

HL7 MODELING & METHODOLOGY COMMITTEE

HL7 Version 3

Message Development Framework

Version 3.3

December 1999

Authors: George W Beeler, Stan Huff, Wesley Rishel, Abdul-Malik Shakir, Mead Walker, Charlie Mead, Gunther Schadow

Copyright © 1999 by Health Level Seven, Inc.

ALL RIGHTS RESERVED.

The reproduction of this material in any form is strictly forbidden without the written permission of the publisher.

Table of Contents

1. INTRODUCTION	1-1
1.1 ACKNOWLEDGMENTS.....	1-1
1.2 DOCUMENT SCOPE.....	1-1
1.3 OVERVIEW	1-2
1.4 WHY A MAJOR NEW VERSION?	1-3
1.4.1 Difficulties with the Existing Process.....	1-3
1.4.2 Opportunities to Improve.....	1-3
1.4.3 Optionality is a Four Letter Word.....	1-4
1.4.4 Limitations of the New Approach.....	1-4
1.5 METHODOLOGY PREVIEW	1-5
1.6 DOCUMENT STRUCTURE.....	1-8
1.7 METHODOLOGY 2000: MESSAGING AND BEYOND	1-9
1.8 REFERENCES.....	1-10
2. GLOSSARY OF VERSION 3 TERMS AND ACRONYMS.....	2-1
3. PRINCIPLES OF VERSION 3	3-1
3.1 SCOPE, TARGET USERS	3-1
3.1.1 Internationalization.....	3-1
3.1.2 Support for Legacy Systems	3-1
3.2 LOOSELY COUPLED SYSTEMS	3-1
3.2.1 Modes and Topologies.....	3-1
3.3 INTER-VERSION COMPATIBILITY.....	3-2
3.3.1 Compatibility with Versions 2.X.....	3-2
3.3.2 Compatibility Among Versions 3.X.....	3-2
3.4 DETERMINING CONFORMANCE	3-3
3.4.1 Application Role.....	3-3
3.4.2 Conformance Claims	3-3
3.5 CONFIDENTIALITY AND SECURITY	3-3
3.5.1 Confidentiality of Patient Information.....	3-3
3.5.2 Authenticated Authorization for Services.....	3-4
3.5.3 Security, Privacy, Non-Repudiation and Integrity.....	3-4
4. MANAGING MESSAGE DEVELOPMENT.....	4-1
4.1 PROJECT SCOPE DEFINITION	4-1
4.2 VERSION 3 METHODOLOGY.....	4-1
4.3 DOCUMENT STRUCTURE.....	4-2
4.4 DATA FIELD DOMAINS	4-2
4.5 QUALITY ASSURANCE PROCESSES	4-2
4.6 PROCESS SUPPORT	4-3
4.7 MANAGEMENT.....	4-3
5. USE CASE MODEL.....	5-1
5.1 OVERVIEW	5-1
5.1.1 HL7 Messages and Use Case Analysis	5-4
5.1.2 Use Case Analysis and Storyboards.....	5-6
5.1.3 Use Case Analysis and the MDF.....	5-7
5.1.4 Actors.....	5-9
5.1.5 Storyboards, Scenarios, and Use Case Paths.....	5-10
5.2 PROCEDURES	5-11

5.3	DOCUMENTATION	5-13
5.4	TUTORIAL SUGGESTIONS AND STYLE GUIDE	5-14
5.4.1	<i>Project Scope Statement</i>	5-14
5.4.2	<i>Use Cases</i>	5-14
5.4.3	<i>Actors</i>	5-17
5.5	CRITERIA.....	5-19
6.	INFORMATION MODEL	6-1
6.1	OVERVIEW	6-1
6.1.1	<i>Information Model Components</i>	6-1
6.1.2	<i>Information Model Notation and Meta-Model</i>	6-1
6.1.3	<i>Types of Information Models</i>	6-2
6.1.4	<i>Information Model Harmonization</i>	6-2
6.2	WORK PRODUCTS	6-3
6.2.1	<i>Static Structure: Classes and Relationships</i>	6-3
6.2.2	<i>Information Content: Attributes, Values, and Constraints</i>	6-8
6.2.3	<i>Dynamic Behavior: States and Transitions</i>	6-15
6.3	PROCEDURE.....	6-17
6.3.1	<i>Construction/Refinement of a Domain Information Model</i>	6-18
6.3.2	<i>Update/Harmonization of the Reference Information Model</i>	6-37
6.3.3	<i>Construction of the Message Information Model</i>	6-40
6.4	SUMMARY OF INFORMATION MODEL STYLE GUIDELINES	6-41
6.4.1	<i>Classes</i>	6-41
6.4.2	<i>Relationship</i>	6-42
6.4.3	<i>Attributes, Data Types, Constraints, and Defaults</i>	6-43
6.4.4	<i>States and State Transitions</i>	6-44
6.4.5	<i>Prepare RIM Change Proposal</i>	6-44
6.4.6	<i>Review RIM Change Proposal with Stewards of Affected Classes</i>	6-45
6.4.7	<i>Participate in RIM Change Proposal Harmonization Meeting</i>	6-46
6.4.8	<i>Define Message-set Specific Association Constraints</i>	6-46
7.	ASSOCIATING VOCABULARY DOMAINS WITH ATTRIBUTES, ELEMENTS, AND FIELDS.....	7-1
7.1	VOCABULARY DOMAINS	7-1
7.1.1	<i>General disclaimer</i>	7-1
7.1.2	<i>Introduction</i>	7-1
7.1.3	<i>Vocabulary Domains, and Vocabulary Domain Specifications</i>	7-1
7.1.4	<i>Validating Vocabulary Domain Specifications and Constraints</i>	7-2
7.1.5	<i>Vocabulary Domain Constraints</i>	7-3
7.1.6	<i>The Domain Specification Database</i>	7-3
7.1.7	<i>Use of the vocabulary domain specification database</i>	7-14
7.1.8	<i>Summary of vocabulary domains used in the specification of vocabulary domains (meta domains)</i> 7-15	
7.2	THE STRUCTURE OF CODED ELEMENTS IN MESSAGES	7-16
7.3	THE GENERAL PROCESS OF MAINTAINING DOMAIN SPECIFICATIONS	7-22
7.4	GOOD VOCABULARY PRACTICES	7-24
7.5	USE OF EXTERNAL TERMINOLOGY'S IN HL7 STANDARDS.....	7-24
7.5.1	<i>Process for registering vocabularies for use in HL7</i>	7-25
7.6	THE USE OF LOCAL VOCABULARIES IN CODED ELEMENTS	7-27
7.7	HL7 VOCABULARY AND THE UMLS METATHESAURUS	7-27
8.	INTERACTION MODEL	8-1
8.1	INTRODUCTION	8-1
8.1.1	<i>Why build the Interaction Model?</i>	8-2
8.2	ELEMENTS OF THE INTERACTION MODEL	8-2
8.2.1	<i>Trigger Event</i>	8-2

8.2.2	<i>Application Role</i>	8-3
8.2.3	<i>Interaction</i>	8-5
8.2.4	<i>Interaction Sequence</i>	8-6
8.3	DIAGRAMMING INTERACTIONS	8-7
8.3.1	<i>Sequence Diagram</i>	8-7
8.3.2	<i>Collaboration Diagram</i>	8-8
8.4	CONFORMANCE AND THE INTERACTION MODEL	8-8
8.5	BUILDING THE INTERACTION MODEL	8-8
8.5.1	<i>Defining Scope</i>	8-8
8.5.2	<i>Building Interactions</i>	8-9
8.5.3	<i>Validating Interactions</i>	8-10
8.5.4	<i>Grouping Interactions</i>	8-10
8.6	INTERACTION MODEL QUALITY CRITERIA	8-11
9.	CONFORMANCE CLAIMS AND ORGANIZATIONAL PRACTICES	9-1
9.1	INTRODUCTION	9-1
9.1.1	<i>Conformance Claims</i>	9-1
9.1.2	<i>Good Organizational Practices</i>	9-2
9.2	CONFORMANCE CLAIM WORK PRODUCTS	9-2
9.2.1	<i>Conceptual Model of HL7 Conformance Claims</i>	9-2
9.2.2	<i>Functional Statement of Conformance Criteria</i>	9-6
9.2.3	<i>Technical Statements of Conformance Criteria</i>	9-8
9.3	GOOD ORGANIZATIONAL PRACTICES	9-9
9.3.1	<i>Good Organizational Practices for Vocabulary</i>	9-9
10.	CREATING MESSAGE SPECIFICATIONS	10-1
10.1	INTRODUCTION	10-1
10.1.1	<i>Overview</i>	10-1
10.1.2	<i>Introduction to Version 3 Message Instances</i>	10-5
10.2	WORK PRODUCTS	10-10
10.2.1	<i>Message Information Model</i>	10-10
10.2.2	<i>Refined Message Information Model</i>	10-13
10.2.3	<i>Hierarchical Message Definition</i>	10-18
10.2.4	<i>Common Message Element Type Definition</i>	10-22
10.3	PROCEDURES	23
10.3.1	<i>Create the Message Information Model</i>	23
10.3.2	<i>Create the Refined Message Information Model</i>	23
10.3.3	<i>Build the Hierarchical Message Definition</i>	26
10.3.4	<i>Creating the Common Message Element Type</i>	29
10.4	DISCUSSION AND HELPFUL HINTS	30
10.4.1	<i>Representing Associations by Containment: Pseudo-hierarchies</i>	30
10.5	CRITERIA FOR EVALUATION OF WORK PRODUCTS	30
10.5.1	<i>Refined Message Information Model</i>	30
10.5.2	<i>Hierarchical Message Definition</i>	31
11.	DEVELOPING HL7 MODELS USING UML AND RATIONAL ROSE	11-1
11.1	INTRODUCTION	11-1
11.2	MODEL TERMINOLOGY, PROPERTIES, DESCRIPTIONS AND STEREOTYPES	11-1
11.2.1	<i>Preparing Rose for properties and stereotypes</i>	11-1
11.2.2	<i>Package vs. Category</i>	11-2
11.2.3	<i>The use of added HL7 properties to capture meta-data</i>	11-2
11.2.4	<i>Capturing rationale. Issues and references in descriptions</i>	11-3
11.2.5	<i>Use of Rose Stereotypes by HL7</i>	11-3
11.3	STRUCTURE OF THE HL7 MODELS AS REPRESENTED IN ROSE	11-5
11.3.1	<i>Overall structure of the Rose model</i>	11-5
11.3.2	<i>Model definition</i>	11-5

11.4	USE CASE MODEL	11-6
11.4.1	<i>Model Structure</i>	11-6
11.4.2	<i>Use case model categories</i>	11-6
11.4.3	<i>Use cases</i>	11-7
11.4.4	<i>Actors</i>	11-7
11.4.5	<i>Use case dependency relationships</i>	11-7
11.4.6	<i>Linking actors to use cases</i>	11-8
11.4.7	<i>Linkage to the subject class for a state transition</i>	11-8
11.4.8	<i>Storyboards</i>	11-8
11.5	INFORMATION MODEL	11-10
11.5.1	<i>Model structure</i>	11-10
11.5.2	<i>Subject areas and data type categories</i>	11-10
11.5.3	<i>Classes</i>	11-11
11.5.4	<i>Attributes</i>	11-11
11.5.5	<i>Generalizations</i>	11-12
11.5.6	<i>Associations</i>	11-12
11.5.7	<i>Composite aggregations</i>	11-13
11.5.8	<i>States</i>	11-13
11.5.9	<i>State transitions</i>	11-14
11.5.10	<i>Data types</i>	11-14
11.5.11	<i>Data type components</i>	11-15
11.5.12	<i>Data type generalizations</i>	11-15
11.5.13	<i>Generic data type instances</i>	11-15
11.6	INTERACTION MODEL	11-16
11.6.1	<i>Model structure</i>	11-16
11.6.2	<i>interaction model categories</i>	11-16
11.6.3	<i>Application roles & Trigger Events</i>	11-16
11.6.4	<i>Interactions</i>	11-19
11.7	CREATING A MESSAGE INFORMATION MODEL (MIM)	11-20
11.7.1	<i>Model Structure</i>	11-20
11.7.2	<i>Assembling the MIM</i>	11-21
11.8	PROPERTIES DEFINED BY HL7 FOR USE IN ROSE	11-22
11.9	OVERVIEW OF USING ROSE TREE II AND REVIEWING MODELS	11-24
11.9.1	<i>Functional Objectives</i>	11-24
11.9.2	<i>User Interface</i>	11-24
11.9.3	<i>Viewing a Model</i>	11-26
11.10	BUILDING AN R-MIM IN ROSE TREE	11-28
11.10.1	<i>Building an R-MIM for the First Time</i>	11-28
11.10.2	<i>R-MIM definition window</i>	11-29
11.10.3	<i>R-MIM node types</i>	11-30
11.10.4	<i>Adding classes to the R-MIM</i>	11-30
11.10.5	<i>Cloning classes in an R-MIM</i>	11-30
11.10.6	<i>Controlling association linkages</i>	11-30
11.10.7	<i>Saving an R-MIM</i>	11-31
11.10.8	<i>Opening a Saved R-MIM from the Repository</i>	11-31
11.10.9	<i>Editing a MOD</i>	11-31
11.11	CONSTRUCTING AN HMD FROM AN R-MIM	11-31
11.11.1	<i>Select root class</i>	11-31
11.11.2	<i>HMD parameters</i>	11-31
11.11.3	<i>Walk the walk</i>	11-31
12.	EXAMPLE_MODEL_FOR_MDF	12-1
13.	SPECIFICATION OF THE HL7 MDF COMPONENTS	13-1
13.1	META-MODEL	13-1
13.2	LIMITATIONS OF THIS META-MODEL	13-1

<i>13.1.1</i>	<i>Vocabulary domain specifications.....</i>	<i>13-1</i>
<i>13.1.2</i>	<i>Use case generalization.....</i>	<i>13-1</i>
<i>13.1.3</i>	<i>Message design model.....</i>	<i>13-1</i>

Figures

Figure 1-1 Four Stages of Message Development.....	1-6
Figure 1-2. HL7 Message Development Process Model.....	1-7
Figure 1-3. Message Development Models.....	1-8
Figure 5-1. This figure shows the UML representation of a prototypical Use Case analysis defining the system boundaries and responsibilities for "Some System." Three Actors using "Some System" have been identified: two human users and one system. The Actors interact with Some System via three Use Cases, each of which represents a system responsibility. Not shown are the associated Products of Value produced for the Actor(s) by Some System as a result of the System's fulfilling of a named Responsibility.....	5-1
Figure 5-2. A schematic representation of the <i>iterative</i> process of Use Case Analysis for defining system boundaries and system responsibilities. System boundaries are defined indirectly through the identification of Actors who lie outside those boundaries. System responsibilities are identified by virtue of the Products of Value that the system produces as a result of an interaction between the Actor and the system. In particular, the Product of Value is the direct result of an interaction between an Actor and the system, and is produced as a result of system fulfillment of one of its responsibilities. The "Product of Value" of Use Case analysis itself, i.e., the collection of Use Case diagrams and associated documentation referred to as the "Use Case model," provides the system's Functional Requirements Specification, as well as a framework for Test Plans and User Documentation/Training materials.....	5-2
Figure 5-3 . An initial "high-level" Use Case analysis has revealed two System Responsibilities. Responsibility 1 has been "decomposed" into two "lower-level" Use Cases which are contained within a subsystem/package named with the <i>noun phrase</i> "Management of Responsibility 1." Note that as the decomposition of a given high-level system Responsibility occurs, the Actors involved with the lower-level Use Cases may not be the same as those at the higher-level. In particular, other packages representing different "higher-level Use Case Responsibility Management" subsystems may themselves become Actors in a given subsystem's Use Cases because the lower-level Use Cases in the subsystem-of-interest produce a Product of Value for the other subsystem.	5-3
Figure 5-4. A simplified high-level Use Case analysis of a virtual HL7 messaging system. System responsibilities are partitioned by message type (and ultimately by message), while Actors are identified by area-of-interest. Each System Responsibility is fulfilled by assembling and transmitting (or receiving and interpreting) the appropriate message(s).	5-4
Figure 5-5. This figure shows a high-level Use Case analysis of a prototypical (real or virtual) healthcare information system. Users (Actors) perform various domain-specific functions using the "system," which is often an interfaced/integrated collection of heterogeneous systems (and/or subsystems.) Message traffic flows between the various systems/subsystems as a direct result of interactions between the various Actors and systems/subsystems. However, the Actors <i>per se</i> are not aware of the messages at the level of construction, transmission, reception, or interpretation. These System Responsibilities are instead handled by the internal messaging subsystems of each of the domain-specific systems/subsystems. A Use Case diagram of a messaging subsystem therefore has non-human Actors. The MDF refers to these Actors as "Application Roles." (See text and Figure 5-6 for explanation).....	5-5
Figure 5-6. This Figure shows the Use Case relationship between the HL7 Messaging System and its Actors, the MDF-defined Application Roles, as well as the larger contextual relationship between the	

HL7 Messaging Conformance System as a set of responsibilities within a Messaging Subsystem (see Figure 5-6). The Actors receiving Products of Value from the HL7 Messaging Conformance System are noted simply as "Messaging Stakeholder"; however, their presence is meant to emphasize the layered architectural application of Use Case analysis in complex systems. Not mentioned are the Responsibilities of the Receiving Application Role ("Receiver Responsibilities," the MDF extension to the notion of Actor)..... 5-6

