## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Summary of Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>July 9, 1998</td>
<td>Initial Release for comments</td>
</tr>
<tr>
<td>July 22, 1998</td>
<td>Update for 7/23 telephone call. Deals with the instance object model and the approach to evolution.</td>
</tr>
<tr>
<td>July 23, 1998-A</td>
<td>Revisions based on the 7/23 telephone call.</td>
</tr>
<tr>
<td>July 31, 1998</td>
<td>Revisions based on the 7/30 telephone call.</td>
</tr>
<tr>
<td></td>
<td>• requirement for a validating parser</td>
</tr>
<tr>
<td></td>
<td>• use of entities for indirection in element definition</td>
</tr>
<tr>
<td></td>
<td>• localization</td>
</tr>
<tr>
<td></td>
<td>• meta-information in an instance</td>
</tr>
<tr>
<td>August 9, 1998</td>
<td>Revisions based on the 8/4 telephone call.</td>
</tr>
<tr>
<td></td>
<td>• discussion on “pointy bracket” syntax and legacy systems</td>
</tr>
<tr>
<td></td>
<td>• added MET-MEI metamodel</td>
</tr>
<tr>
<td>August 13, 1998</td>
<td>Revisions based on e-mail traffic</td>
</tr>
<tr>
<td></td>
<td>• discussion on options for legacy system vendors</td>
</tr>
<tr>
<td></td>
<td>• discussion on ER7 contingencies</td>
</tr>
<tr>
<td></td>
<td>• revisions to MET-MEI metamodel</td>
</tr>
<tr>
<td></td>
<td>• discussion on localization</td>
</tr>
</tbody>
</table>
Contents

Introduction ............................................................................................................................................. 3

Givens .................................................................................................................................................... 4
  Introductions to Types and Components ......................................................................................... 4
  Context ............................................................................................................................................... 4
  “Document” vs. “Message” .................................................................................................................. 5
  Definitions .......................................................................................................................................... 5
  Discussion .......................................................................................................................................... 6
  Inter-Version Evolution Approach .................................................................................................... 10
  Requirement for a Validating Parser ................................................................................................. 13

Work Items ......................................................................................................................................... 15
  Intra-Message Instance Pointers ....................................................................................................... 15
  Instance Object Model ..................................................................................................................... 15
  Representing Message Elements in the Computed DTD ................................................................. 18
  Local Variations (Localization) ........................................................................................................ 19
  Metamodel ......................................................................................................................................... 20
  Naming and Name Scope .................................................................................................................. 29

Other Items Not Addressed ............................................................................................................... 30
  Templates .......................................................................................................................................... 30
  Relationship of Instance Graph to Type Definition ......................................................................... 30
  Meta-Information in an Instance .................................................................................................... 30

Historical Information ...................................................................................................................... 32
  Local Variations .............................................................................................................................. 32

Figures ................................................................................................................................................ 32
  Figure 1. Version 3 process ............................................................................................................. 5
  Figure 2. Message Element Instance Object Model ....................................................................... 16
  Figure 3. Message Element Type Metamodel .............................................................................. 23
  Figure 4. Message Element Instance ............................................................................................ 24
  Figure 5. Gunther’s metamodel ....................................................................................................... 25
  Figure 6. Wes’ model ...................................................................................................................... 27
  Figure 7. Dan’s metamodel ............................................................................................................. 28

Introduction

An open group of members of the HL7 Control/Query Technical Committee are contemplating the following proposition:

XML shall be the only character-coded Implementation Technology Specification (ITS) for Version 3, at least initially. There are no plans to produce an ITS for Encoding Rules-7.

These members have conducted a series of conference calls to analyze the issue. It is a goal of the group to bring a unified proposal to the Control Query group at the September HL7 Meeting, for discussion and endorsement. It will then undertake necessary revisions to the MDF and the beginnings of an ITS for the January meeting.

This document is meant to be updated after each conference call. It is divided into four main sections:

- **Givens** records what has been agreed upon.
- **Work items** represents the author’s attempt to summarize issues that are not yet resolved.
- **Other items not yet addressed** includes topics that have been identified.
- **Historical Information** includes material that seems to have become obsolete. Frequently such material can be useful to help reconstruct a thought process; occasionally it is found to be relevant after all.

We hope that more items will move into the “givens” over time.

Attendance varies from call to call, and all calls are open and announced in advance. However, these individuals have been involved in more than one call.

Paul Biron  Member Kona Editorial Group
Bob Dolin  Member Kona Editorial Group
Joachim Dudeck  Chair, HL7 Germany
Lloyd Harding  Member Kona Editorial Group
Stan Huff  Co-chair, Vocabulary Technical Committee
Ed Larsen  HL7 Member
Doug Pratt  Co-chair SIGOBT
Larry Reis  Co-chair, Control/Query
Wes Rishel  Vice-chair Technical Steering Committee
Gunther Schadow  SIGSecure Co-chair
Mark Shafarman  Co-chair, Control/Query
Dan Trainor  Co-chair, SIGOBT
Mark Tucker  Co-chair, Control/Query
David Webber  Member, XML/EDI Group

Many others have contributed through the lively debate on the Control/Query, Modeling and Methodology, and SGML list servers.
Givens

Introductions to Types and Components

Programming language-ish examples:

```
Type Point Contains
    X of type Real
    Y of Type Real

Type Line Contains
    Start of Type Point
    End of Type Point

Variable A of type Line

A.Start is a Point
A.Start.X is a Real.

A is a variable.

X, Y, Start, and End are not variables. They are Components. No memory is allocated anywhere for an X until the declaration for variable A.
```

In general, types can be atomic or contain Components, each of which has a name and a type.

We must be careful with this analogy, because HL7 doesn’t have variables. HL7 has types and instances.

HMDs define message types. However, HMDs contain a lot more than simply the message type, and this additional information is also a normative part of the standard. The additional information includes semantics and business rules. More on this later.

Context

Figure 1, reproduced from MDF-98, helps to place this discussion in context. (As discussed below, the caption for the gray area will change from “Defining a Message Structure” to “Defining a Message Type” in MDF-99). XML is an ITS. There may be other ITSs. One of the primary benefits of XML is that the small boxes at the bottom labeled HL7 Message Creation and HL7 Message Parsing can rely on ubiquitous standard programs when the message instance is an XML document.
“Document” vs. “Message”

This effort is largely independent of the Kona project being conducted in the SGML Special Interest Group, although there is considerable cross interest. It is common usage to refer to any complete XML production as a “document.” In this summary, the author has chosen to use “message” to avoid confusion with the Kona effort and to emphasize the context of the discussions.

