Selecting Effective Test Messages

Len Gebase
Roch Bertucat
Robert Snelick
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD, USA

Abstract - *One of the first problems that must be addressed when attempting to test the reliability of any software implementation is determining what is to be tested. The number of paths an implementation can follow is generally much too large to be able to test all possibilities. Pragmatic decisions must be made that allow selective testing, but exhaustive testing covering all possible paths is not possible.*

In this paper we investigate an approach for finding sets of messages that provide effective coverage for testing HL7 applications. HL7 messages are constrained by an XML document that determines the form and content of the messages. We describe a method for determining one metric for measuring the set of messages that encompass all paths allowed by the XML document. In general this set of messages is also very large. Nevertheless, the metric provides us with a concrete assessment of the scope of the testing problem. Furthermore, we are able to formulate techniques for filtering this large message set and substantially reduce its size while maintaining the essential coverage capabilities inherent in the larger message set.

Keywords: Combinations; Conformance Testing; Filtering; Messaging Systems; Test Messages.

1 Introduction

In the Software Diagnostics and Conformance Testing Division [1] of the Information Technology Laboratory (ITL) at the National Institute of Standards and Technology (NIST) we are interested in conformance testing methodologies and their application to various areas of information technology. One area of current interest is their application to health care systems. We are currently investigating the Health Level 7 (HL7) messaging standard [2, 3], a widely used standard for moving clinical and administrative information between healthcare applications.

As part of our work with the HL7 healthcare messaging standard, we have developed a tool, *Message Maker* [4, 5], which supports the dynamic generation of HL7 messages. The messages are generated based on an *XML* document or profile. Message Maker provides a powerful means for generating test messages. The messages can be used to support the evaluation of HL7 implementations. Message Maker provides the user with considerable flexibility for controlling the set of test messages that are generated. The question remains, however, as to what set of messages the user should generate. The user can generate an unlimited set of messages, inject errors in messages, control specific elements of a message, as well as vary other message characteristics. But how many messages will be needed to adequately cover the scope of possibilities that an implementation should support? And is the size of this message set small enough to be employed in testing HL7 implementations?

Below we investigate one approach for finding effective test messages based on an examination of tree structures. We attempt to capture all paths allowed by the profile by examining the tree structures representing HL7 messages derivable from the profile. We determine the number of messages that can be generated, and once we have identified the message set, we take further steps to reduce its size, while attempting not to sacrifice the capabilities inherent in the original set.

2 Example Profile

Conformance was formally introduced into Version 2.5 [6] of the HL7 standard. An *Implementation Profile* was the primary mechanism introduced for supporting conformance. The Implementation Profile is an XML [7] document that provides a well-defined and rigorous specification of the set of messages that can be exchanged between HL7 applications. Using profiles, vendors can define precisely the set of messages that they will exchange. Each message derivable from the profile itself has an XML representation. Since any XML document can be represented as a tree, every HL7 message also has a tree representation.

Each element in the profile includes attributes that determine the element's presence, or absence, in the messages derived from the profile. Specifically, elements include a *Usage* attribute that determines if the element must be present, cannot be present, or might be present in a message. Elements also include *MAX* and *MIN* attributes

that determine the element's *cardinality* or the number of times the element can be repeated.

To be precise, the allowable values for the Usage attribute are *R*, *RE*, and *X*. Any element with Usage value of R, must be present, any element with a Usage value of X cannot be present, and any element with Usage value of RE, may be present.

The MAX and MIN attributes may take on any non-negative integer value. MIN must be less than MAX, but MIN is not required to be zero even for an element with a Usage value of RE. In this case, the element may not be required to appear, but if it does, it must appear more than once.

While a relatively large and complex tree is required to represent a typical profile, we will begin by considering an unrealistically small and simple tree. Despite its simplicity, the tree will serve as the bases for extending our work to more general cases. A snippet of our simple XML profile is shown in Figure 1.

```
<Segment Name="S1" Usage="R" Min="1" Max="2">
  <Field Name="F1" Usage="RE" Min="0" Max="3" >
    <Component Name="C1" Usage="RE" >
    </Component>
    <Component Name="C2" Usage="R" >
    </Component>
    <Component Name="C3" Usage="RE" >
    </Component>
  </Field>
  <Field Name="F2" Usage="RE" Min="0" Max="2" >
</Segment
```

Figure 1. Toy Profile.