Figure 5-7. This Use Case diagram depicts the high-level Use Cases for the system "MDF Methodology," (or, more correctly, for the "Use Case Management" subsystem / package of the larger "MDF Methodology" system, i.e., for the "system" that results in the definition of Version 3.x HL7 messages. Actors are outside the system's Boundaries, while the system's (high-level) Responsibilities are depicted via the named ellipses. The *optional* direction association arrows between each actor and the named Use Cases indicate that in each actor/use case interaction, the interaction is initiated by the actor rather than the system (a direction of the association can be reversed in cases where the system initiates the interaction.) Note that both the Technical Committee and the Technical Steering Committee have an interaction with the use case "Develop Scope Statement" indicating that both Actors receive a Product of Value from the Use Case. 5-8

Figure 5-8. A Use Case diagram depicting the high-level Use Cases for the "Patient Management" system (or subsystem) from the Example Model in Chapter 12. All Use Cases are initiated by the associated actors..... 5-9

Figure 5-9. A typical Actor hierarchy. Note that the root of the hierarchy is an abstract Actor, i.e., an Actor that does not represent a real-world system user. Abstract Actors are named in UML using *italics*. 5-12

Figure 5-10. This figure uses a subset of the Use Cases from the Example Model (Chapter 12) to illustrate the use of the <<extend>> and <<include>> stereotyped dependency relationships that can exist between two Use Cases. An *extending* Use Case adds behavior to the *extended* Use Case at specifically defined "Variation Points" within the *extended* Use Case. An <<extend>> relationship between two Use Cases is used when the *extended* Use Case can function on its own in most situations, but requires additional behavior to handle non-error exception conditions. The <<include>> relationship is used to factor out common behavior, which is encapsulated in the *included* Use Case. Neither the *included* nor the *including* Use Case normally functions alone. Note that because both the <<extend>> and the <<include>> relationships are named stereotypes of the UML "dependency" relationship, an association utilizing either of the stereotypes indicates that the "source" has a dependency on the "target," i.e., the source is sensitive to a change in the target. Finally, note that this diagram adopts two alternative graphic conventions (neither of which is currently supported in Rose): the System Boundary between the Actor(s) and the System's (subsystem's) Use Cases is shown; and a direction is indicated on the association link between the Actor and the various Use Cases. In the absence of a directional indicator, it is assumed that the Use Case is initiated by the Actor..... 5-16

Figure 5-11. The Figure represents a hypothetical Use Case diagram for a subsystem called "Lab Test Execution" which lives within a larger system called "Management of Lab Tests." This system (not shown) has a Use Case named "Perform Lab Test." The decomposition of that Use Case (i.e., the set of use cases that collaborate to fulfill the Use Case) are Use Cases within the "Lab Test Execution" subsystem. The Actors and associations for one of the Use Cases ("Order Lab Test") are shown. The fact that the Use Case is associated with two Actors suggests that the Use Case itself may be further decomposed (Figure 5-12). 5-18

Figure 5-12. The Use Case in Figure 5-11 involving association with two Actors has been decomposed into two distinct System Responsibilities, each being initiated by a single Actor. This decomposition may be represented by either indicating that the two Use Cases in Figure 5-12 reside in a subsystem of the system in which the single Use Case in Figure 5-11 resides, or, by use of a collaboration association between the "parent" Use Case and its "children." In either case, the two Use Cases "Order_new_lab_test" and "Order_reflexive_lab_test" define a more granular architectural layer than the layer defined by their parent "Order_lab_test." NOTE: a collaboration..... 5-18

Figure 6-1. One association defined in the information model can have multiple instances. Each instance of an association connects two objects. The number of instances of an association connecting to one object is regulated by the multiplicities.....	6-5
Figure 6-2. Resolving a many-to-many association (a) through an associative intermediary class (b).....	6-7
Figure 6-3. A simple state-transition model representing the life-cycle of an activity.....	6-15
Figure 6-4. State-transition model with nested states and preemption: the activity can be terminated prematurely, if it is not done yet.	6-16
Figure 6-5. State-transition model with parallel states. Regardless of the sub-states above the dashed line, the state “held” can be entered and left independently. When “Note Done” is left, all of the sub-states in “Note Done” will be left too, including “Held.”	6-17
Figure 6-6. Visualizing the ONE-OF constraint on a bundle of outgoing associations representing a choice.	6-24
Figure 8-1. Each component is described below. This model is in contrast this to HL7 Version 2 which had trigger events and, implicitly, interactions; but no formal notion of senders and receivers.	8-2
Figure 9-1. HL7 Conformance Claims	9-1
Figure 9-2. Conceptual Model of Conformance Claims	9-3
Figure 10-1. Creation of Message Specifications and Messages.....	10-1
Figure 10-2. Relationship Among Work Products	10-4
Figure 10-3. Message Types, Interactions, Trigger Events, and Application Roles	10-5
Figure 10-4. Example Version 3 message in a possible XML style.	10-6
Figure 10-5. Hierarchy of message elements in the example message.	10-8
Figure 10-6. Message Information Model from the “Example Model”	10-11
Figure 10-7. Recursion in a Refined Message Information Model.....	10-12
Figure 10-8. Instance example of recursion.	10-13
Figure 10-9. Recursion through an intervening class.	10-13
Figure 10-10. Refined Message Information Model, Graphical Format.....	10-15
Figure 10-11. Interactions for the Example HMD.....	10-24
Figure 10-12. “Shopping List” of Information for a Hierarchical Message Definition	10-24
Figure 11-1. Example of HL7 properties for a Rose class.	11-2
Figure 11-2. Structure of HL7 Models in Rose Browser	11-4
Figure 11-3. Properties for the Model, as captured in Rose.	11-5
Figure 11-4. An example use case diagram with a use case hierarchy.	11-7
Figure 11-5. Storyboard sequence diagram from the example model.....	11-9
Figure 11-6. State transition diagram from the example model.	11-13
Figure 11-7. Partial application role diagram from the example model.....	11-17
Figure 11-9 Example of a model in RoseTree view	11-25
Figure 11-12 MIM definition dialog.....	11-29
Figure 12-1. Example Use Case Model	12-3
Figure 12-2. Example Information Model	12-7
Figure 12-3. State Diagram For Patient Class.....	12-10

Figure 12-4. State Diagram for Patient_encounter Class	12-14
Figure 12-5. Interactions for Patient Subject Class	12-19
Figure 12-6. Application Role Diagram For Example Model	12-25
Figure 13-1. Use Case and Interaction Models	13-4
Figure 13-2. Information Model.....	13-5
Figure 13-3. Data type and Vocabulary Domain Models	13-6
Figure 13-4. Message Design Model (left side)	13-7
Figure 13-5. Message Design Model (right side)	13-8

Tables

Table 7-1. Value Set Definition Table.....	7-6
Table 7-2. Set Operators.....	7-7
Table 7-3. Version Tracking Table for the Vocabulary Domain Specification Database	7-8
Table 7-4. Value Set Relationship Table	7-9
Table 7-5. Source Vocabulary Representation Table	7-11
Table 7-6. Observation Identifier to Value Set Linking Table	7-13
Table 7-7. Meta-Domains Used in the Domain Specification Database	7-15
Table 11-1. HL7-defined properties in Rose.....	11-24

1. Introduction

This document is to be used by members of the Health Level Seven (HL7) Working Group. It provides a methodology for developing HL7 messages for HL7 Version 3.0 and beyond. This document is under development, and will change as the HL7 Working Group gets more experience in developing messages for the Version 3 family of standards.

If this document, and the methodology for message development that it discusses, is a substantial accomplishment, this is because it relies on the work of many other groups and researchers.

- The process by which messages are developed from the Information Model, as well as many characteristics of the methodology as a whole draws heavily from the work done by CEN TC251 WG 3.¹
- The effort to establish modeling as a central activity for standards development relies on the pioneering efforts of MEDIX for its study of the role of data models in standards development.
- Notions of model development and harmonization were worked out by the Joint Working Group for a Common Data Model.
- Many of the approaches HL7 has developed have been implemented and explored by the Andover Working Group.

1.1 Acknowledgments

The HL7 Message Development Framework has been built through the efforts of a large number of authors and reviewers. Special credit is due to Jacob Golder for his efforts to guarantee the proper treatment of Object Oriented techniques within this document. There were many other people who contributed to the ideas and text contained within. The HL7 Methodology & Modeling Committee wishes to acknowledge the contributions of the following people: Norman Daoust, Gary Dickinson, Jack Harrington, Karen Van Hentenryck, Ted Klein, Clem McDonald, Linda Quade, Larry Reis, Rob Seliger, Mark Shafarman, and Mark Tucker.² A substantial contribution has also been made by the participants in the Modeling and Methodology list server supported by the HL7 Modeling and Methodology Committee.

1.2 Document Scope

This document defines the steps in the HL7 Message Development Process, and it indicates the criteria for passing from step to step. In other words, it defines a methodology for defining HL7 message formats. It includes material and references to guide the message developer, and to help ensure the quality of the HL7 standard.

HL7 members should use the document as a tool for understanding the process of building messages and the models that support message development. It is a reference manual that describes each stage of building messages, how to use the tools that supports this process, and the concepts involved.

¹ Comite Europeen de Normalisation, Technical Committee 251, Working Group 3. This group defines the methodology for developing standardized messages to support European healthcare standards development. Refer to the document cited for more information.

² The list of contributors, by its nature, is both dynamic and subject to undeserved omission. The editor has tried to recognize all those who have submitted comments on this document.

1.3 Overview

This document leads a Technical Committee (TC) Project Team³ through the process of building HL7 messages. It supports an HL7 project from the point that the need for a new message or group of messages is perceived until actual messages are defined. HL7 developers should use this document as a guide to the creation of new messages for Version 3 and beyond. HL7 Working Group members will also note that the HL7 Reference Information Model⁴ plays an important part in this process, and that this handbook will only come fully into use once that Reference Information Model has been created and can be used in message development.

The HL7 message development methodology is not just a set of notations, but a systematic process for taking the message developer from original requirements to implementation.⁵ The methodology takes into account a persistent tension between the desire to improve the quality of the end product by following a defined process and the need to deliver a working standard in a short time. It presents existing software applications as software agents that interact with other software agents across different platforms, systems, architectures, and languages, to achieve a greater level of interoperability. This requires sharing the semantic content (and often the pragmatic content) of the represented knowledge between applications. Translation alone is not sufficient because each application carries out its processing on the basis of implicit assumptions about the meaning of what is represented. For two applications to effectively inter-operate, such assumptions must be shared. That is, the semantic content of the various data must be preserved.

This methodology follows a typical development life cycle, and is divided into three distinct phases: requirements, analysis, and implementation. During each phase different aspects of the healthcare environment⁶ are defined, and consistency checks are performed. This development process seeks to insure consistency of the contents created for each phase (within each model), and between the phases (between models).

Currently, the healthcare environment has been divided into sub-domains and each of the HL7 Technical Committees is assigned one or more sub-domains.⁷ Every Technical Committee will define and carry out projects to model messages for use in its sub-domain.

Technical committees should give due consideration to each of the phases and models defined in this document, but not all of these are mandatory, nor is exhaustive detail required for each. However it is important to remember that each phase builds on the results of the previous phase. The HL7 Message Development Methodology presents the process of developing messages as a series of steps, but the actual job of message development will usually involve an iterative process. The number of iterations required depends on the size and complexity of the problem. The major steps listed for each phase provide a general guideline for creating the models defined within that phase.

Please note, that the purpose of this document is to present the development process for creating HL7 messages. The contents of this message development methodology are based on current notions of the best

³ Normally, a project will be taken on and administered by an HL7 Technical Committee. However, provided that is approval is received from the HL7 Technical Steering Committee, some modeling can be done by an HL7 Special Interest Group.

⁴ The HL7 Reference Model will include a consistent expression of the data used in HL7 messages, and of the use cases that those messages support. Refer to the HL7 Version 3, Statement of Principles for further discussion.

⁵ We have based this document on the work by various people including Ivar Jacobsen, James Rumbaugh and Grady Booch (as documented within the Unified Method), Peter Coad, and Ed Yourdon.

⁶ Insofar as it is significant to HL7 messaging.

⁷ To support the creation of HL7 Version 3, the first task of each Technical Committee will be to define its area of concentration by drafting a charter for review and approval by the Technical Steering Committee.

way to build applications and computer systems, in particular on the object oriented methodologies. For a detailed description of modeling methods, please consult the respective method textbooks. **However, the reader should remember that, for HL7, the “system” that is being defined is a group of messages that work together, not an end-user application.**

1.4 Why a Major New Version?

The original process for defining HL7 messages was established in 1987. It has served well since. However, as HL7 has grown in membership and its standards have become more widely used, HL7 has become aware of opportunities to improve the process and its outcomes. HL7 interface costs and implementation times are substantially better than the industry experience with proprietary interfaces. However, these costs and times vary considerably by vendor, and the industry sees a need for improvement. Because of substantial optionality in HL7, it is difficult to specify precise contract terms for HL7 interfaces. This can lead to unrealistic expectations that hurt vendors and buyers equally.

The size and complexity of the HL7 organization has grown considerably since the days when all the members of all the technical committees could sit together in a room. The number of specialized committees in HL7 and the size of each have grown substantially. These factors challenge the organization to provide improved “institutional memory” and more effective access to the most important shared knowledge.

1.4.1 Difficulties with the Existing Process

The process for building version 2.X messages is entirely *ad hoc*. There is no explicit methodology. Members receive no formal guidance in constructing messages. Trigger events and data fields are described solely in natural language. The structural relationships among data fields are not clear. Segments are re-used in many messages and message definitions are reused for many trigger events. In order to accommodate this extensive re-use, most data fields are optional. Chapters are inconsistent in their use of trigger events versus status codes. There is no specification as to when a specific kind of healthcare information system should be expected to honor a trigger event or accept a message.

With version 2.x, a technical committee creates messages by editing word processing documents directly. The metadata is not available in a structured form until the staff and volunteers tediously extract it from the word processing documents after publication.

In summary, there is substantial need to improve this eleven-year-old process in order to handle the breadth and complexity of the challenges HL7 faces today. The industry will gain if a new process supports specifications that are more rigorous.

1.4.2 Opportunities to Improve

Fortunately, software practitioners have learned a lot since 1987. There are better methodologies. Computer support is far more available and cost effective. However, HL7 cannot take advantage of these developments solely by minor tweaks to the old process.

HL7 spent four years characterizing its revised goals and creating a methodology to adapt modern analysis techniques from system building to message design. Initially the Executive Committee chartered an independent task force to establish the broad outlines of the approach. In January 1996, the Technical Steering Committee agreed to adopt the main features of the approach and take over its management. Planning work continued in the Modeling and Methodology Committee and the Control/Query Committee. At the completion of version 2.3, in the spring of 1997, the HL7 technical committees all began to use the Version 3 process.

Here are some features and benefits of the new approach:

- The process is based on an explicitly documented methodology supported by training classes, trained facilitators and computerized tools. This helps functional committees meet the challenges of *de novo* interface design, increased functional breadth, and evolution of assumptions. It helps new members become productive in fewer meetings. The majority of the development time is spent creating use cases and an information model using the Unified Modeling Language that is rapidly becoming an industry standard. This is an enormous aid to providing institutional memory and sharing work in progress across committees and to the membership at large.
- Conformance to HL7 Version 3 is specified in terms of Application Roles. These are abstractions that express a portion of the messaging behavior of an information system. A vendor of a healthcare application describes its conformance by asserting that it *completely* supports all trigger events, messages and data elements associated with one or more Application Roles. This level of specificity allows clearer pre-contractual understanding between vendors and buyers and will serve as a basis for conformance testing.

1.4.3 Optionality is a Four Letter Word

The limitation, if not elimination, of “optionality” is a primary goal of HL7 Version 3. Many of the constructs within this Message Development Framework were designed with that goal in mind. In current HL7 standards, almost every data field is optional. This optionality is necessary since optionality is defined by segment and segments are re-used in multiple messages. Declaration of a data field as optional within a segment allows that segment to be used in many different messages without any further definition.

However, the wide uses of optionality, while helpful in gaining consensus and in writing a somewhat more compact specification, imposes significant costs on interface implementers. In fact, optionality, as a “feature” of HL7 Version 2.n, is associated with most of the issues that Version 3 is trying to solve. Optionality makes it harder to precisely define the semantics of a specific message and makes it virtually impossible to generate conformance claims for messaging.

In Version 3, three steps will be taken to eliminate optionality.

1. Presence or absence of values is defined by trigger event rather than at the level of data definition. Within the RIM, there is no notion of optionality. Each class has a list of relevant attributes. Instead, the presence or absence of a data element value is defined for the Hierarchical Message Description (HMD) associated with a trigger event.⁸ This approach leads to more messages with more precise definitions.
2. The concept of “optionality” has been replaced by “Conditional”, a more rigorous alternative. The presence of one data field can be required or not allowed depending on the value of another field. For example “bed location” is required if visit type is “inpatient” or “day surgery.”
3. Message developers are urged not to use imprecision as a shortcut towards consensus. Instead they should explore whether disagreements over specific data structures should be resolved by creation of multiple trigger events or application roles.

1.4.4 Limitations of the New Approach

Nothing is free or perfect. The new methodology has costs and limitations, particularly related to development effort and complexity. These must be balanced against the benefits.

⁸ Some of the terms used here are new. The reader should take note of these, and press on. They are defined later in the document.

The version 2.x process is more direct than that of Version 3. To change a part of the 2.x standard, one simply edits the change into the appropriate word processing document. In Version 3 one makes changes in the computerized models, and then deals with the downstream consequences in the message structures. The disparity between the processes is most noticeable in introducing small changes into existing interfaces. When designing big changes or new interfaces, each method requires time to achieve joint understanding and arrive at consensus. The well-documented and facilitated approach in Version 3 is arguably faster than an ad hoc process employed by a committee that changes members from one meeting to the next.