Definitions

An italicized term in the definitions is the subject of another definition. This section does not repeat the definition of terms from MDF-98 that have not changed, unless important to the exposition.

message element
The basic unit of structure of a version 3 message. Message elements can contain other message elements. Message elements may contain other message elements.

message element instance
An actual set of data that is part of an actual message.

message element type
A specification for the values that a message element can take on in its instances.

primitive message element type
A message element type that does not contain other message elements.
Composite message element type
A message element type that contains other message elements (components). Each component message element has a name and a type. Each component of an element must have a different name.

Choice message element type
A composite message element type for which only one of the components will be sent in an instance.

Message
A message element that is the unit of information interchange among information systems conforming to HL7 Application Roles.

Message instance
An actual message element instance corresponding to a message type.

Message type
The type of a message. A message type is always a single message element type, although the type will contain many components.

External message element type
When defining a message element in an HMD, the type that is assigned to a component may be defined within the HMD or it may be defined externally. A message element type that is defined externally to an HMD is an external message element type. External message element types include common message element types and primitive message element types.

RIM-derived components
Certain message element components are mapped in the HMD to specific attributes of the Reference Information Model. The type of these components must always be the same and must always be consistent with the attribute type of the corresponding RIM attribute. The message element type that must be used for a Component that is mapped to a RIM attribute is stored with the RIM. The message element type associated with a RIM attribute is always a common message element types.

"Pointy-bracket" syntax
A syntax that generates a subset of all XML productions. Instances in the subset don’t contain any meta-information.

Primitive message element type
The type definitions for scalar message elements are defined where?

Common message element type
Certain message element types are defined in Common Message Element Definitions. These are defined separately from their use as the type of a Component in their HMD.

Hierarchical Message Definition (HMD)
A metaobject that defines message types. It also defines the linkages of the message types to Interactions and the linkages of certain message element types to attributes of the Reference Information Model.

MDF-98

Discussion

Relationship to XML
We have gone back to reexamine the HMD to make it harmonious with XML, and in the process we are simplifying the message model. Some important principles of this effort are:
• XML is an ITS that is downstream from the HMD. The HMD definitions apply equally to the component ITSs and others that may come along. Because of this, the HMD cannot be defined in terms of XML concepts.

• The current assumption is that a DTD that corresponds to a message type can be computed from the HMD.

• At this stage in our discussions, we have not agreed that the every message element type will be expressed in the computed DTD using the same style.

There are aspects of a message type as defined in an HMD that cannot be practically expressed in an XML. These include the specifics of some of the primitive data types and specific limits on multiplicity other than “one” or “many”. It is possible that there are others. In an environment where a validating XML parser is used, a receiving system cannot assume that the output of a successful parse is correct with respect to these other aspects. The converse is also true, software systems that use a validating XML parser can be simpler and faster. This is because they can be assured that most of the structure of the message has been validated before they begin to process the document objects.

**Impact on Legacy Systems**

At the same time, it is not clear that good quality, supported XML parsers will be available for all the operating systems that are typically used to implement information systems that use HL7. HL7 goals have always represented a compromise between making use of the best technology and providing support to “legacy” systems. In particular, we are not sure of XML support for MVS, VMS, OS/400 and various M (nee MUMPS) environments.

We are currently working under an assumption that represents a compromise: the methodology for computing DTDs from the HMD will not require the use of meta-information in message instances, and HL7 will not permit it. This assumption lets developers of legacy systems undertake to develop rather simple parsers for HL7 V3 messages sent using the XML ITS. These messages have tags a la XML, may have attributes defined in the tags, and meet the XML criterion for being “well-formed”. We have taken to calling this syntax the “pointy-bracket” syntax to distinguish it from the full XML syntax which may contain meta-information in instances. The parsers developed using this simplification will not meet the W3C criteria for a non-validating XML parser, but they will be adequate to parse valid HL7-XML messages.

At the same time, we are collecting information that shows some promise that vendors in legacy environments will have access to validating parsers in the future. Current information points to several freeware parsers implemented in Java and the availability of Java Virtual Machines (JVM) in various environments. Currently, there are substantial portability problems among various JVM. This problem is far wider than the Windows-specific features in the Microsoft JVM. This is not unexpected giving how young the technologies are, but it is not yet clear that Parser A which runs on JVM A will also run on JVM B or be supported in that environment.

Until such software is proven there will also be questions about the speed of the implementations and the robustness of associated communications services.

From the information gathered to date we can take hope that validating parsers will be ubiquitous in the legacy environments, but it remains to be determined whether the hope will be realized and if so, when.
<table>
<thead>
<tr>
<th>Operating System</th>
<th>Software Product</th>
<th>Information Nugget</th>
</tr>
</thead>
</table>
| OpenVMS for DEC Alpha | JVM | From: John C. Spinosa, MD, PhD [mailto:john@spinosa.com]  
Sent: Tuesday, August 11, 1998 4:39 PM  
Subject: OpenVMS and JVM  

From the Compaq Web site:  
http://www.openvms.digital.com/affinity/documents/osad_w5_001.html  

**Java Developer's Kit (JDK) for OpenVMS Alpha systems**  
The Java Development Kit (JDK) V1.1.4 allows users to create applets that run in Java-enabled browsers and develop Java applications. Java applications can be written once and run on any operating system that implements the Java run-time environment, which consists primarily of the Java Virtual Machine (JVM). OpenVMS Alpha users can now access Java's functionality using the Java Development Kit (JDK) V1.1.4, available for OpenVMS Alpha Version 7.1 systems. The Java Development Kit (JDK) V1.1.4 is a set of building blocks containing basic development tools and a rich set of class libraries. |

| IBM JVM & Parser | JVM | Date: Tue, 11 Aug 1998 13:00:39 -0400  
From: Dave Rees <drees@worldmachine.com>  

Don't rule out the use of Java as an option. IBM has a freely available validating XML parser written in Java AND they are steadily porting the Java Virtual Machine over to all of their platforms. I don't know about OS/400 and MVS specifically, but it is certainly worth your checking out. The parser can be found at http://www.alphaworks.ibm.com.  