The profile details are omitted; only the aspects of the profile relevant to our example are shown. A real profile would include additional XML elements, and the elements shown would include additional attributes affecting data content. The *Name* attribute is shown, however, and the profile structure is also revealed. All profiles contain the nesting structure shown: Field elements must be wrapped in *Segment* elements, *Component* elements must be wrapped in a *Field*, and *SubComponent* elements—not included in our example--may also be included within a Component element. No further nesting of elements is allowed however.

The semantics of our profile are relatively simple. The segment element, S1, must be present in all messages conforming to the profile and the segment may repeat once. The Field element, F1, may be repeated three times if present and F2 may be repeated twice. The element, F1, has three children, and the child Component, C2, must be present whenever F1 is present; the other children may optionally be present, but neither can be repeated. Component elements do not include either MIN or MAX attributes. However, if the elements did include these attributes, and if the values of the MIN and MAX attributes for the optional components were set to zero and one, respectively, the semantics would be equivalent to those shown without their presence. For the required component including the attributes with both values set to one would convey the same semantics. Noting the equivalence of the two representations enables us to use the same notation for representing all nodes in the tree. We will annotate each node in the tree with *R [m, n]* to designate a required node, or *RE [m, n]* to designate an optional node, that must repeat a minimum of m times and cannot repeat more than n times. Nodes with a *Usage* value of X are not allowed to appear in valid messages and therefore will not be further considered here.

The tree representation of our profile is show in Figure 2.

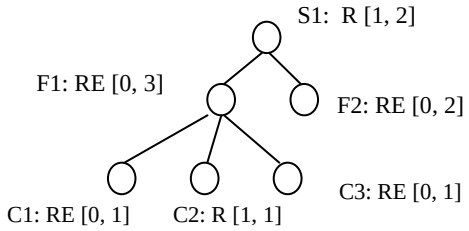

Figure 2. Tree Representation of Toy Profile.

The tree representation shown above does not represent an actual tree, but rather represents some number of possible trees. The members of this set of possible trees represent the class of conformant HL7 messages derivable from the profile shown in Figure 1 above.

To illustrate, Figure 3 shows two tree instances representing conforming messages.

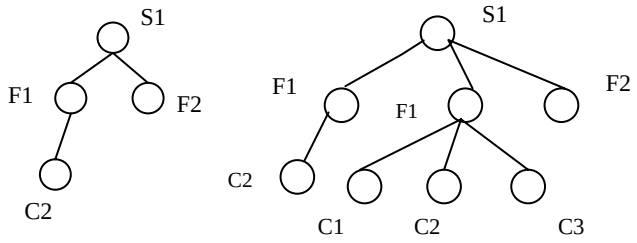

Figure 3. Tree Instances.

Next we enumerate a class of messages belonging to our toy profile.

3 Counting Messages

We want to determine the number of messages that belong to the class of well-formed messages derivable from

the profile shown in Figure 1. We will use the tree representation of this profile shown in Figure 2 to formulate a counting strategy. We are only interested in messages that are structurally distinct; variations based on content, or node values, will not be considered.

To count tree structures we will examine sub-tree variations. We will begin by considering nodes with only leaf nodes for children. We will refer to nodes in the tree as *node elements* to emphasize the relationship between the nodes in the tree and the elements in the XML document. Considering sub-trees rooted at node element, *F1*, it is clear that the total number of sub-trees that can be formed from a single occurrence of this node is exactly 4. We'll use a more succinct lisp-like notation than we used above to enumerate the 4 structures as follows: *(F1 (C1 C2 C3))*, *(F1 (C1 C2))*, *(F1 (C2 C3))*, and *(F1 (C2))*. This number can, of course, easily be derived mathematically by observing that node *C1* and *C3* can appear in one of two ways, i.e., they can either be present or not present, and node *C2* can only appear in one way. The total number of possibilities for the nodes appearing then is simply given by the product of the number of ways in which each node can appear. In general, for any sub-tree rooted at node element, *E*, where all of *E*'s children are leaves, the total number of sub-trees, S_E, that can be formed at *E*, is given by Equation 1.