HL7 has added resources to support this more intense process. These include recruitment of a cadre of trained modeling facilitators, provisions for additional training, extension of the Working Group Meetings to five-days, and establishment of interim meetings to harmonize the models developed during the Working Group meetings. HL7 also invests some of its budget in computer software and staff support to assist the version 3 process, and to pay the travel expenses of certain leaders and facilitators to the interim meetings.

1.5 Methodology Preview

This methodology defines the process of message development through generation of models and graphs during the following stages.⁹ Each stage has a specific focus, and is associated with a model or closely associated group of models.

Stage I	Requirements development starts by defining the scope for a new project, and then provides a description of the domain's business processes through development of the Use Case Model.
Message Requirements	The Use Case Model is built using use case diagrams to capture the project scope, and to allow full definition of the requirements the set of messages is designed to support. The Use Case Model provides a basis for assuring the quality of the later stages of message development, and for defining conformance claims for applications using HL7.

⁹ It is common to distinguish three views in the analysis process: **Function, Structure, and Behavior**. These views correspond to three models: A *functional model* deals with the responsibility aspect and reveals the intended purpose of a task. A *structural model* deals with the form and describes the objects and their relationships. A *behavioral model* deals with procedural aspect and represents the potential behavior of an application or application component in terms of the actions it takes and the states that it enters over time. It also deals with dynamic interactions across the application border—scenarios.

<p>Stage II Message Contents</p>	<p>Structural analysis focuses on the Information Model, which defines the data that messages will carry and that analyzes the state transitions for those classes and subject classes that play a central role in message development. The Information Model is built using class diagrams, and state transition diagrams. Creation of the Information Model provides for consistent and clear definition of the contents of a message or a group of related messages.</p>
<p>Stage III Messaging Behavior</p>	<p>The Reference Information Model (RIM) contains the information model contents for HL7. Each Technical Committee takes responsibility for the contents of the RIM through its participation in the harmonization process and its modeling activities. The Technical Committee will construct a domain information model containing the information needed to support messages within its domain. This domain information model is used as input to the harmonization process that introduces modifications to the HL7 RIM. Use of the HL7 Reference Information Model as the source of the content of messages provides for a more efficient message development process. The harmonization process helps ensure that the RIM contains the concepts that have been developed and refined by the HL7 Working Group.</p> <p>Development of the state transition model for subject classes, those classes that play a central role for messaging, forms a crucial connection between the work at this stage, and the investigation of messaging behavior in the next phase.</p>
<p>Stage IV Message Specification</p>	<p>Behavioral analysis focuses on the Interaction Model, which defines the specific interactions (information flows) that are needed to support the functional requirements. Interactions identify the trigger events that cause information to be exchanged and indicate the situations in which messages that will be needed. These definitions are also used to allow specification of HL7 conformance claims.</p> <p>The Interaction Model is built using the definition of application role classes, a tabular interaction grid, and sequence diagrams to describe the specific interactions (information flows) and application roles needed to support messaging requirements. The interactions provide a template for the flow of messages between HL7 using applications. Conformance claims are structured so as to satisfy use cases and scenarios. The conformance claim will allow an HL7 using application or system to specify clearly its implementation of HL7.</p> <p>Specification involves using the models created during the previous stages to create message specifications that precisely define HL7 messages.</p> <p>The Message Information Model (MIM) is a subset of the Reference Information Model that contains the data relevant to a message or group of messages. The Hierarchical Message Description (HMD) provides a tabular representation that shows the attributes that are included in each message and defines presence or absence of message components for each trigger event. The Implementation Technology Specification (ITS) precisely defines the representation of HMD contents in a specific messaging syntax.</p>

Figure 1-1 Four Stages of Message Development

There are four types of models that are created or modified in the course of message development. This document organizes the message development process around those models, as indicated in the diagram below.

The HL7 Message Development Framework

Phases, activities, and models

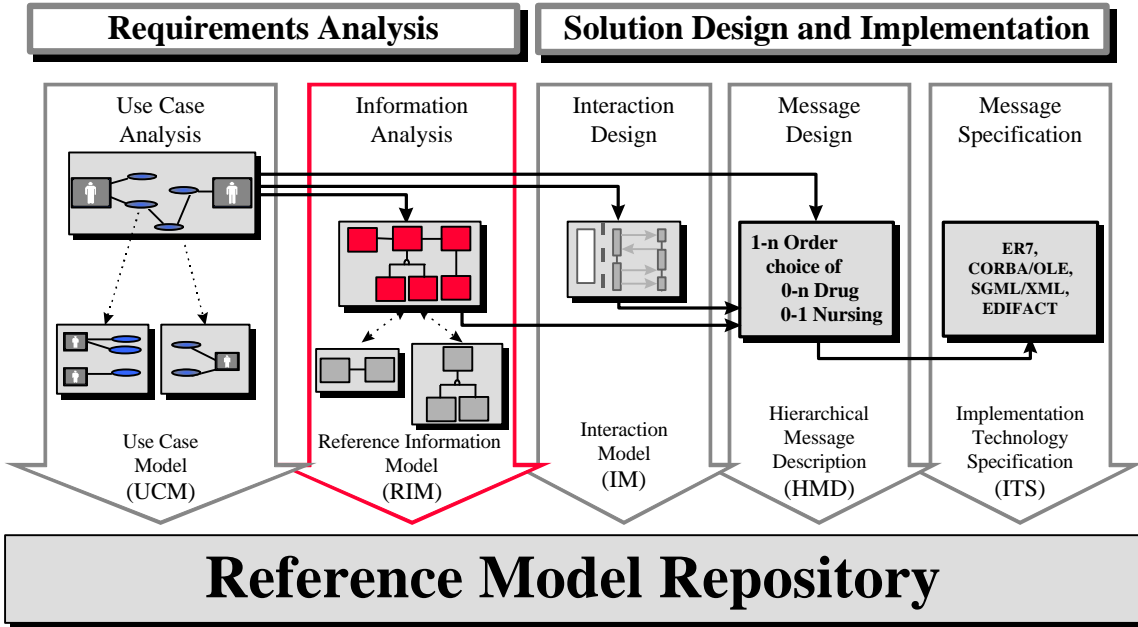


Figure 1-2. HL7 Message Development Process Model

The project life cycle diagram indicates in a linear fashion how a Technical Committee moves through the stages of developing new messages. This diagram shows the Information Model being built before the Interaction Model, even though both are contained in the same stage of message development.

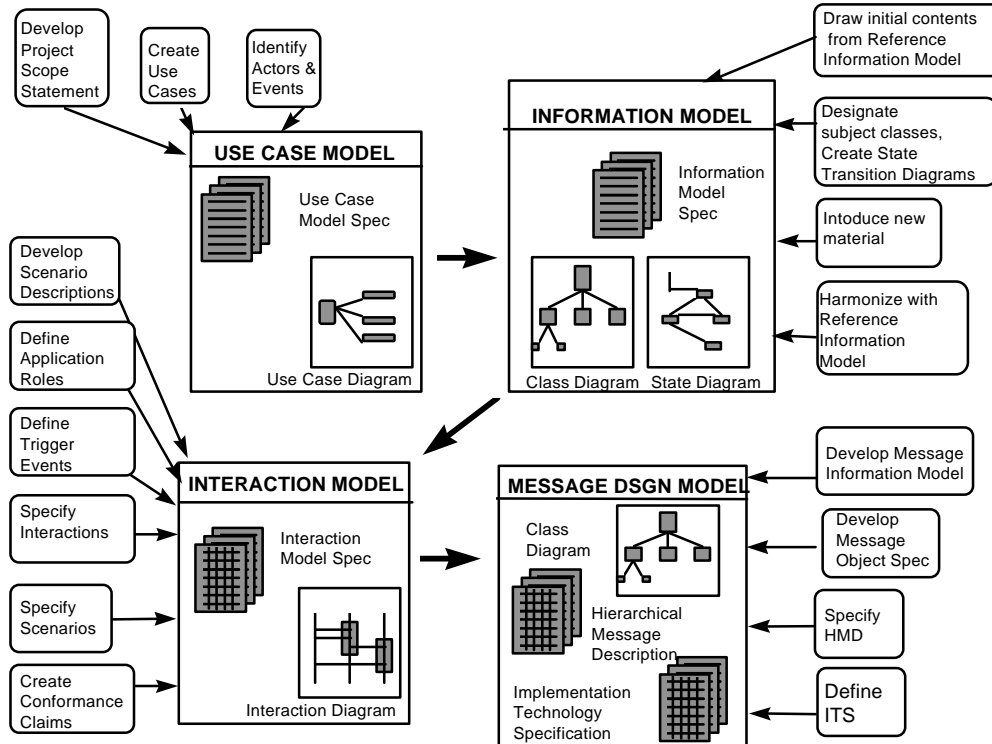


Figure 1-3. Message Development Models

The diagram highlights required steps in the process of designing a message.

The process defined by this Message Development Framework provides for:

- precise definition of the functional requirements that were considered by the message developers
- consistent data definition across the entire standard
- rigorous definition of the contents of each message that is defined
- conformance claims for use by standards implementers that increase the degree of interoperability between applications that make use of HL7

1.6 Document Structure

This Message Development Framework guides the Technical Committee project team through the process of developing HL7 messages. The sections that comprise the MDF are described below.

- **Chapter 1: Introduction** provides an overview to the HL7 Version 3 message development.
- **Chapter 2: Glossary of Version 3 Terms and Acronyms** contains definitions of the key terms that are used within this document.
- **Chapter 3: Principles of Version 3** discusses the premises for Version 3 and reviews the key criteria for success in developing the new version.

- **Chapter 4: Managing Message Development** defines the organizational structures that HL7 has created in order to manage version 3 and to guarantee the development of high quality messages.
- **Chapter 5: Use Case Model** provides the modeling tools to develop the requirements for HL7 messages.
- **Chapter 6: Information Model** provides the modeling tools to define the information used in HL7 messages, and discusses the principles behind the HL7 Reference Information Model (RIM).
- **Chapter 7: Associating Vocabulary Domains With Attributes, Elements, and Fields** discusses the principles for the use of controlled vocabularies within HL7 messages, and indicates procedures that will assist HL7 technical committees in applying these principles.
- **Chapter 8: Interaction Model** provides the modeling tools to define the trigger events and the interactions (or communication needs) for an HL7 message. It also provides tools to define the roles played by senders and receivers of HL7 messages.
- **Chapter 9: Conformance Claims and Organizational Practices** discusses the process by which vendors and users of HL7 systems can discuss conformance to HL7.
- **Chapter 10: Creating Message Specifications** defines the procedures that HL7 Technical Committees will follow in building HL7 messages. It explains how the Use Case, Information, and Interaction models provide the raw material for messages, as well as how that raw material is converted into the finished message.
- **Chapter 11: Developing HL7 Models Using UML and Rational Rose** describes the tooling that HL7 has provided to assist HL7 technical committees and Special Interest Groups in modeling and message construction. It also provides instructions for using these tools.
- **Chapter 12: Example model for MDF** is included within the document to serve as an illustration of the components of the message development process. Pointers to the relevant part of this model are included within the contents of each section.
- **Chapter 13: Specification of the HL7 MDF Components** provides a formal specification for most of the constructs that this document is based on.

1.7 Methodology 2000: Messaging and Beyond

Although HL7 was created as a standard for exchanging messages between healthcare applications, today's HL7 organization is working in a number of areas that fall beyond that scope. The following activities are included:

- Specification of software components by the Visual Integration Special Interest Group
- Direct use of the contents of HL7 models by the Clinical Decision Support Technical Committee which is exploring using the RIM as a solution to the "curly braces" issue raised by use of the Arden Syntax.
- Development of a framework for interoperable Document Type Definitions (DTDs) for use in healthcare. Creation of an architecture for management of clinical documents.
- Specification of application responsibility beyond simple message exchange. This theme is explored by the Accountability, Quality, & Performance Special Interest Group, and is inherent in aspects of the Interaction Model discussed within this document.

To respond to this increase in scope future releases of this document will directly address these issues. In effect, the “Message Development Framework” will morph into an “HL7 Development Methodology” that will seek to address the full range of issues raised by the need for interoperability between independently developed software components in the healthcare environment.

1.8 References

- Data Modeling Framework, Joint Working Group for a Common Data Model
- HL7 Version 3 Statement of Principles, HL7 Version 3 Taskforce
- Method for the Development of Healthcare Messages, CEN TC 251
- Object Oriented Software Engineering, Ivar Jacobson, et. al.
- Object Oriented Analysis, Peter Coad & Ed Yourdon
- Open EDI Reference Model, ISO/IEC CD 14662
- Unified Method, Version 0.8, Grady Booch & James Rumbaugh
- UML Distilled, Martin Fowler with Kendall Scott

2. Glossary of Version 3 Terms and Acronyms

Italicized terms in the definitions are references to other terms defined within this glossary.

Term	Meaning
Application	1. An act of applying or putting to use; a use to which something is put (e.g., “application of HL7”). 2. A software program or set of related programs that provide some useful healthcare capability or functionality.
Application Role	A characteristic of an <i>application</i> that defines a portion of its HL7 interfaces. It is defined in terms of the <i>interactions</i> (messages) that the role that the role sends or receives in response to <i>trigger events</i> .
Association	A type of relationship. An association relationship depicts a reference from one class to another class or itself.
Association composition	See <i>composite aggregation</i> .
Association role name	An association name. Associations have two names, one on each end of an association. Each name is an association role name. The name is a short verb phrase depicting the role of the class on that end of the association.
Association, Mandatory	A mandatory association is an association with minimum multiplicity’s greater than zero on both ends.
Attribute	Attributes are abstractions of the data captured about classes. Attributes capture separate aspects of the class and take their values independent of one another. Attributes assume data values that are passed in HL7 messages.
Attribute Type	The last part of an attribute name (suffix). Attribute type suffixes are rough classifiers for the meaning of the attribute. See also <i>Data Type</i> for contrast.
Bag	An unordered collection of items which need not be unique.
Base Class	A class that was modified to create a <i>Clone Class</i> in a <i>Refined Message Information Model</i> .
Cardinality	The number of elements in a set. See also <i>multiplicity</i> .
Choice	See <i>Choice Message element Type</i>
Choice Message Element Type	A message constructed that includes alternative portions of the message.
Class	1. A kind or category of things or concepts. 2. A definition of <i>objects</i> , in terms of properties (attributes, methods, and relationships) that all objects of the class have in common.
Clone Class	In a <i>Refined Message Information Model</i> , a substitute for a class in the <i>Message Information Model</i> . The clone is used to document constraints associated with a specific name.
CMET	See <i>Common Message Element Type</i> .

Collection Data Type	A data type that represents an aggregation of data elements, all of which are represented by one other data type. The forms of collection are <i>Set</i> , <i>Bag</i> , and <i>List</i> .
Collection Message Element Type	A message element type that can include multiple occurrences of some other message element type. Collections are declared as <i>sets</i> (unordered, no duplicates), <i>lists</i> (ordered, duplicates allowed), or <i>bags</i> (unordered, duplicates allowed).
Common Message Element Type	A work product that defines data types and <i>message element types</i> that becomes available to be incorporated into <i>Hierarchical Message Definitions</i> .
Composite Data Type	This is a type assigned to a message element type which type contains one or more components, each of which is represented by an assigned data type.
Composite Message Element Type	A message element type that contains a list of other, heterogeneous message types.
Composition Aggregation	A type of relationship. In a composite aggregation relationship the source and destination classes are associated as whole to part.
Conformance (column of the HMD)	A column wherein a technical committee states whether a message element must always be included in an HL7 message. The committee may say that it is required (must be included), optional (may be left out) or not permitted (may never be included.) Contrast this with <i>inclusion</i> .
Conformance Claim	A specific claim that is written by HL7 to precisely define the behavior of an <i>application</i> with respect to its HL7 interfaces. Functional conformance claims are simply a statement that an application conforms to an <i>application role</i> . Technical conformance claims are called <i>Technical Statements of Performance Criteria</i> . They define the behavior of an application in some other sense than the messages it sends or receives. These may include the <i>Implementation technology Specifications</i> that it supports, the use of specific optional protocols or character sets, or many other behaviors.
Conformance Claim Set	A list of the identifiers of specific HL7 <i>conformance claims</i> , used by a <i>sponsor</i> to describe the conformance of its <i>application</i> .
Constraint	A definition of the domain of an attribute as a subset of the domain of its data type. Constraining implies restricting a domain as defined by a data type or an attribute of a higher level model (RIM > MIM > HMD.)
Data Type	The type of a data field. Data types may be primitive, composite or collection data types. They define how a data field will be represented in messages. A message element type definition (MET) will be provided for each data type.
DIM	See <i>Domain Information Model</i> .