Note that IBM also provides a number of Java based solutions for legacy integration. Coupled with the parser it may be a good solution to your problems. |

---

**Can We Really Get By Without ER7?**  
This project is in many respects a stalking horse for an initiative to clean up the logical structure of HL7 messages. If that initiative is successful, it will be easier to teach and apply the version 3 methodology. After the hard part of getting the RIM and interaction model right, the process of building the HMD will be more transparent and less time consuming. When we do version upgrades we will be able to spend more time on the important logical issues and less time on the confounding details that must be got right but contribute little to advancing the scope or clarity of HL7.  
The project document starts out by saying that the project is based on an unproven proposition: that version 3 will be passed and accepted with only an XML option for encoding messages as strings. The implicit point is that we would not include a syntax that can be parsed by a set of extensions to existing HL7 parsers. The argument in favor of not doing ER7 are that good quality XML parsers will be ubiquitous. For legacy systems that are (a) on operating systems that might not have ubiquitous XML 


parsers, and (b) survive Year-2K, it is almost trivially easy to parse the subset of XML that we would
force vendors to use.

Will this idea fly?

This question is resistant to logical analysis because it will be decided, by a number of executives with
minimal deliberation and no joint discussion. The ER7 approach to Version 3 does force some non-trivial
expansion to the capabilities of existing parsers. But people are skeptical about trying something new and
usually underestimate the difficulty of modifying something old.

The new approach might be called “uniformly recursive types”. The first premise is that “data types”
don’t stop at the data field level. Segments are also type definitions. Indeed messages are type definitions.
This is simply recognizing a fact, but for some old fogies (like me) it might take a little time to get used to
it.

The next premise, in simplified form, is “every part of a message, no matter how fine- or coarse-grained
is either a composite message element or a primitive message element. Composites can contain a mixture
of composites and primitives. The message itself is nothing more than the composite that contains
everything else. Each subcomponent is identified by name.

The way we were previously approaching structure in version 3 was to say:

- a message is a set of segments or segment groups; messages are identified by a name (a code, really)
- a segment group is a set of segments or segment groups; segment groups are identified by names
- a segment is a set of data fields; segments are identified by name
- a data field is either a primitive or a set of components (aka a compound); data fields are
  identified by their position within a segment
- a component is either a primitive or a set of components; components
- a primitive is a simple string of data with no components; primitives are identified by their
  position within the data field or outer component.

Each of these levels has different syntax rules and specific constraints on how it can be combined with
other levels. Each level has to be treated differently in the analysis required to map a message information
model into a message definition. Numerous special cases arise when a group later needs to expand an
existing message format.

In the old approach “person name” and “the PID segment” are like apples and bicycles. In the new
approach they are both composites, more like little fish and big fish.

Earlier the author did some interviews with system architects and came to believe that vendors would
easily tolerate recursion of data types at the field level and below. That is, they would accept infinite
nesting of subcomponents using a bracketing notation instead of the use of multiple delimiters for a fixed
number of levels of components. The same research confirmed that the vendors would accept recursive
grouping of segments (we already do that).

We did no research on the related issue of eliminating the discontinuity in the middle (segments) and
being uniformly recursive. We did no research on naming components rather than identifying some by
position.

Although the benefits of the uniform element hierarchy are substantial, the author would hot have
considered re-opening it but for the excuse of XML. It is ironic that an XML approach can easily be
crafted to create XML elements that resemble segments, so XML does not compel HL7 to use the more
elegant approach. At the same time, as people have gotten to know XML (and to a certain extent HTML)
they have become more aware of the possibility of a recursive hierarchy of types even if they don’t call it by that name. XML does not compel us to a new model, but it gives us the context and pulpit to advocate it.

Our current strategy is to develop a uniform type model as reflected in this document, and to go as far as the HIMSS version 3 demo showing only the XML syntax. There are a few “hooks” in the model that will permit us to “fall back” to a parallel ER7 representation based on the reception we find. In this approach, segments are an artifact of the ER7 syntax. There is no “segment object” in the message element type and message element instance models and no requirement that DTDs give special treatment to the particular level in the type nesting that gets treated as a segment.

In other words, we will do our best to lead HL7 to a better way of doing things. If, however, the realities of organizational inertia and the economics of upgrades make this impossible, we will limit the negative impacts of segments and positional notation to the maximum extent possible.

Reductions in Potential Ambiguity

The definitions above and our project assumptions provide substantial reductions in the potential for ambiguities in grammars derived from message types. In particular,

- XML parsers, and the software that implement other ITSs, are presumed to provide explicit indications of the scope of message elements that cannot be made ambiguous by any legal combination of message elements and multiplicity.

  This is a sharp contrast with the version 2.X encoding rules, where repeating or optional segment groups were closed by recognizing a segment that cannot appear as the first token of an instance of an iteration of the segment group.

- Because every component is named, there are no anonymous sub-types.

  This is a contrast to XML elements. This XML declaration for type a

  ```xml
  <!ELEMENT a ((x, y, z) | (x, y, x))>
  ```

  can be expressed, but is not acceptable because a parser looking at the tags <a><x> cannot determine immediately whether it is parsing the (x, y, z) or the (x, y, x) choice. Each of the options is a subtype that is not given a tag.

The definitions for HL7 message elements imply that each branch of the choice is a component that has a name. If the elements of a DTD follow the message element of the HMD, it might create a structure like this:

```xml
<!ELEMENT a (xyz | xyx )>
<!ELEMENT xyz (x, y, z)>
<!ELEMENT xyx (x, y, x)>
```

A production from this type a would contain either <a><xyz><x> or <a><xyx><x>. When the parser reaches the <x> it has already seen a tag that lets it know it is in (x, y, z) or (x, y, x).

Inter-Version Evolution Approach

Requirements

Here is the statement of requirements from MDF-98. Italicized material has been added.
HL7 has historically regarded upward compatibility as a prime requirement. This assures that upgrades to the protocol may be introduced in a network gradually, node by node, rather than by an abrupt switchover.