Equation 1: $S_E = \prod_{i=1}^{N} C_i$ where N is the number of children of node E and C_i is the number ways in which the child node, C_i, can appear.

In the example, we have $S_{F1} = 2 \times 1 \times 2 = 4$.

In our example, node element *F1* can appear up to three times. What is the total number of sub-tree combinations, then, that can be formed at *F1* when the number of times F1 appears varies over the range of valid repetition values? We have shown that for one occurrence of *F1*, four sub-trees are possible. For two occurrences of *F1*, it is clear that there are 4 times 4, or 16, possible ways to combine the sub-trees, and for three occurrences there are 4 times 4 times 4, or 64 possibilities. The total number of sub-tree combinations then that can be formed at *F1* from all valid repetitions of *F1* is given by the sum of the number of combinations that can formed for each occurrence of *F1*. In general, for a node element *E* repeating from 1 to N times, the number of sub-tree combinations that can be formed at *E* is given by Equation 2.

Equation 2: $C_E = \sum_{k=1}^{N} (S_E)^k$ where S_E is the number of sub-trees that can be formed at node *E* for a single occurrence of node element *E* and N equals the maximum number of times that *E* can repeat.

At *F1*, we have $C_{F1} = 4^1 + 4^2 + 4^3 = 84$.

We note that since the node element *F1* is not required, there is one additional sub-tree combination, the empty sub-tree, which we should add to our count. This can be accounted for more generally, by noting that for a node element *E* repeating from 0 to N times, the above formula can be re-written as:

Equation 3: $C_E = \sum_{k=0}^{N} (S_E)^k$

Applying the above formula, then, at node *F1*, we have 85 sub-tree combinations that can be formed at *F1*. To proceed up the tree and determine the number of sub-tree combinations that can be constructed at *S1*, we note that if the sub-tree at *F1* were replaced by a single node that varied in 85 ways, then we could again apply Equation 1 above and get the total number of variations for a single occurrence of *S1* by taking the product of the variations at *F1* and *F2*. This observation allows us to apply the formulas above to calculate the number of sub-tree combinations at any node in the tree.

For any arbitrary node in the tree, we can calculate the number of sub-tree combinations that can be formed at that node by traversing down the sub-tree until we reach a sub-tree with only leaf nodes. For any leaf node, *E*, that is required, the value of S_E is always 1, and for a leaf node, *E*, that is not required, the value of S_E is always 2. We can then calculate the value of S_E for a node, *E*, that has only children that are leaves using our product formula, and thus proceed back up the sub-tree as illustrated above. As long as we traverse all branches of the sub-tree, the order in which we traverse the sub-tree does not matter.

In general a node can repeat from N to M times, where N and M are non-negative integers and M is the minimum and N the maximum number of times the node can repeat. The equations below can be used to handle the more general cases:

Equation 4: $C_E = 1 + \sum_{k=M}^{N} (S_E)^k$ (node *E* optional, $M > 0$)

Equation 5: $C_E = \sum_{k=M}^{N} (S_E)^k$ (node *E* required, $M > 0$)

where S_E is the number of sub-trees that can be formed at node *E* for a single occurrence of node element *E*, M is a

positive integer equal to the minimum number of times the node can repeat, and N equals the maximum number of times that *E* can repeat.

If node *E* is optional and repeats from 0 to N times, Equation 3 above is used to find the number of sub-tree combinations.

Noting that the number of sub-tree combinations that can be formed is equal to the number of structurally distinct messages that can be constructed, we can now apply our method to the tree shown in Figure 2. Above we found that $C_{F1} = 85$. For *F2*, we have $C_{F2} = 3$. We then have at *S1*, $S_{S1} = 85 \times 3 = 255$. Node *S1* can repeat from one to two times. Therefore, $C_{S1} = 255^1 + 255^2 = 65280$.

Thus we see that even for our overly simplified profile, there are still 65,280 possible messages that can be generated. Even this number of messages is generally too large to work with, and the number for a more realistic profile would be much larger. Before examining a more realistic profile below, we continue working with our toy profile to illustrate our approach for reducing the number of messages through various filtering techniques. Before we proceed with filtering messages, however, we want to take a closer look at our counting formulas.