Domain	1. The problem or subject to be addressed by a set of HL7 messages or by a system (“application domain”.) See also <i>domain expert</i> , <i>domain information model</i> .) 2. The set of possible values of a data type, attribute, or data type component. Any domain is a subset of the domain of one data type. The domains of data types can be restricted through <i>constraints</i> . 3. More specifically “vocabulary domain:” the set of coded concepts that are acceptable for a coded message element in a specific message element type. See <i>Domain Specification</i> .
Domain Expert	An individual who is knowledgeable about the concepts in a particular problem area within the healthcare arena and/or is experienced with using or providing the functionality of that area.
Domain Information Model	A working model specific to a project used by a technical committee to capture information requirements. The DIM begins as a subset of the RIM. Changes applied to the DIM are forwarded as change proposals for the RIM.
Domain Specification	The specification of a vocabulary domain, i.e. the specification of the applicable coded concepts for coded message element instances. See <i>domain</i> (3.)
Encoding Rules-7	The <i>Implementation Technology Specification</i> that describes how to send HL7 version 3 messages using streams of printable characters.
ER7	See <i>Encoding Rules-7</i> .
Event	A stimulus that causes a noteworthy state change of an object. A signal that invokes the behavior of an object. See also <i>Trigger Event</i> .
Event Class	An event class represents an activity that occurs at a location, at a date and time, for a duration, involving one or more participants, for a reason.
Feature	A <i>property</i> of a class or object.
Function Point	Any function, user transaction, or other interaction or event in the <i>sponsor’s application</i> which, when it occurs, does or may correspond to an HL7 <i>Trigger Event</i> . Used to describe the conformance of an information system with the HL7 standard.
Generalization	1. The act of generalizing, i.e. defining a general concept or class that subsumes one or more special concepts or classes. 2. A so generalized concept. Antonym: <i>specialization</i> ; see also <i>inheritance</i> .
Graphical Expression	An expression of a model that uses graphic symbols to represent the components of the model and the relationships that exist between those components.
Hierarchical Message Definition	The work product that defines HL7 message types. A single Hierarchical Message Definition (HMD) describes one or more <i>message types</i> . An HMD includes the message elements, the conditions under which the message elements will be present or repeat, and their definition in terms of their relationship to elements of the <i>Message Information Model</i> .

HL7 Concept ID	A unique identification assigned to a concept by HL7. There is only one ID for a concept although it may be related to numerous <i>surface forms</i> that are code values that represent the code or textual explanations of the concept.
HMD	See <i>Hierarchical Message Definition</i> .
Implementation Technology	A method of encoding and sending HL7 messages. Version 3 implementation technologies will include <i>ER7</i> , object-oriented, and, perhaps, EDIFACT.
Implementation Technology Specification	A specification that describes how HL7 messages are sent using a specific <i>implementation technology</i> . It includes, but is not limited to, specifications of the method of encoding the messages, rules for the establishment of connections and transmission timing, procedures for dealing with errors.
Inclusion	The specification in the <i>Hierarchical Message Description</i> of whether an element of a <i>message type</i> may be null in some message instances. Contrast this with <i>conformance</i> .
Information Model	A structured specification of the information requirements of a project. An information model expresses the classes of information required, and the properties of those classes, including attributes, relationships, and states.
Inheritance	The fact that a specialization (subclass) has all the properties of its generalization (superclass.) See also <i>properties</i> .
Interaction	An element of the <i>Interaction Model</i> that is the confluence of the sending <i>application role</i> , the receiving application role, and the <i>trigger event</i> . It specifies the <i>message type</i> , <i>preconditions</i> and <i>post-conditions</i> for sending the message and <i>receiver responsibilities</i> .
Interaction Model	The component of the Message Development Framework that defines the specific <i>interactions</i> (information flows) that are needed to support the functional requirements defined within the <i>use case model</i> . <i>Application roles</i> , <i>trigger events</i> , and <i>scenarios</i> are also defined within this model.
Internal Data Type	An HL7 data type defined to support the definition of other data types, but which may not be assigned as the type for a data field in a Hierarchical Message Description.
ITS	See <i>Implementation technology Specification</i> .
Leaf Level Use Case	A <i>use case</i> that has no subordinate use cases
LIFO	See <i>pushdown stack</i> .
List	A collection of elements whose members are ordered.
Literary Expression	An expression of a model in text. The literary expression balances the dual needs for a rigorous, unambiguous expression of the model and for a rendition that can be easily read and interpreted by individuals who understand the general concepts underlying object-oriented models, but who may not be schooled in formal model definition languages.
Mandatory	See <i>inclusion</i> .

Mandatory association	An association with a multiplicity minimum greater than zero on one end.
Mandatory, fully	An association with a multiplicity minimum greater than zero on both ends.
Message Design Model	The component of the Message Development Framework that, based on the <i>Use Case Model</i> , the <i>Information Model</i> , and the <i>Interaction Model</i> defines the format of HL7 messages. <i>Message Information Models</i> , and <i>Hierarchical Message Definitions</i> are defined within this model.
Message Development Framework	The collection of models, methods, and tools that comprise the methodology for specifying HL7 Version 3 messages.
Message Element	A unit of structure within a <i>message type</i> .
Message Element Instance	An element within a <i>message instance</i> .
Message Element Type	A portion of a <i>message type</i> that describes one of the elements of the message.
Message Information Model	A subset of the reference information model specific to the information content of a set of <i>message types</i> . The MIM may specify constraints on the information that exceed those specified in the RIM.
Message Instance	A particular message that is sent. Two messages may be described by the same <i>message type</i> and identified with the same <i>interaction</i> and yet vary in the instance because of variations in values, presence or absence of the data being sent and different cardinalities of collections. Otherwise identical messages may be different instances if they are sent at different times or by a different sender.
Message Trace Diagram	A diagrammatic representation of a <i>scenario</i> .
Message Type	The organization of message elements that is specified in a Hierarchical Message Definition. A message type in effect constitutes a set of rules for constructing a message given a specific set of instance data. It also defines the rules for parsing a message to recover the instance data. The message type is independent of the Implementation technology Specification, so that if the same data is sent using the same message type and different implementation technology specifications it will be possible to transliterate from one to the other.
Meta-model	A model that includes types whose instances are also types. Meta-models are often used to specify other models. For example, the meta-model for a relational database system would specify the types 'Table,' 'Record,' and 'Field.'
MIM	See <i>Message Information Model</i> .
MIM Walk	A portion of the method for constructing <i>Hierarchical Message Definitions</i> from a <i>Refined Message Information Model (R-MIM)</i>
Model	A representation of a problem or subject area that uses abstraction to express the relevant concepts. A model is often a collection of schema and other documentation.

Multimedia-enabled free text data type	The data type used to capture free text and all data that is primarily interpreted by humans using rendering agents that are out of the scope of HL7. Supported media types include plain text (characters,) HTML, XML, and word processor documents, but also audio, images, and other types of multi-media data.
Multiplicity	1. In the information model multiplicity is a specification of the minimum and maximum number of objects from each class that can participate in an association. Multiplicity is specified for each end of the association. 2. In the <i>HMD</i> , multiplicity depicts the minimum and maximum number of occurrences of a message element expression in a collection.
Not Permitted	See <i>conformance</i> .
Null value	A family of values that can appear in message instances that indicate that data is not present in the message instance; the variations describe alternate reasons that the data is not present.
Object	1. A thing or concept that can be perceived . 2. An instance of a class. 3. A part of an information system containing a collection of related data (in the form of variables) and methods (procedures) for operating on that data. The term is used inconsistently in the literature, referring sometimes to instances and other times to classes.
Object Identity	The feature that the existence of an object is independent of any values associated with the object.
Object Request Broker (ORB)	A software mechanism by which software components interchange requests and responses.
Object-Based	Any method, language, or system that supports object identity, classification, and encapsulation. An object-based system does not support specialization. Ada is an example of an object-based implementation language.
Obsolescent message type	A <i>message type</i> that has been marked for deletion in a future version of HL7. In a subsequent release of the standard it may be declared an <i>obsolete message structure</i> .
Obsolete Message type	A <i>message type</i> that has been completely replaced with another, and is no longer a part of the standard. Before reaching this state it will have been declared an <i>obsolescent message type</i> .
Optioaliy	See <i>inclusion</i> .
Predicate Statement	A statement that is either true or false
Post-condition	A <i>predicate statement</i> describing the end state of classes affected by a used case.
Precondition	A <i>predicate statement</i> describing the required states of classes for a use case that enables the use case to be performed.
Primitive Data Type	Is a data type that is defined as a single entity, and whose full semantic is contained in its definition.

Predicate Reference	In the <i>Hierarchical Message Description</i> , a message element that is referred to in the predicate defining the conditional presence of another message element.
Primitive Message Element Type	A message element type that contains a single datum. Examples include String and Number. All other message element types are combinations of message element types.
Property	Any attribute, association, method, or state model defined for a class or object. Properties are the subjects of class inheritance.
Push-down Stack	An information structure for which the last entry added is always the first element removed. This is also known as a “last in-first out” (LIFO) list.
Receiver Responsibility	An obligation on an Application role that receives an <i>interaction</i> as defined in the <i>interaction model</i> .
Recursive Message	A message type where an object view contains itself. This can occur when the <i>MIM walk</i> hat builds a message leads from a class back to the same class.
Reference Information Model	An information model used as a reference for project-specific <i>Domain Information Models</i> . The RIM is a composite of project specific <i>DIMs</i> with <i>DIM</i> content conflicts resolved.
Reference Model	The model which serves as the authoritative repository of the data, use cases and other items that have been developed by HL7.
Refined Message Information Model (R-MIM)	A special version of a message information model that is used to describe constraints on a Message Information Model that will be applicable to one or more <i>Hierarchical Message Definitions</i> .
Reflexive association	An association between a class and itself.
Required	See <i>inclusion</i> .
RIM	See <i>Reference Information Model</i> .
R-MIM	See <i>Refined Message Information Model</i>
Role class	A class that depicts a role assumed by a stakeholder, person, or organization.
Role name (of an association)	See <i>Association role name</i> .
Root Class	The class that is represented by the <i>Root Message Element</i> .
Root Message Element	The message element that is the basis for the hierarchy that is the entire message.
Scenario	A statement of healthcare relevant events defined as a sequence of interactions. The scenario provides one set of interactions that the modeling committee expects will typically occur in the domain. Usually, a sequence diagram is constructed to show a group of interactions for a single scenario. Each scenario is displayed as a subset of the interaction model.
Set	An HL7 data type that contains an unordered list of elements of some other single data type. A form of a <i>collection message element type</i> .

Specification	A description of a problem or subject that will be implemented in a computer or other system. The specification includes both the description of the subject and aspects of the implementation that affect its representation. Also, the process of analysis and design that result in a description of a problem or subject which can be implemented in a computer or other system.
Specialization	1. The act of specializing, i.e. defining a special concept or class subordinate to a general concept or class. 2. Also specialized concept. Antonym: <i>generalization</i> ; see also <i>inheritance</i> .
Sponsor (of an <i>Application</i>)	The vendor, in-house developer, or provider of public domain software for a healthcare information system.
State	A property of an object that characterizes a step in the its life cycle. All states are represented by <i>state flags</i> . An object can be in multiple partial states simultaneously. An object has always one joint <i>state</i> , i.e. the combination of all partial states effective at a particular time.
State Attribute	An attribute of a subject class used to specify the joint state of an object. In a message, the state attribute is a set of state flags each representing currently active partial states.
State Diagram	A graphical representation of a state transition model showing states as vertices (nodes) and transitions as directed arcs (arrows) between the nodes.
State Flag	A discrete value of a single enumerated domain of partial states. State flags are included in a <i>state attribute</i> in a message instance that indicates the joint state of an object.
State Transition	A recognized transition from one state of a class to another state or back to the same state. These transitions are closely associated with trigger events.
State Transition Model	A specification for the life cycle of a class.
Statement of Conformance Criteria	A statement that describes the specifications for conformance to some specific aspect of an HL7 specification.
Steward committee	The technical committee with primary responsibility for specifying properties for a collection of classed in the RIM. The steward committee must be consulted on any changes to the properties of classes under its stewardship.
Stewardship representative	An individual member of the steward committee, authorized by the committee to speak for the committee and to represent the interest of the steward committee.
Storyboard	A statement of healthcare-relevant events defined as a sequence of interactions or use cases. The storyboard provides one set of interactions that the modeling committee expects will typically occur in the domain. Usually, a sequence diagram is constructed to show a group of interactions for a single storyboard. A storyboard may also be displayed as a sequence of use cases.
Subclass	A class that is the <i>specialization</i> of another class and inherits properties from that other class (<i>superclass</i> .) Antonym: <i>superclass</i>

Subject Area	A convenient aggregation of model classes used to partition large models into manageable subsets
Subject Class	A kind of Class that has been designated by a Technical Committee as interesting, because it is the focus of a set of use cases and/or trigger events.
Sub-state	An identifiable <i>state</i> of a class that has a more specific definition than and is entirely encompassed within the scope of its <i>super-state</i> .
Subsume	1. To include or place within something larger or more comprehensive (e.g., individual person and organization are subsumed under the concept of stakeholder) 2. A transformation on attributes and associations of a class in a <i>Refined Message Information Model</i> , wherein they are move to specialization of the class, and the general class is removed from the <i>R-MIM</i> .
Superclass	1. A class which has <i>specializations</i> , i.e. that is the <i>generalization</i> of one or more other classes (sometimes called “parent”.) Antonym: <i>subclass</i> .
Super-state	A <i>state</i> of a class that encompasses two or more independent <i>sub-states</i> .
Surface Form (of a concept)	A code value or textual description that represents a concept identified by an <i>HL7 Concept ID</i> . There may be many different surface forms associated with the same concept ID.
Technical Statements of Performance Criteria	See <i>Conformance Claim</i> .
Trigger Event	An event within the processes of healthcare which, when recorded or recognized by a healthcare information system, creates the need for an information flow to one or more other healthcare information systems. The information flow may be implemented by one or more <i>interactions</i> . Trigger events are closely associated with <i>state transitions</i> and <i>leaf level use cases</i> .
Type	A specification that limits the form that a message element may take. The general concept applies equally well to the types of message elements that represent attributes of the RIM (formerly called data types) ad types of large message elements analogous to segments, segment groups, or the entire message in HL7 version 2.x..
Unified Modeling Language (UML)	A (mainly) graphical language that is used to express object-oriented information relationships and designs. The UML is a standard of the Object Management Group.
Use Case	A description of a process by which actors do things that lead to the need for information interchange. Use cases may break down into component use cases. A use case may appear as a component in several other use cases. When a use case contains component use cases, the order in which they occur is not specified. See <i>scenario</i> .
Use Case Model	The component of the Message Development Framework that includes definition of the scope of an individual project, and, through development of <i>use cases</i> , provides a description of the project’s business processes. <i>Actors</i> and candidate <i>subject classes</i> are also developed within this model.

User

1. In the context of conformance claim, the organization that uses an *application*. This is frequently the buyer but in some cases the user and *sponsor* organizations may have be parts of the same organization or otherwise have a business relationship other than vendor-buyer. 2. In the context of *use cases* a person who interacts with an *application* to either enter or obtain information.

Vocabulary Domain Specification

A column in the *Hierarchical Message Definition* that specifies the *Vocabulary Domain* associated with a specific, coded data field. The column contains the identifier of a vocabulary domain that determines the valid concepts.

3. Principles of Version 3

3.1 Scope, Target Users

Version 3 shall be a standard for exchanging messages among information systems that implement healthcare applications.

3.1.1 Internationalization

Version 3 shall be developed to permit HL7 Affiliates to use it or to develop localized variants.

3.1.2 Support for Legacy Systems

As with prior versions, Version 3 shall be designed using a technological approach that permits implementation in "legacy systems." These are systems that have been implemented in technical environments that do not necessarily conform to or provide support for recent or pending "open systems" standards, such as those published by the International Standards Organization, Open Systems Foundation, Object Management Group. Similarly, HL7 will not require proprietary features of any operating system or software vendor. In practical terms, this means that Version 3 shall be able to exchange messages formatted in strings of printable characters, as is the case for all previous versions.

This does not preclude HL7 from deciding to develop variants of its specification that make use of the modern technologies provided that

- System builders will not be required to buy software from a sole source in order to implement Version 3, and
- Messages generated in these formats have the same data content so that translation between the printable character format and other formats is easy.

3.2 Loosely Coupled Systems

As with prior versions of HL7, Version 3 shall not be a standard for the functionality of the systems which exchange HL7 messages. However, where Version 3 includes a requirement to accept or send certain data or to send specific messages in response to trigger events or other messages, the application system may have to develop functionality to support those requirements.

3.2.1 Modes and Topologies

Version 3 messages may be sent using many modes and topologies. Messages may be sent in a manner that requires an immediate response, as unsolicited updates through a store and forward network, or in batches where the manner and timing of message transfer is not specified. Version 3 includes support for "one-to-many", store-and-forward distribution of messages by an external software agent.

HL7 does not require a specific mapping of messages in a one-to-many environment, but the Version 3 notion of application roles strongly suggests one useful paradigm. When a trigger event occurs in a system that is fulfilling one application role it will frequently have an obligation to interact with multiple systems that implement different application roles. The sending system can send a single union message that contains the information for all the application roles that are on the network. These union messages are candidates for one-to-many distribution.

3.3 Inter-version Compatibility

3.3.1 Compatibility with Versions 2.X

The goals of Version 3 cannot be achieved while maintaining full compatibility with previous versions. However, Version 3.0 shall be developed to cover the information content of the last version in the 2.x series including attributes and trigger events. This should not be construed to mean that all attributes and trigger events shall be included in 3 in exactly the same form as within the Version 2 series.

A network that includes a mixture of systems that implement Version 2 and Version 3 will require message translation. Because the Version 2 standards have substantial optionality, the translations will use rules specific to the systems on that network. It is expected that interface engines and other translation software will be able to provide the translations between site-specific interpretations of 2.x and any implementation of Version 3.

3.3.2 Compatibility Among Versions 3.X

The goals for upward compatibility in Version 3 are:

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 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.
- 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.
- 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.
- 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.
- 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.
- The obsolescent message structure may be declared obsolete and dropped when still another HL7 version is issued
- 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.
- 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. To the maximum degree possible, these restrictions should not impose limitations on the evolution of the overall reference model. 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.

3.4 Determining Conformance

3.4.1 Application Role

An Application Role is a named description of a domain-specific way in which a healthcare information system may interact with others through HL7 Messages. The names of the Application Roles should be different than the names generally used to describe healthcare information system products, because they will be more fine-grained. A typical healthcare information system will probably fill several or many such Application Roles. {For example, the Functional Committees may define Application Roles such as "lab order sender", "lab order filler", "enterprise patient identification store", etc.)