The goals for upward compatibility in Version 3 are:

A. HL7 will provide the maximum degree possible of interoperability among systems operating on older and newer versions of HL7 protocols in the Version 3 family through compatible enhancement.
   a) A message structure that is modified in a newer version of the protocol must be acceptable to a system designed for the prior V3 release. However, a system built for the prior release will only extract the information that was defined for that prior release.
   b) A message structure created in accordance with an older version of the V3 protocol must be acceptable to a system designed for a later V3 release. In some cases, this will mean that the system built for the newer release will not receive certain information fields because they were not a part of the older version of the message structure in use by a specific sender.
   This requirement effectively mean that a system builder will have to declare its support for a range of protocol versions.

B. Where compatible enhancement is not possible, HL7 will require evolution in the protocol to happen gradually, so that users can introduce the change into their networks gradually.
   a) The messages associated with all interactions that are newly defined in a version of HL7 must not be sent to a receiver that conforms to the older version.
   b) A message structure may be declared obsolescent in one release, with a stated alternative message structure. Both the obsolescent message structure and its alternative can be used by all systems supporting that release.
   c) The obsolescent message structure may be declared obsolete and dropped when still another HL7 version is issued.
   d) An obsolescent message structure may not be declared obsolete for at least two years from the date of the version that first declared it obsolescent.
   e) The above notwithstanding, if a new Implementation Technology Specification (ITS) is introduced, HL7 may specify that conformance to the ITS does not require dealing with message structures that are obsolescent when the new ITS is introduced.

C. To the maximum degree possible, these restrictions should not impose limitations on the evolution of the overall reference model. Section 5.1.2.2 of the MDF-98, includes specific techniques for maintaining message compatibility in the face of evolutionary changes in the model.

D. There are no restrictions on making changes to the information model if those changes do not impact the message structures that were described in a prior version of the Standard.

Use Cases
In various e-mails we have identified the following types of changes to message types.

Upgrade old OPTIONAL to new REQUIRED
Add New Item
   add new OPTIONAL
   add new REQUIRED
Remove Old Item
was OPT
was REQ

Structural Changes
(new type has same meaning as old, but different structure)
Scalar to composite
Did not repeat before, now repeats
Value moved from one place to another
  (was this message element, now is a component of this message element)
  (was in this message element, now in another message element)

The current MDF essentially repeats the version 2 approach for meeting these requirements. These can be
generalized by saying

• Receivers that are on an earlier version of the sender must ignore message elements that were not
described in the receiver’s version of the HMD.
• Receivers that are on a later version than the sender must have special programming to deal with
message elements that are required in the receiver’s version but were not in the senders.

The second rule is quite onerous, although no more so than it was in version 2. The difficulties are
somewhat reduced by noting that it only applies to message elements that were newly added. In addition,
the obsolescence feature of version 3 means that the special-case code may ultimately be removed.
In version 2, the evolution rules supported an orderly evolution of data types. This is neither an addition
nor a deletion; it is a change. The approach to inter-version compatibility must support these kinds of
changes.

Approaches
In the new approach, every subelement is named and every name is represented in each production. This
is obvious for XML; it becomes part of the requirement for other ITSs. This feature means that it will no
longer be required to add new message elements at the end of existing message elements. This is a
substantial improvement because it simplifies merging two different sets of extensions to a message
element, as when work occurs independently in two different countries.

Within the XML ITS, there seems to be great benefit in ensuring that each message is parsed according to
the DTD associated with the sender, even when the receiver is working with an earlier or later version of
the HMD. It is much easier to deal with the various rules for ignoring unexpected message elements or
compensating for missing ones when operating on the Instance Object Model than it is when parsing.

One approach that has been advanced is to use three techniques:

• **Additions** are simply inserted into a message element where they fit logically; receivers on older
type object models conceptually prune these nodes from the instance tree.
• **Deletions** are noted as obsolescent; senders continue to send them; receivers that are on a newer
version of the protocol recognize them as obsolescent and prune the corresponding instances from
the instance tree.
• **Easy changes** are treated as a combination of additions an deletions; the changed message element
appears redundantly during the obsolescence period.
• **Difficult changes** (defined below) are treated using a version choice type.

A version choice is a special compound. Each component is a 2-tuple; the first subcomponent identifies
the applicable version and the second component is message element as it should appear in the message
for the specified version. Unlike a choice, all the alternatives of a version choice appear in each message
instance. Receivers replace a version choice in the instance tree with the “subsubcomponent” corresponding to the version of the receiver.

This table reviews the use cases with respect to these treatments.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Treatment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upgrade old OPTIONAL to new REQUIRED</td>
<td>Treat as an addition (receivers must have special-case code to deal with it not being present during the obsolescence period.)</td>
</tr>
<tr>
<td>Add New Item</td>
<td></td>
</tr>
<tr>
<td>add new OPTIONAL</td>
<td>Treat as an addition</td>
</tr>
<tr>
<td>add new REQUIRED</td>
<td></td>
</tr>
<tr>
<td>Remove Old Item</td>
<td></td>
</tr>
<tr>
<td>was OPT</td>
<td>Simply stop sending it</td>
</tr>
<tr>
<td>was REQ</td>
<td>Treat as a deletion</td>
</tr>
<tr>
<td>Structural Changes</td>
<td></td>
</tr>
<tr>
<td>(new type has same meaning as old, but different structure)</td>
<td>Treat as a difficult change.</td>
</tr>
<tr>
<td>Scalar to composite</td>
<td>Treat as the addition of the subelements in the element type</td>
</tr>
<tr>
<td>Did not repeat before, now repeats</td>
<td>Treat as an addition</td>
</tr>
<tr>
<td>Value moved from one place to another</td>
<td></td>
</tr>
<tr>
<td>(was this message element, now is a component of this message element)</td>
<td>Treat as a difficult change.</td>
</tr>
<tr>
<td>(was in this message element, now in another message element)</td>
<td>Treat as an addition and a deletion.</td>
</tr>
</tbody>
</table>

**Requirement for a Validating Parser**

This discussion centers on Figure 2 and particularly the box labeled “ITS-Specific Software Agent” for the receiver. Many of the participants believe that most implementers will use a validating parser as part of the software represented by that box. This implies, however, that the receiver has access to the DTD of the sender.