3.1 Concept of Order

We began our attempt at counting sub-trees by examining the node element *F1*. We found that for a single occurrence of F1 there are four sub-trees that can be formed; we represented them as: *(F1 (C1 C2 C3))*, *(F1 (C1 C2))*, *(F1 (C2 C3))*, and *(F1 (C2))*.

We want to take another look at the sub-tree combinations we can form at *F1*. We'll simplify our representation of the sub-trees above by adding some definitions. We will define *N1* to be exactly *(F1 (C1 C2 C3))*, we will define *N2* to be exactly *(F1 (C1 C2))*, *N3* to be *(F1 (C2 C3))*, and *N4* to be *(F1 (C2))*. In this way, we can represent all sub-tree combinations that can be formed by two occurrences of *F1*with the first occurrence of *F1* equal to *N1* as:

(N1 N1), *(N1 N2)*, *(N1 N3)*, and *(N1 N4)*.

We can represent all combinations where the first occurrence of *F1* is *N2* as:

(N2 N1), *(N2 N2)*, *(N2 N3)*, and *(N2 N4)*.

When the first occurrence is *N3*, we have:

(N3 N1), *(N3 N2)*, *(N3 N3)*, and *(N3 N4)*.

And for *N4*, we have: *(N4 N1)*, *(N4 N2)*, *(N4 N3)*, and *(N4 N4)*.

We can see that we have sixteen combinations, but if order is not significant, some of the combinations are equivalent. If we count combinations and order is insignificant, we only have ten combinations. To make this calculation in general, we make use of formulas from combinatorial mathematics. When repetitions are allowed (and order does not matter), the expression $C'(n,r)$ can be used to represent the number of combinations for choosing *r* objects from a set of *n* objects. The value of the expression is given by:

$$C'(n,r) = (n + r - 1)! / r!(n - 1)!$$

When two objects are selected from a batch of four, the value of the expression is ten as we got above.

We can then replace Equation 4 and Equation 5 above, with the following formulas for counting sub-tree combinations when order does not matter. From above, we have the number of sub-tree combinations that can be formed at E given by:

Equation 6: $C_E = 1 + \sum_{k=M}^{N} C'(s_E, k)$ (node *E* optional, *M* > 0)

When node *E* is required, we have:

Equation 7: $C_E = \sum_{k=M}^{N} C'(s_E, k)$ (node *E* required, *M* > 0) where S_E is the number of sub-trees that can be formed at node *E* for a single occurrence of node element *E*, M is a positive integer equal to the minimum number of times the node can repeat, and N equals the maximum number of times that *E* can repeat.

If node *E* is optional so that the node may appear from 0 to N times, we replace Equation 3 above for the number of sub-tree combinations with:

Equation 8: $C_E = \sum_{k=0}^{N} C'(s_E, k)$

If we apply Equation 8 at *F1*, we have $C_{F1} = 1 + 4 + 10 + 20 = 35$, instead of the 85 combinations we calculated above. If we continue and calculate the number of sub-tree combinations possible for our toy example above, we find that there are only 5,670 combinations instead of the 65,280 that are possible when order is significant.

4 Message Filtering

We now want to look at techniques for filtering the message set. Rather than create messages for all possible sub-tree combinations, we seek to find a subset of messages that does not grow exponentially with node cardinality. We will outline a filtering approach whereby we attempt to maintain the structural richness of the original message set. We do this by including all sub-trees that we identified above, but we do not include all the sub-tree combinations in our messages. We follow a practical approach that is based largely on an examination of the end point values.

Leaf nodes are handled in a simple and somewhat arbitrary manner, but a manner that will not sacrifice the structural richness we are seeking to maintain. For optional leaf nodes, we include two sub-trees. In one we omit the node. In the second, we construct a sub-tree for the maximum end point value.

For non-leaf nodes, we examine a number of cases below. For all cases, if the minimum endpoint value is zero, we change it to one, and work with that case instead. Once we complete our calculations, we then add one additional sub-tree. Since the minimum endpoint value does not have to be zero for an optional node, we apply the same strategy for all optional nodes.

For the cases considered below, let L equal to the lower endpoint value and U equal to the upper end point value for a given node. Let N equal to the number of sub-trees that can be formed at that node.

First we consider required nodes that have only leave nodes for children.