All sequences of Messages in Version 3 shall be related to a Trigger Event. The Trigger Event shall be clearly identified in a common field in all Messages.

3.4.2 Conformance Claims

These Roles shall be a basis for conformance claims. In making a conformance claim, a system developer shall identify the application roles to which it wishes to claim conformance. For these application roles, the Version 3 specification will state directly the trigger events the system shall recognize, messages that the system shall send in response to trigger events or other messages, and the data content of these messages. The specification also states the messages that a system conforming to the application role shall receive and process.

Other conformance claims shall be created by HL7 to describe the ability of an information's system to conform to non-functional aspects of the specifications. These may include specific message encoding formats, communications paradigms, or measures to ensure the integrity and confidentiality of messages.

All conformance claims shall be sufficiently specific to be used as contractual terms in system acquisitions and to be the basis for testing by an independent testing organization.

3.4.2.1 User Requirements with Respect to Conformance Claims

User sites may contract with vendors to implement deviations from the HL7 conformance claims. Such arrangements are outside the scope of HL7. Failure of a vendor to agree with such an arrangement will not be considered a failure of its application to conform to HL7.

3.5 Confidentiality and Security

3.5.1 Confidentiality of Patient Information

It is expected that healthcare application systems that implement Version 3 will be required to have significantly more functionality to protect the confidentiality of patient information than has been common in the past. The new functions may include, but are not limited to, limiting the right to view or transfer selected data to users with specific kinds of authorization and auditing access to patient data. Version 3 should contain the necessary data objects, attributes and transaction contents to support conveying the necessary information from one healthcare application system to another, so that these systems may perform the confidentiality functions. The Functional Committees, the Control Group, and the Modeling and Methodology Committee shall all consider these issues while developing the HL7 Data Model and Version 3 messages.

3.5.2 Authenticated Authorization for Services

It is expected that the healthcare application systems that implement Version 3 will be required to have significantly more functionality to authenticate requests for services and reports of data than has been common in the past. The new functions may include, but are not limited to, electronic signature and authentication of users based on technologies more advanced than passwords. Version 3 should contain the necessary data objects, attributes and message contents to support conveying the necessary information from one healthcare application system to another, so that these systems may perform the authorization and authentication functions. The Functional Technical Committees, the Control Group, and the Modeling and Methodology Committee shall all consider these issues while developing the HL7 Data Model and Version 3 transactions.

3.5.3 Security, Privacy, Non-Repudiation and Integrity

It is expected that the technological platforms upon which Version 3 information systems developers implement applications that use HL7 will be required to have significantly more capability to protect the confidentiality and integrity of patient information than has been common in the past. The new functions may include, but are not limited to, public- and private-key encryption, and correspondent system authentication and non-repudiation. The Control Group should monitor these developments to see that Version 3 can be implemented on technological platforms that support these new functions.

4. Managing Message Development

This chapter documents the procedures used to develop Version 3. These procedures are based on the Version 3 Methodology, and on the HL7 Bylaws. The goal is to assure that the standard is developed expediently, and that it is appropriately documented, consistent with the requirements for an approval by a balanced consensus, and that it complies with the requirements for certification by the American National Standards Institute.

4.1 Project Scope Definition

When a functional technical committee undertakes a project to develop new normative material in Version 3 it shall create a brief description of the project for approval by the Technical Steering Committee. This project scope statement¹ will also be used for coordination of projects with other standards organizations. In fact, project scope statements must be reported to the American National Standards Institute (ANSI) so that ANSI can fulfill its standards coordination role. The project scope statement shall contain a concise description of the needs to be met by the transactions.

Project Scope Statements should be reviewed by the Architectural Review Board² and must be approved by the HL7 Technical Steering Committee.

When the work product is published as a draft or for ballot, the project scope statement shall be published with it. HL7 shall maintain and publish from time to time a list of all project scope statement that have been approved but for which the corresponding messages have not been approved.

If, in the course of a project, the technical committee finds its work is exceeding the bounds of the authorizing project scope statement it shall submit an amended statement for approval.

4.2 Version 3 Methodology

The development of Version 3 Messages will follow the methodology specified by the HL7 Modeling and Methodology Committee, and documented within this Message Development Framework. It may amend the methodology from time to time. Included within the methodology will be a definition of work products that will be delivered by the technical committees. Portions of these work products shall be designated for three different treatments:

1. included as a normative portion of the eventual standard,
2. included as an informative portion of the eventual standard,
3. not included with the standard, but published with minutes and archived.

The methodology shall meet the following requirements. These are annotated to indicate the work products described in the Message Development Framework that meet the requirements. The annotations further state whether the work product is normative or informative.

¹ The project scope statement should not be confused with the Technical Committee or Special Interest Group charter to which it is related. The TC or SIG charter defines the scope of an HL7 committee and is subject to approval by the HL7 Technical Steering Committee. The key criteria for a project scope statement is that it should be consistent with the charter of the committee that works on it.

² The role of the Architectural Review Board is discussed below.

Requirement	MDF Construct(s)	Level
A means of providing context to the definitions of Trigger Events	Use cases, state diagrams	Informative
A means of specifying the information content of the Messages through a common information model that clarifies the definitions and ensures that they are used consistently across all Version 3 Messages defined by all Technical Committees	Reference Information Model	Informative, but may be subject to a formal comment process
A means of specifying responsibilities of the senders and receivers of Messages	Interaction model	Normative
A common description of the exact fields of a message and their grouping, sequence, optionality, and cardinality	Hierarchical Message Definitions	Normative
<p>Separate syntax specifications, describing the algorithms used to encode and transmit the messages in various implementation technologies including (a) an XML based character stream syntax (b) an object-oriented representation of the messages; and, (c) a printable character stream syntax similar to the current HL7 specification.</p> <p>The various syntax specifications for a message shall not depart from the common description of its contents, so messages in the different formats shall be functionally interchangeable.</p>	Implementation Technology Specifications	Normative

The Modeling and Methodology Committee shall not have the right to determine the contents of work products created by the technical committees. It shall serve to facilitate their development, and to negotiate changes among the functional technical committees to ensure standard-wide consistency. If disputes cannot be resolved by the technical committees and/or the Modeling and Methodology Committee, they shall be resolved by the Technical Steering Committee with the approval of the HL7 Board of Directors.

4.3 Document Structure

The structure of Version 3 standards documents has not been determined at this time. When it is defined, all normative content will be available in a machine-readable form that is structured for easy assimilation into databases, software configuration tables, or automatically generated software.

4.4 Data Field Domains

To the maximum extent practical, the normative specifications shall include precise definitions of the values that a domain may take on. The means of specifying this may include algorithms or enumeration of values. Enumerated values shall be identified with a unique concept ID that can be related to one or more sets of codes. To the maximum extent possible, domain specifications shall make use of published authoritative sources of code values and contents. To the maximum extent possible, coded data fields shall include information to support the systematic release of revisions to the code set.

4.5 Quality Assurance Processes

In order to ensure the highest quality of the work products produced during the course of Version 3 message development, the HL7 Board of Directors has created the Architectural Review Board.

The Architectural Review Board shall work with the Technical Committees and Modeling and Methodology Committee to define measures of quality of the work products of the standard. It shall also ensure that the Version 3 Process is explained in a manner that is clear and well understood by the membership.

The Architectural Review Board shall ensure that the work products are reviewed against such measures of quality, and that adherence to the defined message development process is documented in a manner consistent with HL7 bylaws and Procedures. The Architectural Review Board shall not have the right to amend work products or disallow their distribution. It shall have the right to include with any work product a statement describing areas where it has determined that the work product has not met established measures of quality.

Disputes with the findings of the Architectural Review Board will be resolved through the Technical Steering Committee and, if necessary, the Board of Directors.

4.6 Process Support

HL7 should provide reasonable funds to support the Version 3 processes including, but not limited to, acquisition and use of computer-based tools to support the development of work products, document conformance to process and create final deliverables. Appropriation of funds will be made by the Board of Directors in a manner consistent with the overall financial viability of the organization.

4.7 Management

The standards development process shall be governed as defined by the HL7 bylaws and Policies and Procedures. All development of procedures called for in this Statement of Principles shall be ratified by the Technical Steering Committee unless the bylaws or Policies and Procedures require different approval.

5. Use Case Model

5.1 Overview

Use Case analysis, first described by Ivar Jacobson in his seminal 1992 book Object Oriented Software Engineering, is a technique that enables analysts, users, and developers to discover and formally represent the essential aspects of a specific system's *boundaries* and *responsibilities*. What makes Use Case analysis unique is that the representation is directly derived from the perspective of the *user* of the system, each "use case" being an "instance of system usage" by one of the user's of the system-of-interest. In this context, the fundamental framework behind Use Case analysis can be viewed as consisting of three elements: 1) Identification of the *user* -- who may be either a person or another system -- as a discrete, identifiable entity that is *outside* of the boundaries of the system-of-interest; 2) Definition of a set of specific *system responsibilities* ("instances of system usage") where each responsibility is, by definition, executed *inside* the boundaries of the system-of-interest; and 3) Identification of *Products of Value* that are produced by the system-of-interest as a direct result of the "instance of usage," i.e., as the result of the fulfillment of a specific system responsibility. Use Case analysis refines the general notion of a 'user' to that of an 'Actor,' with Actors being formally defined as a system user playing a specified 'role' in defined 'usage contexts.' Thus, a given user may be represented as multiple Actors, and a single Actor may represent multiple users. System responsibilities are *named with a verb phrase*, and are executed as a direct result of an *interaction* between the system and a user. Thus, a concise definition of a Use Case incorporating the three described elements is thus stated as follows: *"A Use Case is an interaction between the system and an Actor that causes the system to fulfill a responsibility and, as a result, to produce a product of value for the Actor."*

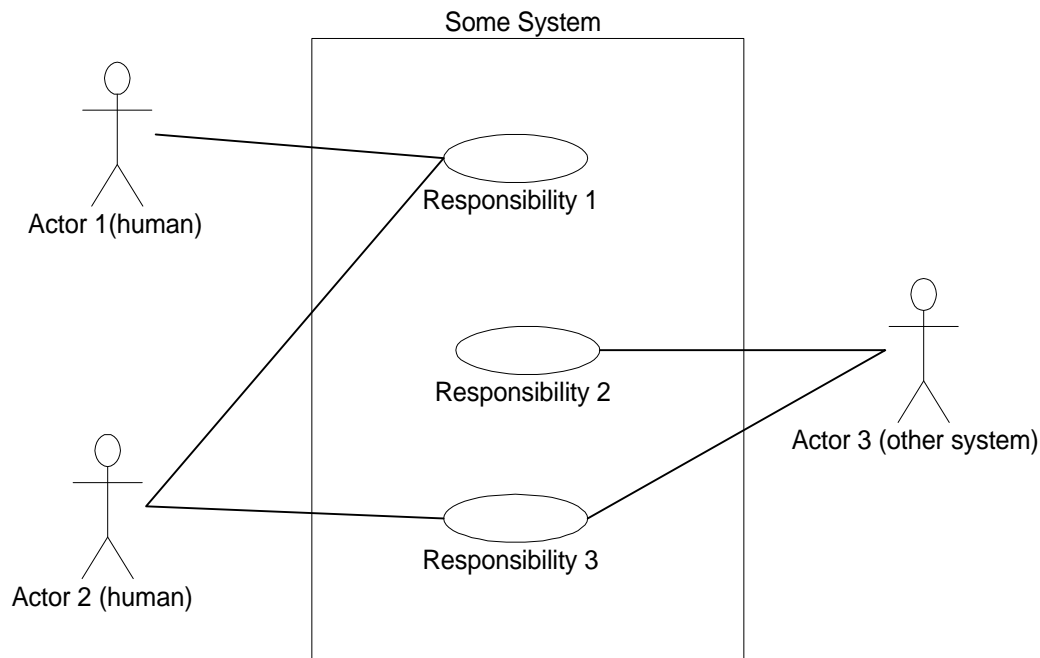


Figure 5-1. This figure shows the UML representation of a prototypical Use Case analysis defining the system boundaries and responsibilities for "Some System." Three Actors using "Some System" have been identified: two human users and one system. The Actors interact with Some System via three Use Cases, each of which represents a system responsibility. Not shown are the associated Products of Value produced for the Actor(s) by Some System as a result of the System's fulfilling of a named Responsibility.

Use Case analysis is, by definition, a highly *iterative* process of discovery. It is also by nature an *architectural* process since the notion of a "system" can be partitioned into subsequent layers of subsystems, subsystems within subsystems, etc. each of which has its own set of Actors and Responsibilities. Historically, Use Case analysis has been used with considerable success during the "requirements" or "specification" phase of development of complex information systems. In particular, by virtue of focusing, clarifying, and ultimately defining system boundaries and responsibilities in a graphic format, Use Case analysis becomes a highly effect tool for preventing the "scope creep" that often plagues the development of complex systems. In addition, the process of Use Case analysis also produces two additional "Products of Value:" a framework for developing system Test Plans, and the outline of the system's User Documentation. Test Plans flow naturally from Use Case analysis since each Use Case documents in detail *what a System is expected to do* to meet a particular responsibility. User Documentation is easily derivable from Use Case analysis by virtue of the fact that the collected Use Cases for a given system accurately describe *what a system does for its users*, i.e., what Products of Value can a user expect to receive from a System for a given interaction. Thus, the power of Use Case analysis is that the process produces a work product -- the complete Use Case model -- which can be viewed from three different points-in-time in the overall development lifecycle of a complex system (Figure 5-2).

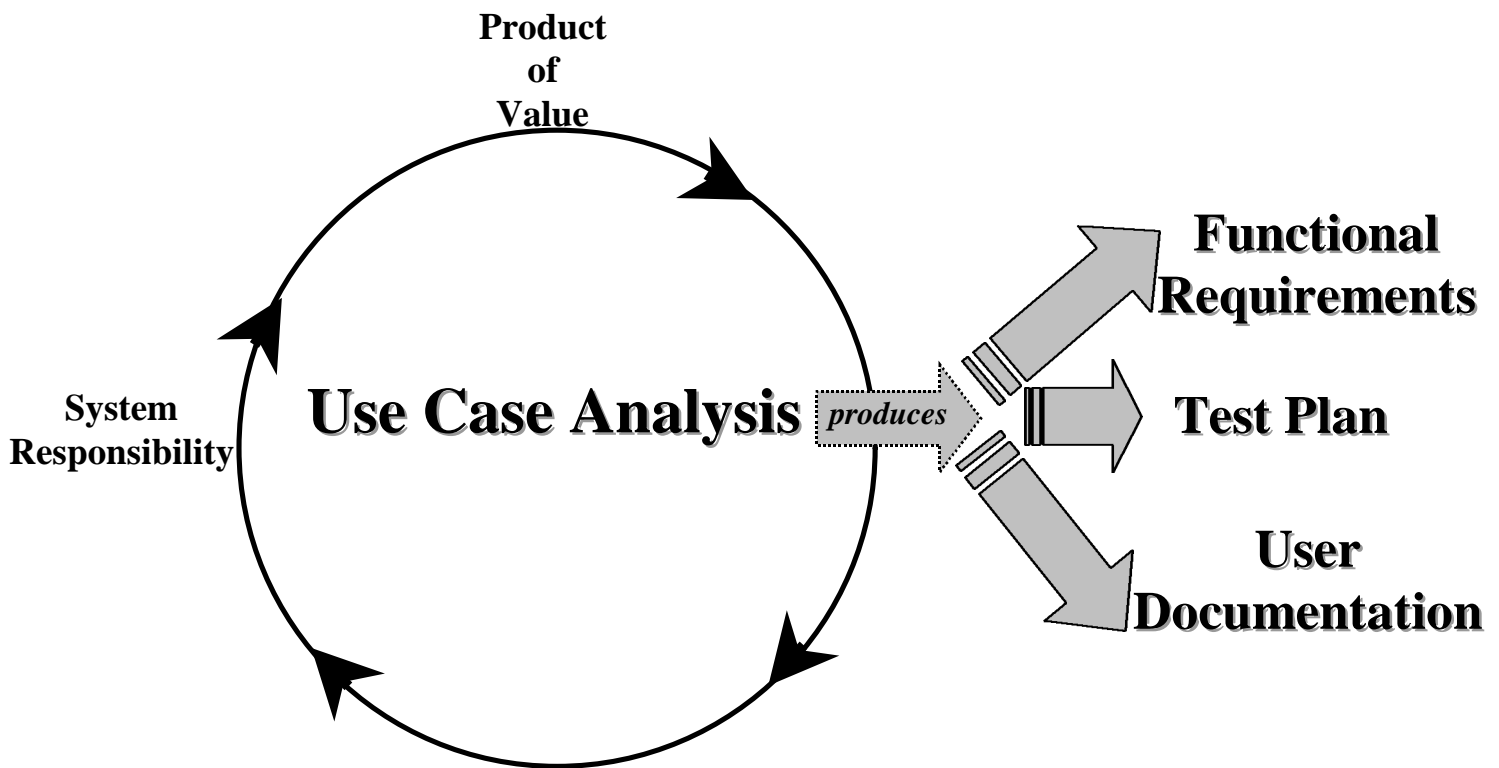


Figure 5-2. A schematic representation of the *iterative* process of Use Case Analysis for defining system boundaries and system responsibilities. System boundaries are defined indirectly through the identification of Actors who lie outside those boundaries. System responsibilities are identified by virtue of the Products of Value that the system produces as a result of an interaction between the Actor and the system. In particular, the Product of Value is the direct result of an interaction between an Actor and the system, and is produced as a result of system fulfillment of one of its responsibilities. The "Product of Value" of Use Case analysis itself, i.e., the collection of Use Case diagrams and associated documentation referred to as the "Use Case model," provides the system's Functional Requirements Specification, as well as a framework for Test Plans and User Documentation/Training materials.