We have not addressed the mechanisms by which System B would have access to the DTD of system A, particularly if system B is on an older version of HL7 than system A. HL7 versions do not change rapidly, so there is no need for the receiver to dynamically access DTDs from a repository. They could be distributed using the model of “master file updates”.

On the other hand, some HL7 members may find the simple requirement to dynamically switch DTDs between incoming messages to be onerous. This would be true if they chose not to use an off-the-shelf validating XML parser to implement their ITS-specific agent. For these implementations, adapting to a change in the sender’s DTD may not be simple as providing a different string to the parsing software.

Further, an HL7 message that is valid with respect to its DTD may not be valid with respect to its HMD. There are features of the messages that cannot be practically validated using parsers based on the current version of XML. In addition, the model in Figure 2 shows that the sender and the receiver may definitely be on different HL7 versions. The “message dance” algorithm for matching the instance tree to the type tree certainly rely on the message being well-formed (in the XML sense). However, the additional checking necessary to confirm validity could be combined with the checking necessary to validate the parse tree against the HMD/type tree.
Arguably, HL7 could serve its membership better by requiring implementers to use off-the-shelf parsers, because the parsers are likely to be more robust than home-grown parsing algorithms. However, HL7 has never attempted to standardize the implementation of its specifications. To do so risks a flurry of objections from members whose legacy systems do not have ready access to a selection of off-the-shelf parsers.

The current status of these discussions is a compromise among the various points of view.

- All version 3 messages will be valid productions of a DTD that is computed from the HMD that describes the message.
- The changes in message types that occur through evolution follow specific rules defined in Inter-Version Evolution Approach.
- We are not aware of any feature of our methodology that requires an implementer to use a validating parser or to have access to the sender’s DTD.
- We have not ruled out using a validating parser and requiring a receiver to have access to the sender’s DTD. If a proposed feature of the methodology requires this, we will evaluate the benefit of the feature versus the cost of making the requirement.

As discussed under Relationship to XML, it is not clear that we will even require a non-validating parser that fully meets the requirements of W3C.
Work Items

Items in this section have been discussed to some extent, although there is no representation that the working group has a consensus on how to do them.

Intra-Message Instance Pointers

We expect MDF-99 to include the ability to identify message element instances in one part of a message and refer to them in elsewhere in the message. We may want to consider adding definitions like the following to the “Givens”.

**target message element**  Message elements may be designated as the targets of pointers. Instances of target message elements may be referred to by pointer message elements. Being the target of a pointer does not create a distinct type.

**pointer message element**  A message elements type can be designated as being a reference to one other specific target message element type.

Including a pointer message element as a component of a larger message element can be used to create a linkage between an instance of the larger message element and instance of the target message element.

In addition to the above, we should decide if we agree on the following principles.

- Pointers are type-specific. If the type declaration for message element A says it is a pointer to message element B, an instance of A cannot point to an instance of a message element of any other type except B.
- Every instance of a pointer message element must point to a target instance.
- The implementation of pointers is ITS-specific.
- If IDs are used to implement pointers, the same ID value may be used to identify target message element instances of different types.
- If IDs are used to implement pointers, the same ID value may be reused in another message to identify an unrelated message element.

Instance Object Model

The discussion in this section relies on these new definitions.

**Instance Object Model**  A graph of nodes that represent a message instance. Each node represents a message element instance.

**Type Object Model**  A graph of nodes that represent a message type. Each node represents a message element type.

Conceptually, the dark gray portion of Figure 1 could be replaced with Figure 2. The HL7 Instance Object Model is a collection of objects (instances) drawn here as a graph. The nodes are instances of message elements; each is an instance of a message element type as defined in the HMD. The solid edges represent containment and the broken-line edges represent pointers.
The word “conceptually” is an important part of the preceding paragraph. HL7 has traditionally not seen fit to standardize the design of software. Current ideas conformance testing in version 3 envision the entire “HL7-Conformant Application” as a black box.

HL7 has, however, written recommendations for object models of messages. That is what SIGOBT had done and is doing.

We can also conceptually think of the portion of the HMD that represents type information as a “type object model”. We can further note that the type object model that was used to create the message is that of the sender. The Instance Object Model that is built in the receiver’s space may not correspond exactly to its Type Object Model. These conceptual definitions allow us to define the matching process as an algorithm that simultaneously steps between nodes of the Instance Object Model and makes matching steps among nodes of the Type Object Model.

(Personal note: I visualize this as a “tree dance”. The tree dance is two tree-walks occurring together, with one foot walking the nodes of the instance model and the other walking the nodes of the type model.)

Figure 2 highlights the assumption that the sender and receiver are on different versions of HL7. (That assumption is described in Inter-Version Evolution Approach, p 10.) This assumption leads to an important issue: how much can the receiver assume about the correctness of the instance object graph after it has been delivered by the ITS-specific agent. In terms of the XML ITS, we could make one of two statements:
• The ITS-specific agent has determined that the XML was well-formed, so there can be a parse tree. There is no guarantee that the sender created an XML utterance that is valid with respect to the sender’s DTD.

• The ITS-specific agent has determined that the XML was valid with according to the sender’s DTD.

The latter statement assumes that the receiver dynamically determines and applies the correct DTD for a message, since the sender might be on 3.3 while the receiver is on 3.2. This may reduce the complexity of the “tree dance” since it doesn’t need to program for some error conditions. This represents an operational constraint that is more than trivial, and perhaps substantial.

The question is not, however, unique to XML. Any system of replicating objects from sender to receiver will have a similar set of issues. What will happen if the “message factory” for the receiver is on a different version than the “message factory” for the sender? The answer depends on how closely the message factory is tied to the specifics of an HL7 version.

One can ponder the relationship between the Message Element Instance object and a node in the XML Document Object Model. One hopes that our chosen use of XML permits the ITS-specific software agent to implement the Message Element Instance object as a wrapper around $\text{XMLNode}$ or some other important interface in the XML document object model.

There is material relevant to this topic under Local Variations (Localization).

There is substantial need for a systematic way to deal with extensions to a release of version 3 before they become embodied in a subsequent release. In version 2 this was called “Z segments” and “Z codes”; one can debate how systematic it is; but it assures that local segment IDs will not be countermanded by subsequent releases of HL7.

The need for local variations occurs at the national level and again at the scope of an individual HL7 network. (We have historically called the latter level a “site”, but that usage is probably dated.)