Case 1. $L = U = N$. In this case, a message with the node element repeated U times is constructed. With each repetition of the node, a distinct sub-tree is represented. Since the number of sub-trees, N, that can be formed at the node is equal to U, all sub-tree structures are encompassed with this one message.

Case 2. $L + U = N$. In this case, a message with the node element repeated U times is again constructed. A second message with the node element repeated L times is also constructed. A distinct sub-tree is represented for each occurrence of the node in the two messages. Since, the total number of sub-trees, N, is equal to the sum of L and U, all sub-trees can be maintained with two messages.

Case 3. $L + U > N$. In this case, two messages are again constructed, one with the node repeated U times and one with it repeated L times. For the first N occurrences of the node, a distinct sub-tree representation is included. For the remaining $N - L + U$ occurrences, sub-trees are arbitrarily selected.

Case 4. $L + U < N$. For this case, let K be a positive integer equal to $N - (L + U)$, i.e., $K + L + U = N$, the number of sub-trees. Then $2 + upper\ (K \div U)$ messages are constructed, where *upper* is a function that maps its argument to the nearest integer that is greater than or equal to its argument. When $K \leq U$, this results in constructing three messages. If $K > U$, then the number of times that K divides U determines how many messages are constructed.

First two messages are constructed in the same manner as above, one with the node repeated U (maximum) times and the second with the node repeated L (minimum) times. After these two messages have been constructed, there are K sub-trees that have not been accounted for in either message. When $K \leq U$, $upper\ (K \div U)$ equals one. And since K is less than U, and the element can be repeated up to U times, with one additional message the remaining K sub-trees can be accounted for. If $K > U$, then when U divides K, with K / U additional messages all sub-trees can be accounted for. If U does not divide K, then the remainder will be the number of sub-trees that are unaccounted for. Since this number must be less than U, with one additional message all sub-trees can be accounted for. Thus, $2 + upper\ (K + U)$ is the number of messages we want in both cases.

An example of the above can be illustrated by selecting a few arbitrary numbers. If $L = 2$ and $U = 5$ and there are 10 sub-trees so that $N = 10$, then $K = 3$. It's clear that with the first two messages 7 sub-trees can be accounted for and with one additional message the remaining 3 can be accounted for. So, 3 messages are needed, which is the number the above formula yields. If N is 20 instead, then K is 13. The above formula yields 5 messages in this case. We can see that with 4 messages 17 sub-trees are accounted for (one of the messages accounts for only 2 sub-trees) and to account for the remaining 3 one more message is needed.

The cases examined above apply to required nodes. Optional nodes can easily be handled by adding one to the results obtained for required nodes to account for the empty sub-tree.

To extend our approach to include all nodes, we proceed from the bottom of the tree as we did above and work upwards. Once we establish the numbers for sub-trees at nodes near the bottom of the tree, we treat the nodes as leave nodes and continue moving up the tree until we reach the root of the tree. Rather than trying to elaborate the details abstractly as we've done above, we'll illustrate by applying the technique to the toy tree we constructed in Figure 2 above.

4.1 A Filtering Example

Here we illustrate the technique described in the previous section by applying the technique to the profile tree shown in Figure 2. We begin by examining node element F1. There are four sub-trees to account for at F1. Using the notation introduced above, the four possibilities are represented as follows: *(F1 (C1 C2)), (F1 (C1 C2 C3)), (F1 (C2 C3)), (F1 (C2)).*

Ignoring the optional case initially, at *F1*, $L = 1$, $U = 3$ and $N = 4$. Thus *Case 2* above applies where $L + U = N$. In this case, two messages are constructed. Three of the sub-trees are accounted for with one message, and with one additional message the remaining sub-tree is accounted for. Ignoring the other branches of the tree for now (here, we can use any valid configuration), we would construct one message for the maximum endpoint containing the following structure: *(F1 (C1 C2)), (F1 (C1 C2 C3)), (F1 (C2 C3))*

The second message with the node element repeated only one time would capture the one remaining sub-tree: *(F1 (C2))*

Since the node element F1 is optional, another message would be needed to account for the empty tree.

Next, we move up the tree to node S1. The node is treated as a node with children that are all leaf nodes would be treated. At F1, the child node is treated as a node that can vary in one of three ways to account for the three messages that need to be incorporated. At F2, two variations are accounted for. So, S1 is treated as a node where six sub-trees need to be incorporated into the messages that are to be constructed.