As alluded to above, Use Case analysis is typically a "top-down" process. As the analysis proceeds and more details are discovered about various Actors' expectations for a system, one begins to "decompose" a given high-level Use Case into collections of "lower-level" Use Cases. *Because Use Case analysis is an architectural view of a system, Use Case "decomposition" does not proceed along traditional functional decomposition lines, but rather along lines defined by terms such as "collaboration," "realization," and/or "subsystem components."* One popular approach to this "layering" of Use Cases revolves around viewing a particular "parent" responsibility of a given "high-level" Use Case diagram as the "root" of a subsystem which contains the collection of Use Cases involved in actually fulfilling the named responsibility. This layering approach can easily be extended to multiple levels, with a "child" Use Case on one level acting as the "root" Use Case for a package/subsystem of Use Cases at the next "lower" level. *Note that the Actors identified at the "root" level may not be the Actors for subsystems, since the "system-of-interest" for a given Use Case diagram is now a subsystem within the original system.* (Figure 5-3)

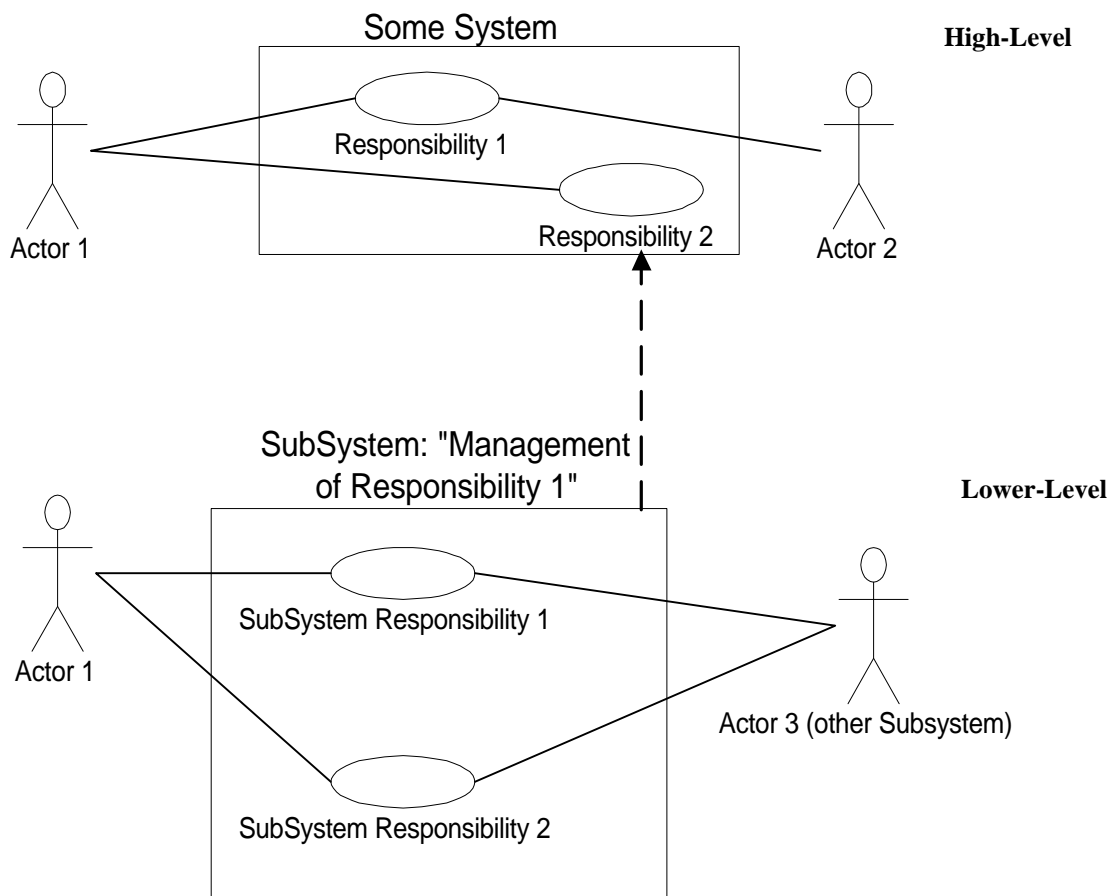


Figure 5-3 . An initial "high-level" Use Case analysis has revealed two System Responsibilities. Responsibility 1 has been "decomposed" into two "lower-level" Use Cases which are contained within a subsystem/package named with the *noun phrase* "Management of Responsibility 1." Note that as the decomposition of a given high-level system Responsibility occurs, the Actors involved with the lower-level Use Cases may not be the same as those at the higher-level. In particular, other packages representing different "higher-level Use Case Responsibility Management" subsystems may themselves become Actors in a given subsystem's Use Cases because the lower-level Use Cases in the subsystem-of-interest produce a Product of Value for the other subsystem.

5.1.1 HL7 Messages and Use Case Analysis

Historically, the principle task of HL7 has been to define the *structure* and *content* of the messages that must be exchanged between systems supporting the various activities of healthcare delivery. The HL7 message set must therefore include information regarding the pertinent details of various administrative, financial, and clinical activities. Clearly, the definition of such a message set must start with in-depth knowledge of the information that the users of the various healthcare information management systems need to have exchanged between systems. The CEN TC251 document, *Method for the Development of Healthcare Messages* states: "The study of the user requirements is the first major activity in the message development process. Based on real world scenarios, the information exchange needs are identified. The main success factor for this activity is the domain knowledge available to the development team."¹ This statement can be paraphrased to: "One needs to understand the problem that the system is trying to solve in general, as well as in particular how the users propose to use the system to solve the problem, and this understanding needs to be gained *before* one begins designing and building the system that will provide the solution to the problem."

The HL7 Version 3 effort has extended the notion of simply defining the "structure and content of the set of messages that must be exchanged between systems supporting the various activities of healthcare delivery." In particular, one of the essential driving forces in the Version 3 effort is to move the standard to a point where messages are not only specified in terms of their structure and data content, but also in terms of the larger context of "system conformance" (Chapter 9), i.e., the associated, message-specific *System Responsibilities* for a system claiming to send and/or receive a particular message. Complete definitions of both message content *and* system conformance will result in a marked decrease in the amount of "site specificity" encountered when interfacing/integrating two "HL7-compliant" information systems because the interaction between the systems can be approached in terms of *both* specific message content *and* associated HL7-compliant conformance responsibilities.

Because HL7 is concerned with defining a system of *messages* and not with specifying *application behavior*, the notion of "system conformance" might, at first glance, appear to be somewhat contradictory. However, Use Case analysis provides the necessary unifying framework. The output of the initial phase of Use Case analysis for the set of HL7 messages defines the "system-of-interest" as a virtual messaging system whose Actors are the various healthcare domain experts/users/other systems desiring to send and receive messages. The responsibilities of this virtual system consist of constructing and sending the appropriate messages. A high-level Use Case diagram of the virtual system is shown in Figure 5-4.

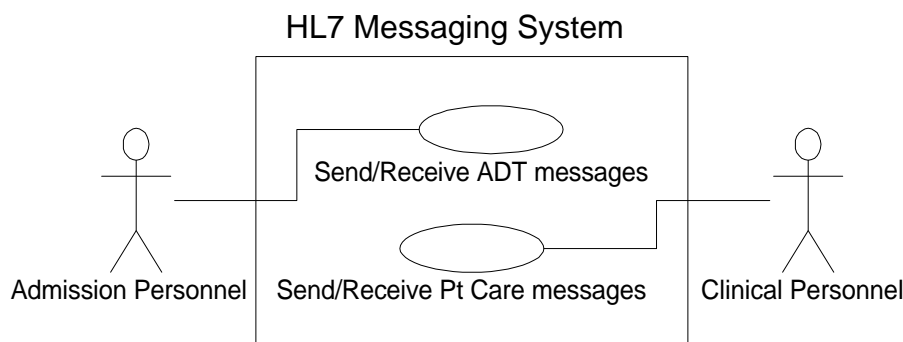


Figure 5-4. A simplified high-level Use Case analysis of a virtual HL7 messaging system. System responsibilities are partitioned by message type (and ultimately by message), while Actors are identified by area-of-interest. Each System Responsibility is fulfilled by assembling and transmitting (or receiving and interpreting) the appropriate message(s).

The obvious problem with the model in Figure 5-4 is that the system for constructing/sending and/or receiving/interpreting does not actually exist as a physical system with which real-world Actor can interact! To the contrary, the "real-world" of healthcare information systems is most often an unpredictable assemblage of multiple systems from multiple vendors, each with their own set of data collection, presentation, analysis, and/or transmission capabilities. Although it is certainly true that the real-world Actors shown in the Figure do *define* the semantics of the messages that must be sent and received, actual "HL7 messaging" is a "System Responsibility" of a *subsystem* within one or more physical healthcare information systems (Figure 5-5).

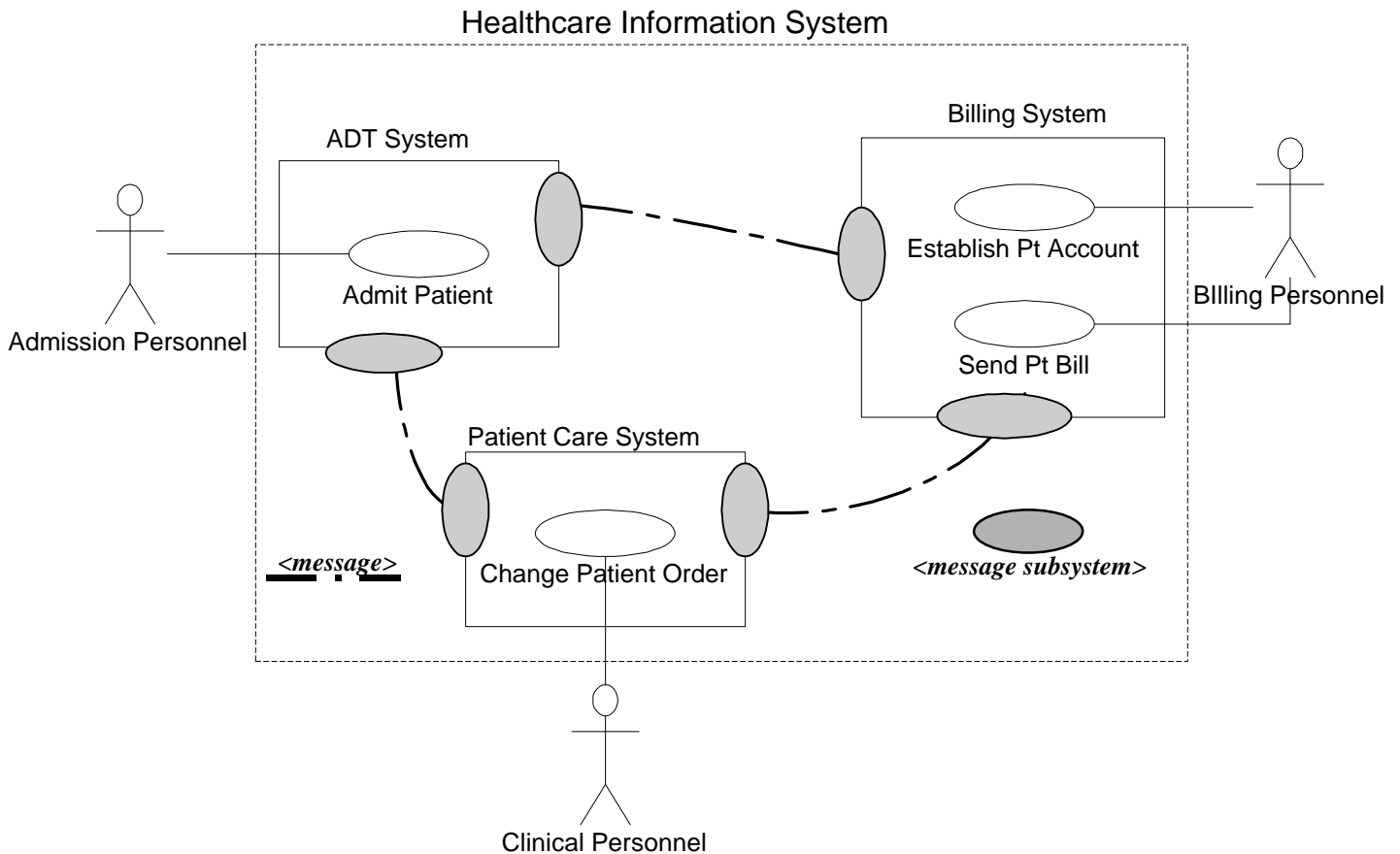


Figure 5-5. This figure shows a high-level Use Case analysis of a prototypical (real or virtual) healthcare information system. Users (Actors) perform various domain-specific functions using the "system," which is often an interfaced/integrated collection of heterogeneous systems (and/or subsystems.) Message traffic flows between the various systems/subsystems as a direct result of interactions between the various Actors and systems/subsystems. However, the Actors *per se* are not aware of the messages at the level of construction, transmission, reception, or interpretation. These System Responsibilities are instead handled by the internal messaging subsystems of each of the domain-specific systems/subsystems. A Use Case diagram of a messaging subsystem therefore has non-human Actors. The MDF refers to these Actors as "Application Roles." (See text and Figure 5-6 for explanation).

As alluded to in Figure 5-6, and further elaborated in Chapters 7 and 8 of this document, HL7 and the MDF has developed and adopted the concept of *Application Roles* to provide the necessary formalism for defining enforceable, message-specific conformance claims. From a Use Case perspective, Application Roles become the Actors for the "HL7 Messaging System." Unlike most Actors in Use Case analysis, however, Application Role Actors are "Actors with specific Responsibilities." As explained below and

elsewhere in this document, the documentation of an "atomic-level" HL7 Use Case is most often defined on a message-specific basis, and requires at least two Actors -- the Sending and Receiving Application Roles. The System Responsibility of the Use Case is the construction, transmission, reception, and interpretation of the specified message(s), and the Product of Value of the Use Case are the messages

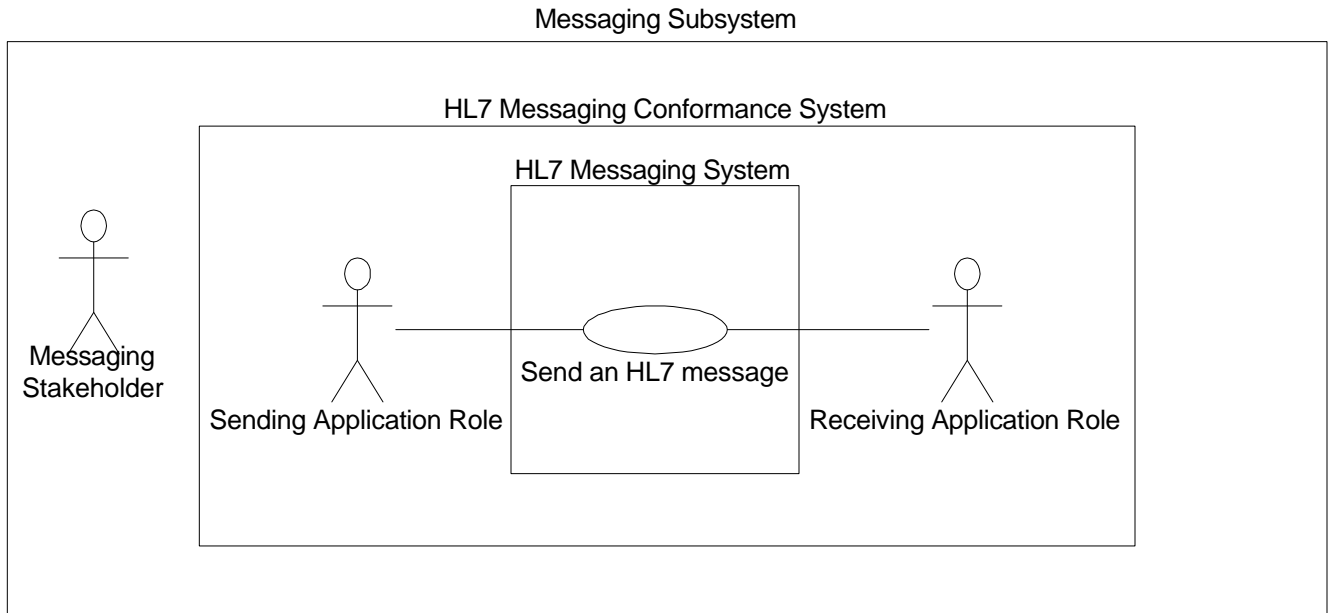


Figure 5-6. This Figure shows the Use Case relationship between the HL7 Messaging System and its Actors, the MDF-defined Application Roles, as well as the larger contextual relationship between the HL7 Messaging Conformance System as a set of responsibilities within a Messaging Subsystem (see Figure 5-6). The Actors receiving Products of Value from the HL7 Messaging Conformance System are noted simply as "Messaging Stakeholder"; however, their presence is meant to emphasize the layered architectural application of Use Case analysis in complex systems. Not mentioned are the Responsibilities of the Receiving Application Role ("Receiver Responsibilities," the MDF extension to the notion of Actor).

When studying Figure 5-6, it is important to note that it is the composite of the HL7 Messaging System and its associated Application Roles AND the additional documented "Receiver Responsibilities" that form the HL7 Messaging Conformance System which is the Product of Value of the MDF process. Thus, the system of HL7 messages and associated Application Roles and Receiver Responsibilities collectively fulfill a set of System Responsibilities within the context of the larger HCIS.