Some are examining an additional question that was not considered under version 2: how can localized messages be used beyond the locale? They are looking for a way to assure that the base standard content is accessible by examining messages generated according to locale-specific requirements. As a group we have not articulated this as a requirement; it has not been discussed.

The approach we choose must be consistent with XML, but it must be usable in other implementation technologies as well.

Open Issues about Functional Requirements

Historically HL7 has thought of all implementations as being specific to a “site”. Early in its history, the site was probably a hospital. In recent years, however, we have spoken more about the “HL7 network”, meaning set of sites that are administered in common. Increasingly the assumption of common administration may need scrutiny. HL7 supports one-to-many transactions and inter-enterprise scope. It seems that during the life of version 3 a single message may start out within one domain of administration and be distributed to other sites in multiple different domains.

This expansion of scope will require us to re-examine what we mean by localization. Is it sufficient to consider countries to be domains onto themselves? It is likely that a single HL7 message instance will need to carry local variations for more than one country? This is not a question where it is safe to take the broadest view, because local variations in the definition of HL7 frequently reflect locally defined functional requirements for the underlying information systems. Furthermore, the additional coordination requirements to harmonize local variations could reduce the timeliness of HL7.

While there are unknowns, certain requirements can be proposed.
Unambiguous localization. Localized variations should be identified in message instances in a way that clearly identifies the domain of localization. A message that has been localized to one domain should not carry contents that could conceivably be falsely recognized in another domain.

Multiple concurrent localizations. Message instances should be able to simultaneously carry information described by different sets of localizations. There can be argumentation as to whether multiple national locations should be required, but the version 3 approach should not preclude them. Furthermore, there may be a hierarchy of localizations that are applied for regions within a nation within an economic region.

No evolution conflicts. Messages that have been localized for a specific release of version 3 should not be made invalid by subsequent releases to the standard. (Currently national franchises add elements to the end of segments. This creates a problem when future HL7 releases define other data for the same field positions.)

An Approach to Localization Based in the Message Element Type Model
Support for Localization, p. 22, contains an approach to localization. Because it is placed in the type model, it applies equally to all ITSs.

Metamodel, p 19.

Representing Message Elements in the Computed DTD
In the algorithm for computing an HMD, what style should be used represent message elements.

Alternatives
Some alternatives that have been presented include

• Primitives in Attributes. Represent composite message elements as elements with subelements; represent primitive message elements as empty elements with a standard attribute name for the value.

• Primitives as PCDATA. Represent composite message elements as elements with subelements; represent primitive message elements as DTD elements with PCDATA between their tags.

• Alternating Tag-Type. Represent each message element as a pair of nested DTD elements. The outer element names the element and the name of the inner element is the name of the type of the message element. Represent composite message elements as elements with subelements; represent primitive message elements as DTD elements with PCDATA between their tags.

• Multiple Representation Styles. Use different styles for different HL7 data types. Composite elements that arise during the construction of an HMD would be represented as DTD elements with subelements. Many HL7 composite data types would be represented the say way. However primitive HL7 data types and selected composite HL7 data types could be represented using different techniques. Some might be represented with all or part of their content in attributes.

Requirements
There should be a way to represent HL7 data types that contain marked up text without including the DTD for the marked up text in the DTD for the message.

The representation style must not “excessively” extend the length of messages over the length they would have had using the ER7 syntax defined in MDF-98. We do not have a clear definition of “excessive”, but
generally concur that if the messages were three times longer this would be acceptable in light of the benefits of using XML.

It is likely that the representation styles we choose initially will evolve over time. Evolution can be motivated by at least three influences

- experience using XML for HL7 messages
- evolution in XML itself, and the related standards that are still under development
- an evolving understanding of how the chosen techniques interact with related technologies.

**Possible Techniques**

The group has shared ideas about the representation style.

CDATA may permit the inclusion of marked-up text.

Here are several ways that attributes might be used.

- ID and IDREF for intra-message pointers
- VSN as an attribute of type NMTOKENS to designate elements that are maintained for compatibility with prior HL7 versions. (Several tokens could associate the same element with more than one prior version.)
- TY as an attribute of type NM_TOKEN to explicitly transmit the type of an element.
- an attribute that describes encapsulated data contained in the element
- V as a CDATA attribute that contains the value for EMPTY elements.

**Other Considerations**

A technique that has been discussed from time to time is to create a level of indirection in element definitions by using entities.

```xml
<!ENTITY % ACK.CONTENT "MSH,MSA,ERR?">  
<!ELEMENT ACK (%ACK.CONTENT;)>  
<!ELEMENT ERR (%ERR.CONTENT;)>  
<!ELEMENT MSA (%MSA.CONTENT;)>  
<!ELEMENT MSH (%MSH.CONTENT;)>  
```

Such an approach may facilitate the use of supplementary meta information to alter a standard DTD for localization, message evolution or for other purposes.

**Local Variations (Localization)**

There is substantial need for a systematic way to deal with extensions to a release of version 3 before they become embodied in a subsequent release. In version 2 this was called “Z segments” and “Z codes”; one can debate how systematic it is; but it assures that local segment IDs will not be countermanded by subsequent releases of HL7.

The need for local variations occurs at the national level and again at the scope of an individual HL7 network. (We have historically called the latter level a “site”, but that usage is probably dated.)

Some are examining an additional question that was not considered under version 2: how can localized messages be used beyond the locale? They are looking for a way to assure that the base standard content
is accessible by examining messages generated according to locale-specific requirements. As a group we have not articulated this as a requirement; it has not been discussed.

The approach we choose must be consistent with XML, but it must be usable in other implementation technologies as well.

Open Issues about Functional Requirements

Historically HL7 has thought of all implementations as being specific to a “site”. Early in its history, the site was probably a hospital. In recent years, however, we have spoken more about the “HL7 network”, meaning set of sites that are administered in common. Increasingly the assumption of common administration may need scrutiny. HL7 supports one-to-many transactions and inter-enterprise scope. It seems that during the life of version 3 a single message may start out within one domain of administration and be distributed to other sites in multiple different domains.

This expansion of scope will require us to re-examine what we mean by localization. Is it sufficient to consider countries to be domains onto themselves? It is likely that a single HL7 message instance will need to carry local variations for more than one country? This is not a question where it is safe to take the broadest view, because local variations in the definition of HL7 frequently reflect locally defined functional requirements for the underlying information systems. Furthermore, the additional coordination requirements to harmonize local variations could reduce the timeliness of HL7.