Thus, $L = 1$, U = 2, and $N = 6$ at S1. This falls under Case 4 above with $L + U < N$. In this case $K = 3$, and *upper* $(K \div U) = 2$, so that four messages are constructed. The four messages can be represented as follows:

(i) (S1) (S1 (F1 (C1)(C2)) (F1 (C1)(C2)(C3)) (F1 (C2)(C3)))
(ii) (S1 (F1 (C1)(C2)) (F1 (C1)(C2)(C3)) (F1 (C2)(C3)) (F2)(F2))
(iii) (S1 (F1 (C2))) (S1 (F1 (C2)) (F2))
(iv) (S1 (F2))

In the first message, the element S1 is repeated twice, the maximum endpoint value. In the second, the minimum end point value is used. In the third, the element is again repeated the maximum number of times allowed. At this point, one sub-tree is still not represented, and this structure is handled with the forth message.

We have thus substantially reduced the number of messages that need to be constructed for our toy tree from 65,280 (when order is significant) to only 4.

4.2 Applying the Counting and Filtering Techniques

We have developed a Java implementation of the above techniques and have successfully applied them to substantially larger profiles than the toy example we looked at above. We have also incorporated the implementation into our Message Maker application. This addition to Message Maker gives it the capability to provide a count of the total number of structurally distinct messages for any profile, provide a message count for the various filtering techniques, and generate the filtered message sets.

The table below shows the results of applying two filters to a more realistic test profile. The second filter is a modification of the one described above. The essential difference between it and the one described above is captured in how we handle the element node F1 in our toy example. Rather than accounting for four sub-trees, we only consider two, one with all optional children included and one with only required nodes included. By applying both filtering techniques we were able to reduce the message set from over 8 million to less than 8 hundred; we are investigating additional techniques that should further reduce the message set.

Order Significant		Order Insignificant			
None	F2	None	F2	F1	F1&2
8257536	8064	5308416	5184	786432	768

Table 1: NIST Test Profile

5 Conclusions

We have successfully formulated a technique for identifying HL7 messages conforming to an XML profile. The technique gives us a measurement for assessing the scope of our testing problem. While the scope of the problem is generally very large, through filtering techniques, we are able to reduce it to a workable size. We can apply filters that substantially reduce the size of the total message set without significantly sacrificing the structural qualities of the original set. Furthermore, we are capable of generating HL7 encodings of the filtered message sets.

As of yet, we have not attempted to assess the effectiveness of our message sets in evaluating HL7 implementations. In the next phase of our work, we plan to undertake the development of a testing framework that will enable us to make this assessment. We anticipate that the message sets generated using the techniques discussed in this paper will provide an effective set of messages for assessing the core behavior of an HL7 implementation. To comprehensively test an implementation, it will be necessary to also incorporate error messages, i.e., messages that adhere to the implementation profile in a number of ways, but violate

important aspects of the profile requirements. Also, it will be necessary to consider content variations of the test messages. Support for generating messages in both the latter cases is provided by our Message Maker implementation. With the capabilities built into Message Maker combined with the use of the techniques described in this paper, we anticipate the capability to generate sets of messages that will enable thoroughly assessing the behavior of implementations of the Hl7 standard.

6 References

[1] Software Conformance Testing & Diagnostics Division, Information Technology Laboratory, NIST; http://www.itl.nist.gov/div897/

[2] Health Level 7 (HL7); http://www.hl7.org

[3] HL7 Standard Version 2.4, ANSI/HL7 V2.4-20

[4] Message Maker; Developed by the National Institute of Standards and Technology (NIST). http://www.nist.gov/messagemaker.

[5] Message Maker User's Guide. Version 1.4 January 2006, R.Snelick, L.Gebase, S.Henrard. http://www.itl.nist.gov/div897/ctg/messagemaker/docs/UsersGuide1.4.pdf.

[6] HL7 Standard Version 2.5, ANSI/HL7 V2.5-2003. June 26, 2003. Chapter 2 Section 12 "Conformance Using Message Profiles", pg 43-62.

[7] Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation 04 February 2004. http://www.w3.org/TR/2004/REC-xml-20040204