5.1.2 Use Case Analysis and Storyboards

Use Case analysis forms the foundation of the *formal* definition of HL7 v3.x messages. Furthermore, as noted above, domain-expert users of healthcare information systems should be the authoritative source as to both the context and semantic content of the messages defined by HL7. However, Use Case analysis is often not the best *initial* mechanism for discovering either the context or semantic content of a specific set of messages. A less formal method often referred to as "storyboarding" is often more helpful for two reasons: 1) Use Case models (i.e., collections of Use Cases) contain *no syntax for temporal sequencing*, i.e., no general structure or notation for specifying that "Use Case 1 must occur before Use Case 2." However, the healthcare domain experts who have the knowledge necessary for specifying message content and exchange requirements used to define and specify message context and content often find time-ordered descriptions to be the most natural expression of their domain knowledge; and 2) The actual users of healthcare information systems are normally not -- from the formal perspective of Use Case analysis -- the Actors in the Use Cases which fulfill the message-specific System Responsibilities of message construction, transmission, reception, and/or interpretation. Rather, these Actors are often other

systems/subsystems, the details of which HL7 does *not* want to specify, but whose general nature is captured in the notion of Application Roles. Storyboarding, the semi-structured process of collecting time-sequenced anecdotes or "stories" in a somewhat *ad hoc* fashion from domain experts, is often a more effective and thus essential prelude to formal Use Case analysis. In particular, storyboards, i.e., the documentation of simple narratives involving a series of interactions and/or message communications that "makes sense" to the domain expert, typically contain more than one Use Case and multiple Application Roles. As such, they serve as a valuable source of material from which individual Use Cases can be mined, i.e., explicitly extracted and formally modeled.

5.1.3 Use Case Analysis and the MDF

Within the context of the MDF, HL7 has made the decision to use Use Case analysis as its methodology for capturing user requirements. However, as explained in the previous Section, Use Case analysis is often most productively done as the first *formal* process to follow the more *informal* process of Storyboarding. When complete, Use Case analysis and the resulting set of Use Case models define the functional content of the message set being defined by the MDF process. (This step is often referred to as "functional analysis" in more traditional system-design parlance.). Use Case analysis, therefore, lays the foundation for the other models generated by the MDF process, and is thus of pivotal importance in producing Version 3.x messages.

Use Case analysis for the "HL7 Message System" begins with an understanding of the business or clinical processes which cause messages to be exchanged between applications at different sites or within different systems. When an end-user performs a business function using a particular application, that application may need to communicate with one or more other applications in order to accomplish the task or tasks required to support various business or clinical processes. The message-dependent details of these processes are gleaned from domain experts during one or more Storyboarding sessions. The specifics of the communicated content is defined, analyzed and described within a set of Use Cases and the associated Use Case model(s). The contents of these communications are the messages that are being sent and received, and define HL7's "healthcare-specific domain of interest."

The formal process of Use Case analysis (and modeling) begins with the structured definition of "high-level" Use Cases, and proceeds to the identification of one or more architecturally-intermediate levels of Use Cases. Analysis terminates when suitably granular "leaf-level" Use Cases have been identified. (More details about the specifics of leaf-level Use Cases are included in subsequent Sections of this Chapter, as well as in Chapter 8, Interaction Model.) *Note that as described in previous Sections of this chapter, the process of Use Case decomposition is **not** an instance of functional decomposition, but rather one of architectural decomposition.* In general, leaf-level Use Cases are associated with life-cycle changes in RIM Subject Classes (see Chapter 6, Information Model). These life-cycle changes, in turn, produce the Trigger Events that initiate the flow of HL7 messages. Although leaf-level Use Cases are often discovered in the course of Storyboarding, they may also be uncovered by analyzing the life-cycle (i.e., state changes) of RIM Subject Classes.

In the larger arena of object-oriented system analysis, each Use Case is documented using a combination of narrative text, structured associations of "Actor Actions" and "System Responsibilities," and one or more associated diagrams including Activity and/or Interaction diagrams. Within the context of the MDF, Activity Diagrams (not formally discussed as part of the MDF) are often helpful in structuring particularly complex Storyboards. Leaf-level Use Cases, on the other hand, can nearly always be adequately documented with carefully worded narrative text detailing the initiating trigger for the Use Case, a formal specification of the content the message, and the message's associated Application Roles and Receiver Responsibilities. Each Use Case is then "realized" with a Sequence Diagram (one of the two forms of Interaction diagrams) which details the specific message traffic between the Application Roles. If required for presentation to domain experts, collections of Use Cases are grouped together in categories, packages, or subsystems as explained in previous Sections of this Chapter. In this case, each of these logical organizational units fulfills one or more HL7 Messaging Conformance System Responsibilities. Collections of related Use Cases are often depicted graphically in a Use Case model. (As explained below,

Use Cases within a system / category / package / subsystem / collaboration may be *syntactically* organized in a Use Case model by using the formally defined UML <<specialize>>, <<include>>, and/or <<extend>> stereotype relationships.)

Once a Use Case has been defined, the details of the analysis can be studied to identify candidate domain concepts and objects, and their collaborations and interactions, (e.g., in healthcare, such concepts and objects would include problem, intervention, specimen, patient, provider, etc.) Objects of importance in the HL7 domain are ultimately be defined and represented in the Reference Information Model (RIM) (see Chapter 6), while the specifics of many of the interactions and collaborations will be detailed in one or more dynamic diagrams (e.g., Sequence and/or State diagrams) as explained in Chapters 6 and 8.

Figure 5-7 and Figure 5-8 show the UML iconographic representation of a Use Case model, the primary components of which are "Actors," "Use Cases," and "Associations." Note that each Use Case diagram represents the "System Boundaries and System Responsibilities" for a particular system, subsystem, package, or collaboration that is not directly named in the diagram. *Thus, by convention, the actual drawing of the system's boundaries (Figure 5-1, Figure 5-4, and Figure 5-7) is unnecessary.*

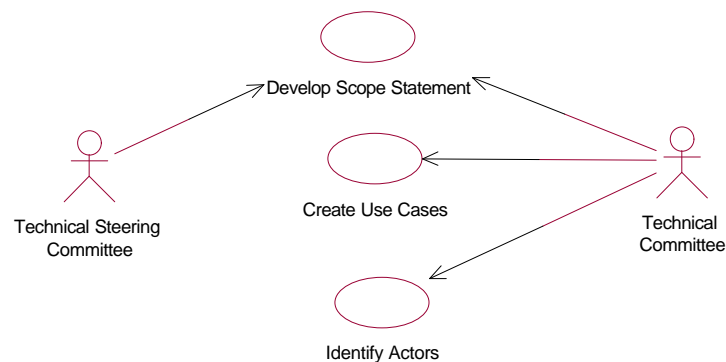


Figure 5-7. This Use Case diagram depicts the high-level Use Cases for the system "MDF Methodology," (or, more correctly, for the "Use Case Management" subsystem / package of the larger "MDF Methodology" system, i.e., for the "system" that results in the definition of Version 3.x HL7 messages. Actors are outside the system's Boundaries, while the system's (high-level) Responsibilities are depicted via the named ellipses. The *optional* direction association arrows between each actor and the named Use Cases indicate that in each actor/use case interaction, the interaction is initiated by the actor rather than the system (a direction of the association can be reversed in cases where the system initiates the interaction.) Note that both the Technical Committee and the Technical Steering Committee have an interaction with the use case "Develop Scope Statement" indicating that both Actors receive a Product of Value from the Use Case.

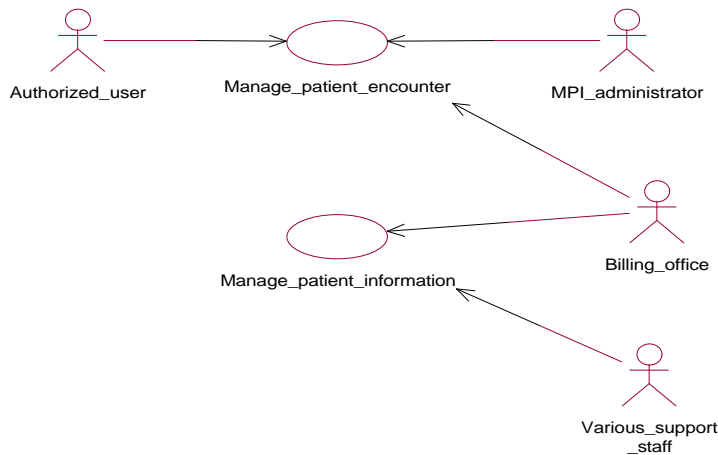


Figure 5-8. A Use Case diagram depicting the high-level Use Cases for the "Patient Management" system (or subsystem) from the Example Model in Chapter 12. All Use Cases are initiated by the associated actors.

In addition to specifying messaging requirements, the HL7 Use Case Model provides a medium for communications with users of the HL7 Standard. To quote Jacobson,

"Another important characteristic of the requirements model is that we can discuss it with the users and find out their requirements and preferences. The Use Case model is easy to understand and formulate from the user perspective, so we can easily talk to the users and see if we are building the correct {set of messages} according to their requirements. Since the Use Case model is the first model to be developed, we can evaluate whether the users are pleased with what we are about to design, before we start to build the actual system."

5.1.4 Actors

As mentioned previously, an Actor is a context-based user of the system-of-interest, i.e., the system whose Responsibilities are described by a collection of Use Cases. Actors also serve to help define a system's Boundaries since they are, by definition, *outside* the system-of-interest in contrast to the system's Responsibilities, which are, by definition, *inside* the system. Taken together, the identification of Actors and the specification of a system's Use Cases completely define that system's Boundaries and Responsibilities.

An actor may be either a human user or another system outside the logical (and often physical) boundaries of the system-of-interest. Actors interact with the system-of-interests to cause that system to produce a Product of Value for the Actor. In a typical information system, Actors interact with a system in a variety of ways including, e.g., start it, stop it, maintain it, change its state, put data in, and get data out. Individual Actors are named and defined in terms of the "usage context or role" that they assume when interacting with the system in the course of a Use Case. The same user may be represented by many Actors with each named Actor representing one of the many roles that that user can assume when interacting with the system. Alternatively, a single Actor may represent many different real-world users of the system, a fact that reflects that many users may interact with a system in the same role. Thus, the formal definition of an Actor is stated as follows: *an Actor defines/names "a user (human or system) of the system-of-interest interacting with the system in a particular usage context (role)."* As shown in several of the previous Figures, named Actors interact with the system by being "associated" with one or more named Use Cases.

In the context of HL7, an Actor is an entity that needs to participate in the communication of healthcare information, either as a sender or as a receiver of the information. Thus, there are various human Actors (i.e., domain experts) involved in interacting with a variety of healthcare information systems to cause those systems to send and/or receive HL7 messages. Likewise, a number of non-human (system) Actors

can also be identified early in the analysis process. When speaking about a system as an Actor, HL7 strongly suggests that the Actor simply be named "Some System" rather than being named more specifically, e.g., "The MPI System" or "the Radiology System." Names such as these carry strong connotations of associated functionality and thus depart from the scope of HL7 to specify only message content and context and *not* application functionality. On a more general level, one must always be careful in Use Case analysis in general (and in HL7 Use Case analysis in particular) to be sure that the identified Actors are, in fact, *directly* interacting with the system-of-interest (i.e., the set of HL7 messages) and are not, instead, one or more levels of indirection removed from that system. Indirect Actors can always be spotted by virtue of the fact that they do not *directly* receive a Product of Value from the system-of-interest, i.e., in this case one or more HL7 messages. Therefore, two important features of HL7 real-world Actors are of particular importance in the context of HL7 Use Case analysis. First, real-world Actors are most often identified in the course of Storyboarding and are not, in fact, Actors in the HL7 Messaging System, but rather Actors interacting with the larger Healthcare Information System. As such, they are not represented in HL7 leaf-level Use Case models. Second, the Actors depicted in HL7 leaf-level Use Cases are the Sending and Receiving Application Roles (see Chapters 5, 9) whose identification then defines the responsibilities of the HL7 Conformance Messaging System which subsumes the HL7 Messaging System (see Figure 5-6).

5.1.5 Storyboards, Scenarios, and Use Case Paths

In everyday language, the terms "story," "storyboard," and "scenario" are often used interchangeably. There is little question of the value of such "narratives" in collecting domain knowledge. The question is: "Do these narratives, these collections of domain knowledge, have a 'formal' name that can be used to distinguish them from other formalisms encountered in the course of use case analysis and modeling -- and, if so, what is that name?" Unfortunately, the answer is "There is, at present, no singly agreed-upon name."

Practitioners of methodologies for elucidating system requirements using less formalisms than is associated with Use Case analysis (e.g., CRC Cards) often use the term "scenario" synonymously with both "story" and "storyboard" to mean "*a semi-structured narrative, often relayed by domain experts, which traverses a longitudinal sequence of events.*" Prior to the release of the definitive UML specification, the term "scenario" was also often used in the context of Use Case modeling as a synonym for the collection of activities specified by one (or more) Use Case(s). Consider, for example, the following quote from CEN's TC 251 on Use Case modeling:

“The scenario (Use Case) specification:

- extends the high-level scenarios of the scope statement.
- gives examples of healthcare parties acting in the scenarios.
- considers variances of the message exchange patterns.
- describes real-world situations where the messages defined by the standard may be used, but may equally specify explicitly real world situations where the messages were not intended to be used, is of an illustrative nature, but the results are used for the formal specification of the normative components.”¹

In this context, the terms "scenario," "storyboard," and Use Case all appear to have the same meaning.

With the publishing of the UML Specification, the term "scenario" was *very specifically* defined to mean "*a single path of execution through a single Use Case.*" Thus, a given Use Case can be *realized* in a number of different execution paths, each represented by the traversing of a single execution path *within* the Use Case. Furthermore, this definition allows the term "storyboard" to revert to its less-formal meaning of "a sequenced collection of events-of-interest" where it can be used as a resource for, but is not a defining feature of, Use Case analysis.

¹ MDHM, Page 15

Because of the confusion associated with the terms, the MDF has chosen to not use the term 'scenario in any context, choosing instead to use the term *Use Case path* to describe the details of a particular activity flow within a use case. The MDF meta-model (Chapter 13) defines the relationships between Use Cases, Storyboards, and Use Case Paths.

In summary, the best way to think about a "Use Case Path" (or, outside of HL7 and the MDF, a "Use Case Scenario") is to remember the following analogy: *'A Use Case Path is to a Use Case as an Instance of a Class is to a Class.'*

5.2 Procedures

The Use Case model captures requirements for healthcare messaging by describing the business processes involved in communication of healthcare domain information between computer applications. The HL7 Modeling and Methodology Committee recommends that each Technical Committee follow the following procedures in building their Use Case model:

Inputs:

- A group of domain experts who are ready to work.
- A Facilitator skilled in guiding domain-specific knowledge discussions while simultaneously abstracting and modeling the domain knowledge so that it is both understandable to the domain experts, and structured in a fashion suitable for subsequent MDF analysis and modeling.
- Reference material that includes existing models and documents produced by other Technical Committees, and the HL7 Reference Information Model²

A list of suggested sequential steps is as follows:

Develop Project Scope Statement: Develop an initial problem description and create a statement that defines what is to be done. The Project Scope Statement can ultimately be structured as either a domain-specific Storyboard, or a high-level Use Case that encompasses the entire project domain (with the caveat that the Actors in the high-level Use Case will not be the Actors in the final Use Case model.) Once the Scope Statement has been written and agreed upon by the Technical Committee, it should be reviewed and approved the HL7 Technical Steering Committee, the HL7 body responsible for ensuring that Scope Statements are relevant, feasible, and unique. As the Technical Committee builds its Use Case model, the Project Scope Statement may be revised to reflect new insights and/or understanding of requirements. However, any substantial revisions must be reviewed by the Technical Steering Committee.

Identify Actors: It is often helpful to begin the process of Storyboarding and/or Use Case analysis by identifying the key stakeholders in the system-of-interest, i.e., in Use Case parlance, the Actors. Clear and consistent definitions of Actors -- *each* Actor should have a short, concise, and *clear* definition associated with its (noun) name -- result in consistent definitions of system boundaries. Conversely, the inability to define Actors in a clear and consistent fashion often indicates an unclear Scope Statement. Left unchecked, this deficiency can lead to unrelenting "scope creep" with the inevitable result of having a Technical Committee full of frustrated and unproductive domain experts.

Identifying Actors, particularly through the process of Storyboarding, is also quite helpful as a tool for Use Case discovery. Once an Actor is identified, write down the Product(s) of Value that the Actor expects / wants / needs to obtain from the system. (Remember, in HL7, the "system" is the set of messages so the "Product of Value" is the content of the message itself. Also, remember that when you get to the level of

² Over time, HL7 will develop a Reference Use Case Model that will provide a starting point for this process. That model is not in place yet.

the message, the Actors will be the Sending and/or Receiving Application Roles associated with the message (see Chapter 9)). Finally, write down a *verb phrase* that describes the Responsibilities (i.e., message content) that the system must fulfill in order to produce each Product of Value. In the world of HL7, this verb phrase will often be the action accomplished by the sending and/or receiving of the message.

The process of Actor identification is an iterative, knowledge-discovery process. If the Technical Committee finds that it is identifying too many Actors -- a fact which is most often indicated by insignificant differences between Actors -- it should attempt to abstract a few of the Actors' key characteristics "upward" as characteristics of an "abstract Actor," i.e., an Actor that does not represent a "real-world" user of the system, and start then start "working down" the abstraction hierarchy. Actor hierarchies are quite often discovered in the course of Use Case analysis, and can easily be represented using the UML "generalization/specialization" relationship icon in combination with the Actor icon, as shown in Figure 5-9. Real-world Actors -- both abstract and concrete -- will often be represented in the RIM as classes and/or role classes.