While there are unknowns, certain requirements can be proposed.

**Unambiguous localization.** Localized variations should be identified in message instances in a way that clearly identifies the domain of localization. A message that has been localized to one domain should not carry contents that could conceivably be falsely recognized in another domain.

**Multiple concurrent localizations.** Message instances should be able to simultaneously carry information described by different sets of localizations. There can be argumentation as to whether multiple national locations should be required, but the version 3 approach should not preclude them. Furthermore, there may be a hierarchy of localizations that are applied for regions within a nations within an economic region.

**No evolution conflicts.** Messages that have been localized for a specific release of version 3 should not be made invalid by subsequent releases to the standard. (Currently national franchises add elements to the end of segments. This creates a problem when future HL7 releases define other data for the same field positions.)

**An Approach to Localization Based in the Message Element Type Model**

Support for Localization, p. 22, contains an approach to localization. Because it is placed in the type model, it applies equally to all ITSs.

**Metamodel**

Several of the participants have used metamodels to illustrate their points. This is valuable and several are included here.

**Purpose of the Metamodel**

We have not discussed the long-term purpose of the metamodel. Differences in metamodels often stem from different ideas as to the purpose. Some possible purposes are:

* to illustrate the relationships inherent in the definition of type, and the relationships between types and instances
• to define exactly the instance object model and the type object model (as defined above), with suitable care to see that their APIs are practical for the programs that must be written to use them
• to update the metamodel of all of version 3 that is in MDF-98 and serves as a basic for tooling that supports the development of the standard.

Models

Several different models are shown below, with the author’s attempt to capture some of the dialogue that has gone on.

The Message Element Type/Message Element Instance Model

Figure 3 and Figure 4 together are the most recent submission. The dotted line connecting Message_element_type to Message_element_instance is intended to show a correspondence but not necessarily an association that would be implemented by containment or object references. Conceptually each MEI object corresponds to exactly one MET object. However, it may not be possible to navigate that relationship directly. All the subclasses of Message_element_type end in the suffix “MET”. All the subtypes of Message_element_instance end in “MEI”.

The metamodel also differs from prior models in that it includes an explicit object to contain repetitions. Figure 3 emphasizes the message type model and its relationship to other HL7 metaobjects. The blue boxes are proxies for elements of the information model portion of the full HL7 metamodel. The green boxes represent things that are defined by the metaobjects in the message type model. They include

• Message types—the specification of messages that can be sent using HL7.
• Common message element descriptions—types defined by HL7 functional committees to represent important building blocks for message types. These are combinations of attributes from the RIM defined for a specific functional purpose. Examples of common message element descriptions might include the identification of an inpatient or a reference to an order.
• Compound HL7 data types—fundamental types defined externally to the RIM. Person name is an example of a compound HL7 data type.

Certain subspecies of Message_element_type can represent attributes of the RIM. The RIMRelatedMET class captures this characteristic. If the attribute is a primitive data type (such as the string id in the Patient_encounter class) it will be a PrimitiveMET that represents the RIM attribute. More frequently, however, the RIM attribute is had a compound datatype (such as primary_psnm in the Person class.) In this case a CompositeMET will have the association with the RIM attribute and it will contain multiple PrimitiveMET components through the has_components and is_of_type associations.

The HL7 information model supports only a two-level hierarchy: classes and attributes. This precludes there being a RIMRelatedMET that contains another RIMRelatedMET. The metamodel does not capture that characteristic directly; instead, it uses a business rule.

A primary emphasis of the type model is a uniform set of metatypes that can recurse to create the hierarchical information structures of messages. This supports natural implementations in object APIs and XML.
Limited Support for Segment-Positional Syntax

However, there remains a contingent requirement to support a segment-field-position syntax such as ER7. The model treats this requirement as an add-on, so as not to force the inelegant segment structure on the other ITSs. It primarily uses business rules that are constraints in the hierarchy of message element types.

- No segment can contain a segment, because segments are delimited by a fixed record delimiter.
- Every RIMRelatedMET must be contained in a larger Message_element_type that is a segment, either directly or through intermediate Composite_element_types.
- The sequence of the components of a CompositeMET is used to identify the component parts of a segment or a compound data field.

Two features of the metamodel support these requirements:

- The CompositeMET class has a boolean \textit{is\_segment}.
- The ComponentDescriptor class has an attribute \textit{sequence}.

Support for Localization

In the message element type hierarchy the ComponentDescriptor class has an attribute \textit{locale} which is intended to be a code. The intent is that all the information in a flagged component is specific to the definition of the locale. This allows for locale-specific information to be added to a message type. A scheme for generating unambiguous IDs is required.

In theory, the ID does not need to be transmitted in a message instance, because the locally-tagged code has its own name.

Because of the hierarchy of uniform METs the approach can be applied anywhere from data types on up. It also supports message instances carrying multiple locale-specific inclusions.

What happens when a localized message is processed by a receiver that is using a message element type model that does not include the localizations? The treatment is identical to that of a receiver that receives a message generated a sender on a later version of the protocol. It has not been programmed to do anything with the localization, so it ignores that branch of the instance graph.

**Important open issue:** this approach does not deal with the requirement to omit data from a message in order to meet locale-specific requirements.
Figure 3. Message Element Type Metamodel.
Figure 4. Message Element Instance
Figure 5. Gunther's metamodel.
Gunther’s metamodel clearly illustrates the logical relationships. Wes has commented that the association between RIM class and Composite is illusory. “Pointer relationships” are implicit in the type definitions, rather than being described by a feature of the model.

Wes’ metamodel, below, is more addressed to the process described above in “Instance Object Model”, p 15. It is particularly concerned with the notion that the message may have been built with a Type Object Model that is different than the Type Object Model of the receiver, so that it may be necessary to navigate the instance model without recourse to the type model.

Gunther has commented that having explicit pointers as shown in Wes’ model should not be allowed in HL7.

Dan’s metamodel, below, includes more attention to the species and relationships among the instances.
Figure 6. Wes' model.
Figure 7. Dan's metamodel.
**Naming and Name Scope**

RIM names are long and frequently repeat in the HMD. Although we are not rigorously concerned about message length, the consensus is that message element names in the HMD should be abbreviated when they appear as a tag or an attribute name in a DTD.

