Client/Server Data Access with Java and XML, John Wiley & Son's Inc., 1998.

[9] Charles, E., and Kahn, Jr., "Self-Documenting Structured Reports using Open Information Standards," Proceedings of 9th World Congress on Medical Informatics, IOS Press, pp. 403-407, 1998.

[10] Platt, D., The Essence of COM with Activex: A Programmer's Workbook, Prentice Hall, 2000.

[11] Vanhelsuwe, L., Mastering JavaBeans, SYBEX Inc., 1997.

[12] Siegel, J., CORBA 3 Fundamentals and Programming, 2nd Edition, John Wiley & Son's Inc., 2000.

김 정 선

1986년 서울대학교 컴퓨터공학과 졸업(학사). 1988년 Iowa Statr University 전기 및 컴퓨터 공학과 졸업(공학석사). 1994년 Iowa State University 전기 및 컴퓨터 공학과 졸업(공학박사). 1994년 ~ 1996년 한국전자통신연구원(ETRI) 선임연구원. 1996년 ~ 현재 한양대학교 컴퓨터공학과 조교수. 관심분야는 Parallel & Distributed Processing, Component Based Development, Distributed Object Computing.

박 승 훈

1981년 서울대학교 공과대학 전기공학과(학사). 1984년 서울대학교 공과대학 제어계측공학과(석사). 1990년 University of Florida at Gainesville, Electrical Eng. (박사). 1985년 ~ 1990년 한국전자통신연구소 ISDN연구부 연구원. 1991년 ~ 2000년 건국대학교 의용생체공학부 부교수. 2000년 ~ 현재 경희대학교 전자정보학부 교수. 관심분야는 의료정보시스템, 생체 신호 처리와 해석, 방사선 치료 시스템, 객체지향 시스템

나 연 묵

1986년 서울대학교 컴퓨터공학과 졸업(공학사). 1988년 서울대학교 대학원 컴퓨터공학과 졸업(공학석사). 1993년 서울대학교 대학원 컴퓨터공학과 졸업(공학박사). 1991년 미국 IBM T.J.Watson 연구소 객원연구원. 1993년 ~ 현재 단국대학교 공학부 컴퓨터공학 전공 부교수. 2001년 ~ 2002년 University of California, Irvine 객원교수. 관심 분야는 데이타베이스, 객체지향 데이타베이스, 데이타 모델링, 데이타베이스 설계, 멀티미디어 데이타베이스, 멀티미디어 정보 검색, 멀티미디어 시스템