There are concerns that the HMD structure in MDF-98 creates multiple message element components with the same name.

- Within the RIM, attribute names are implicitly qualified by class names. Our modeling style guide encourages the reuse of names under certain circumstances.

- When a CMED is invoked more than once in an HMD all of its components, and their components, etc., appear again. This is a case of re-using a type; the pattern of name usage should be identical in both invocations.

- The Message Object Diagrams support multiple objects drawn from the same class in the RIM. Their different uses may represent multiple uses of the same type, as when the cluster of Provider-Person-Stakeholder-Stakeholder_ID appears twice because of different associations in the MIM.

- Multiple objects drawn from the same class in the RIM can also represent different types. A message element about a patient will include different attributes of the Patient class than a message element about a practitioner.

DTD element names cannot be defined more than once. In the current version of XML they cannot be qualified.

Both the issue of name length and name re-use argue for the ability to create pseudonyms in the HMD. Issues of reinvoking the same type argue for a different, not yet unspecified, solution.
Other Items Not Addressed

Templates
Stan Huff has introduced the notion of templates. In essence, templates are an alternate form for describing types with less reliance on the RIM. Templates are consistent with the RIM but may express constraints that cannot be traced back to the RIM. For example, the RIM may allow a general recursion to create a level of observations, and a wide choice of code domains in identifying the observations. Based on that model, specific templates may be constructed to describe specific observations such as a blood pressure reading or a measurement of body temperature. (Each of these observations represents a collection of specific subobservations. For example, a blood pressure reading includes systolic and diastolic numeric values (with units) a single method (e.g., adult cuff) applicable to both readings, a single patient posture applicable to both readings, etc.
The theory is that modeling the tens of thousands of such observations in the RIM would not be practical.
We have not worked out how templates will work. They should be harmonious with XML but they should be expressed in terms of the HMD or some other metaobjects that transcend a specific ITS.

Relationship of Instance Graph to Type Definition
We need to characterize the instance graph objects in detail, based on an examination of the processing necessary to match an instance graph created by the ITS agent (and matching the sender’s message type) with the receiver’s message type. Suggestion: talk about a type graph separately from the HMD.

Meta-Information in an Instance
XML allows meta information to come in a message instance. This can include the entire DTD and supplementary DTD content. This issue interacts with a discussion contained under Requirement for a Validating Parser. One argument for avoiding a requirement for a validating parser is that it is “easy” to write a non-validating parser. But the W3C says that a non-validating parser must process meta information in the instance.
The entire notion of embedded metainformation is one that we must examine as a group. We have agreed to the following statement. "Every HL7 Version 3 XML message will conform to a DTD that is computed from the HMD."
Any correct validating XML parser must be able to take a correct HL7-XML message along with its DTD and produce a parse tree of XML nodes along with the indicator that the message is valid. Any vendor whose messages fail such a test are not conforming to the DTD or to the other rules of XML, so the vendor cannot be said to be producing correct HL7-XML messages.
However, we also didn't say that the sender can add additional DTD material in the message instance. (We didn't say either way.) If we were to allow this, we must understand how to know that the additional DTD material does not conflict with the relationship between the message type and the semantics that is described in the “official HL7” HMD.
Assume that we can put some limits on totality of DTD definitions that apply to an HL7-XML message instance. How would we specify those limits? An easy way would be to say they are limited to the DTDs that are computed from the HMDs. This particular policy would support proposition that it is "easy" to parse HL7-XML messages, create an instance parse graph, and determine if the message was well-formed.
On the other hand, suppose HL7 decides to make use of the ability to include supplementary DTD information in a message instance. Perhaps we would use it for localization or for version evolution. Perhaps another use of the capability might arise. One presumes that we can find ways to prescribe the use of supplementary DTD material so that it does not conflict with the semantic definitions of the HMD, or so it can be dealt with systematically in the processing of the instance parse tree.

Any such policy would support the theory that it is "hard" to parse HL7-XML messages, etc., because the specified function includes reading, validating and using DTD material in the message instance. Given the way things work in standards group we have to recognize that if we give up the ability to use DTD in the instance now, it will be hard to get it back later. As vendors begin to use home-grown parsers rather than 3rd-party parsers they will argue vociferously against future changes to the spec that would require them to add that capability later.
Historical Information

Local Variations
We initially discussed ways to use XML to address the problems described in the Local Variations (Localization) section.

XML-Specific Variations
We have identified two XML-specific approaches to local variation.

Two approaches to local variations we have discussed are:

- Entity Redefinition
- Architectural Transformations.

In each case it is assumed that the base HL7 message type will be modified to meet the needs of the locale. An appropriate administrative process will have defined a locale-specific DTD.

Entity Redefinition
We could adopt a style wherein DTD elements are defined indirectly through entity definitions. The benefit is that entity definitions within a message instance take precedence over entity definitions in an external DTD. Therefore, any element could be redefined locally within a message.

HL7 DTD
-------------
<!ENTITY % ACK.CONTENT "MSH, MSA, ERR?">
<!ELEMENT ACK (%ACK.CONTENT;)>  
<!ELEMENT ERR (%ERR.CONTENT;)>  
<!ELEMENT MSA (%MSA.CONTENT;)>  
<!ELEMENT MSH (%MSH.CONTENT;)>  

Locally customized message
------------------------------------------
<!DOCTYPE ACK SYSTEM "HL7.DTD" [  
<!ENTITY % ACK.CONTENT "MSH, MSA, ZOO?, ERR?">  
<!ELEMENT ZOO (FOO, BAR)>  
<!ELEMENT FOO (ST)>  
<!ELEMENT BAR (ST)> ]>  
<ACK>  
.... message goes here ...  
</ACK>

As described, this approach offers no assurance that the entity redefinitions are consistent with the locale-level specifications.

Architectural Transformations
In this approach the locale-specific DTD replaces the HL7 DTD for the same message type. In addition the necessary specifications are written for an architectural transformation from the locale-specific message to the HL7 message. This approach supports the need to administer local-specific DTDs at the level of the HL7 network. In addition, messages are shorter because each instance does not need to carry the entity redefinitions.