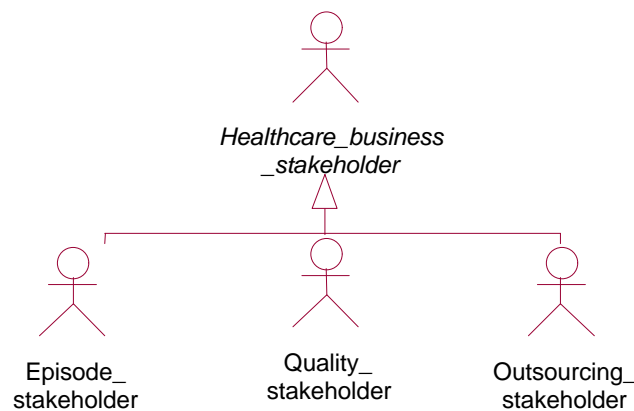


Figure 5-9. A typical Actor hierarchy. Note that the root of the hierarchy is an abstract Actor, i.e., an Actor that does not represent a real-world system user. Abstract Actors are named in UML using *italics*.

Identify and Document Use Cases: Work with domain specialists and existing standardization efforts to discover Storyboards and their resulting Use Cases-of-interest. The Scope Statement for the project should be considered as the root "high-level" Storyboard and/or Use Case for each Technical Committee. Other Use Cases are discovered and developed by expanding and more fully defining the content of the Scope Statement.). It is important to remember that the identification of Use Cases is often iterative.

It is often helpful to group together System Responsibilities for a given Actor. These collections of Use Cases can be organized to elucidate various types of functional/structural relationships between the Use Cases.

Once the discovery process stabilizes, each Use Case is described in more detail. These descriptions should not be developed too early, since the Use Case model typically undergoes several changes. When analyzing and describing Use Cases (or the Storyboards from which the Use Cases are mined), it is not unusual to uncover unclear expressions, ambiguities, or omissions in the expression of the scope statement. When this happens, the Scope Statement should be revised and reviewed with the Technical Steering Committee if necessary. Document each Storyboard and/or Use Case in sufficient detail to show the high level content of information flows and prerequisites for information exchange. HL7 Use Cases can usually be specified fairly completely in words. State the Use Case name, the Actors (Application Roles and Receiver Responsibilities if this is a leaf-level Use Case), and the Product of Value (i.e., the message semantics) of each Use Case. In addition, be sure to include the Use Case's preconditions, post-conditions,

initiating event (e.g., state change in a RIM Subject class), and a clear description of each path through the Use Case.

Associate Actors and Use Cases: Once you have defined a logical collection of Actors and their associated Use Cases, you can assemble a graphic representation of these elements in a Use Case diagram, i.e., a Use Case model. Be sure that each Use Case and Actor are involved in *at least* one Association relationship. Use the <<*specialize*>>, <<*extend*>>, and <<*include*>> dependency stereotypes as needed to syntactically link use cases (see discussion below for the semantics of these stereotypes).

Each time you complete an iteration of the above steps, you will come closer to producing a "complete" Use Case model for a domain-of-interest. Realize that a complete Use Case model will most often require multiple iterations of Actor and Use Case identification, documentation, and association.

Outputs:

- Approved statement of Project Scope Statement.
- Defined Actors (possibly organized in an Actor Hierarchy) that will interact with the system within the defined Scope.
- A set of documented Use Cases, each Use Case having been documented using the recommended graphic and literary expressions.
- A Use Case model which associates Actors and Use Cases.
- An initial idea as to candidate Subject classes for the scoped domain, i.e., those classes most important to realizing the message content specified in the Use Case model. (Subject classes are defined and discussed in Chapter 6, Information Model).

5.3 Documentation

UML specifies a simple diagrammatic technique for representing a Use Case model.³ Actors, be they human or system users, are represented by stylized human "stick figures" with the name of the Actor displayed at the Actor's "feet." (Note that in UML, an Actor is a stereotype of Class and can therefore be used in all syntactic constructions in which a Class object is valid. As a result, any iconographic representation that is valid for a UML Class is a valid representation of an Actor.) Use Cases are represented by an ellipse either containing or sitting above the name of the Use Case. The Use Case name should always be a verb phrase that describes the System Responsibility fulfilled by the Use Case.

Technical Committees should find that white board creation of Use Case diagrams is especially valuable in group brain-storming sessions. In general, Use Case documentation generally includes -- for *each Use Case* -- the following components: a) a clearly defined list of Actors (Application Roles), each of which either initiates the Use Case (or is notified of its initiation) (Sending AR). In addition, the Product of Value of the Use Case should also be identified; b) a clear definition of *each* path taken through the Use Case; c) a statement of the Use Case's pre- and post-conditions; and d) further detail, as needed, to indicate the Use Case's dependencies on/syntactic associations with other Use Cases.⁴

³ See OOSE, Section 6.4, Page 126 - 130, also UML, Chapter 4, Section 3.

⁴ Refer to the example model included within this document for use case examples. Also, the Formal Specification of the HL7 MDF Components, contains further discussion on the graphical and literary expression of use cases.

5.4 Tutorial Suggestions and Style Guide

The process of Storyboarding and the resulting Use Case model lay the foundation for defining HL7 messages and their content. It is the starting point of message development, and serves the purpose of clearly defining the scope of the set of messages being defined. In general, it is important to strike a balance between laying a firm foundation and getting bogged down in the initial stages of the process. In some cases, if the scope of a project is clearly understood, as would be the case if the Technical Committee was "retrofitting" existing Version 2 messages into a Use Case model, it is possible to omit construction of a Use Case model altogether.

5.4.1 Project Scope Statement

Scope definition is the starting point for a project. It defines the area of healthcare functionality that is supported by the set of messages to be developed. The Project Scope Statement must be communicated to both the HL7 Architectural Review Board and the Technical Steering Committee for its review. Review of the Project Scope Statement should focus on confirming the need for HL7 standard messages within the project scope, evaluating the priority to be given to the project as compared to other possible projects, and on ensuring that the Technical Committee or Special Interest Group submitting the Project Scope Statement is, in fact, the appropriate group to be responsible for the defined scope.

As an overall guideline, a new project should take up a current need for standardizing information that flows between a number of parties in healthcare. It is also relevant to consider whether or not the scope of a proposed project is too broad, since this can lead to the Technical Committee (or SIG) getting bogged down. The result will inevitably be frustration, inefficiency -- and no message development. Finally, both the Technical Committee and the Technical Steering Committee should consider whether the Technical Committee has sufficient resources to complete the projected scope. Finally, it is essential that the Architectural Review Board and Technical Steering Committee be careful so as not to approve overlapping Project Scope Statements, since multiple efforts may lead to multiple solutions and resulting effort expended in solution harmonization.

5.4.2 Use Cases

In general, a Use Case should define a specific behavior, function or System responsibility within the boundary of the Project Scope Statement. A Use Case should describe the activity or activities that are carried out by the system in order to meet the Use Case's Goal, i.e., to produce the Use Case's Product of Value. The Use Case should either begin as a the result of an initial stimulus from one of its associated Actors, *or* should notify at least one of its associated Actors as its first action in the case where the Use Case is self-initiated. (NOTE: Self-initiated Use Cases are common in association with System Responsibilities based on time clocks.) Following the Use Case's Initial Event, a Use Case proceeds through a series of interactions between one or more Actors and the healthcare environment. The Use Case ends when the Product of Value has been delivered to the Actor(s). In the context of HL7, leaf-level Use Cases are initiated when a Trigger event is received by a Sending Application, and terminated when the Receiving Application Role has executed its Receiver Responsibilities

When developing Use Cases, the HL7 committee should look for Use Cases that allow a particular Actor to carry out a well-defined task or activity, i.e., to send or receive a well-defined message or message set. This guideline will help ensure that the Use Case is neither too large nor too small. An HL7 Use Case should always focus on the need for communication of healthcare information between independent parties/applications/systems within the healthcare environment.

Use Cases may be expressed at various levels of abstraction. A Use Case may be a "parent" to several "child" Use Cases. *However, if the relationship between a single Use Case and several other Use Cases is truly one of parent/child, each child Use Case must meet the following criteria: a) it must add to or*

override the behavior of the parent Use Case; and b) it must be able to be substituted in any situation when the parent Use Case was used.

Often it is the case, in the context of Use Case analysis, that the terms "parent" and "child" do *not* carry the hierarchical semantics associated with the UML <<*specialize*>> association relationship. Rather, a parent Use Case may be diagrammed to be "realized" by a "collaboration of child" Use Cases. Equivalently, the parent Use Case may become the marker for a system, while the child Use Cases make up the System Responsibilities of one of the system's subsystems.

At the lowest level of decomposition, each leaf-level use case should involve an association with only one Actor of type *Sending Application Role*, but may be associated with several Actors of type *Receiving Application Role*. The Use Case should be executed in response to a *single* state transition for a subject class.⁵ If the execution is linked to multiple Trigger events, transitions and/or Sending Application Role Actors, further decomposition should be considered.

Try to avoid making a given Use Case too complex. A complex Use Case should be decomposed into multiple related Use Cases using the techniques described in the next Section.

Remember that, for HL7, the "system" being defined is a group of messages that work together, not an end-user application. Leaf-level Use Cases are directly associated with Trigger events for messages. Once Use Case development has reached the point of discovering messages triggered by state transitions / life-cycle transitions in a RIM Subject class, **stop**. *You do not need to decompose the Use Cases any further.*

Storyboarding and Use Case analysis are the initial steps in the MDF methodology and are the least formal. Their semantics are somewhat loose, and their detail somewhat sparse, facts that are consistent with their purpose of being the "first-cut" at system requirements. Use them for what they are good for and do not expect them to provide information they cannot. Sequence Diagrams, the RIM, State Diagrams, MIMS, MODs, and HMDs all add information -- but all are grounded on *accurate* Use Case analysis.

5.1.1.1 Organizing Use Cases

As the Use Case models developed by HL7 Technical Committees (or SIGs) get relatively large, it will be useful to group Use Cases into logical packages, subsystems, and/or collaborations. Logical groupings should be chosen that are meaningful to the Technical Committees and other potential users of the Use Case model. Logical partitioning often involves the use of three UML <<*stereotype*>> dependency associations between Use Cases:

- <<*specialize*>> (Use Case 2 adds interleaves additional behavior into that of Use Case 1);
- <<*include*>>(Use Case 1 uses Use.Case 2 as part of its execution);
- <<*extend*>> (Use Case 2 adds additional behavior to Use Case 1 at a specified Variation Point)

⁵ A subject class is a class within the Information Model that is expected to have special relevance for messaging. See the Information Model chapter for further discussion.

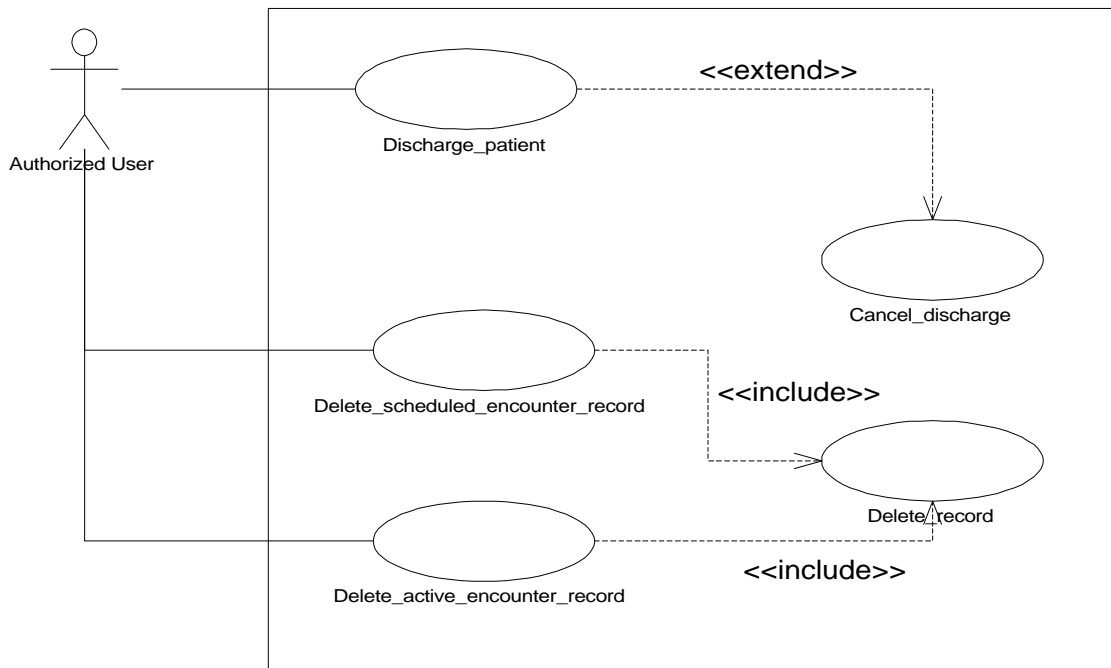


Figure 5-10. This figure uses a subset of the Use Cases from the Example Model (Chapter 12) to illustrate the use of the <<extend>> and <<include>> stereotyped dependency relationships that can exist between two Use Cases. An *extending* Use Case adds behavior to the *extended* Use Case at specifically defined "Variation Points" within the *extended* Use Case. An <<extend>> relationship between two Use Cases is used when the *extended* Use Case can function on its own in most situations, but requires additional behavior to handle non-error exception conditions. The <<include>> relationship is used to factor out common behavior, which is encapsulated in the *included* Use Case. Neither the *included* nor the *including* Use Case normally functions alone. Note that because both the <<extend>> and the <<include>> relationships are named stereotypes of the UML "dependency" relationship, an association utilizing either of the stereotypes indicates that the "source" has a dependency on the "target," i.e., the source is sensitive to a change in the target. Finally, note that this diagram adopts two alternative graphic conventions (neither of which is currently supported in Rose): the System Boundary between the Actor(s) and the System's (subsystem's) Use Cases is shown; and a direction is indicated on the association link between the Actor and the various Use Cases. In the absence of a directional indicator, it is assumed that the Use Case is initiated by the Actor.

5.1.1.2 "Leaf-Level" Use Cases

Since the Use Case model is ultimately developed by decomposing the scope statement into meaningful components, it has a well-defined starting point. The stopping point is sometimes less clear. However, the criteria for stopping are reasonably well-defined in the concept of the "leaf-level Use Case", and careful attention to their occurrence should minimize the times when a Technical Committee "goes too far" in its Use Case analysis.

To say that leaf-level Use Cases play an important role in the message development process is an understatement, for these Use Cases essentially *define* (in an admittedly non-precise manner) the details of each HL7 message. Specifically, the Technical Committee should mine its Storyboards and/or high-level Use Cases for "chunks" of domain-specific information that must be transmitted between healthcare information systems. These chunks can then be further subdivided into individual instances of message transmission and/or reception. Of particular interest with respect to these instances is their association with identifiable life-cycle changes in key domain concepts/entities, entities which will ultimately be represented in the RIM as Subject classes, as well as their association with identifiable Sending and Receiving

Application Roles. *Each leaf-level Use Case should be associated with only one life-cycle transition and one Sending Application Role.*

Each leaf-level Use Case is, by virtue of its association with a state transition for a Subject class, a candidate for initiation by a Trigger event. However, it will not be possible to begin to resolve the final choice of Trigger events until later in the analysis, when the Technical Committee builds the Information and/or Interaction Models.

5.4.3 Actors

Within the context of the MDF, an Actor is a party (i.e., a person, organization, or role thereof) in the healthcare domain that is involved in providing and/or receiving information within the context of the Use Case. More specifically, within the context of a leaf-level Use Case and its associated messages, an Actor is an HL7 Application Role. Thus, an Actor may initially be identified as a human user functioning within a particular "usage context" (i.e., a "role"), or, alternatively, a somewhat generic external computer system. Ultimately, HL7 Actors are Application Roles acting on behalf of real-world Actors. As mentioned previously, Application Roles are an MDF concept useful in the construction of HL7 Conformance Claims (see Chapter 9). Regardless of the level of abstraction, however, an Actor is a usage-context-sensitive role played by such an identifiable domain entity.

Because the "system of interest" in HL7 is the set of messages that must be communicated between the various healthcare Actors, the "HL7 Messaging System" itself has no real "behavior." (As explained in the initial Sections of this Chapter it is the larger HL7 Conformance Messaging System that actually exhibits certain message-based behaviors.) This sometimes causes confusion when defining Use Cases. In particular, when an external system is identified as an Actor, the Technical Committee must give careful consideration to the way it which the Actor is defined and named and, in particular, ask themselves if the system could not simply be abstracted as an Application Role. Unless there is good reason, the external system should simply be labeled as 'Some System' or, better yet, as "Sending Application Role," "Receiving Application Role," etc. The term "Some System" is chosen to reinforce the notion that HL7 Use Cases are not built with *a priori* knowledge of the functionality of specific healthcare information systems. If the Technical Committee has determined that a particular set of messages is only reasonable for systems playing a specific role, it can then provide a more specific name for such an Actor. For example, a set of messages might be defined specifically to manage person identification, and the technical committee might feel strongly that "MPI mediator" is the proper name for the Actor/Application Role. When the Technical Committee is building the Interaction Model, these actors should be reviewed for insight into the roles that need to be supported by the Senders or Receivers of messages (see Chapter 8).

In developing the Use Case Model, the Technical Committee should pay attention to Use Cases in which more than one Actor is involved. In some cases, these Use Cases should be "decomposed" (i.e., collected into a named "subsystem" or "package") correctly delineate the relevant differences in functionality Figure 5-11 and Figure 5-12 describes this type of Use Case "layering."

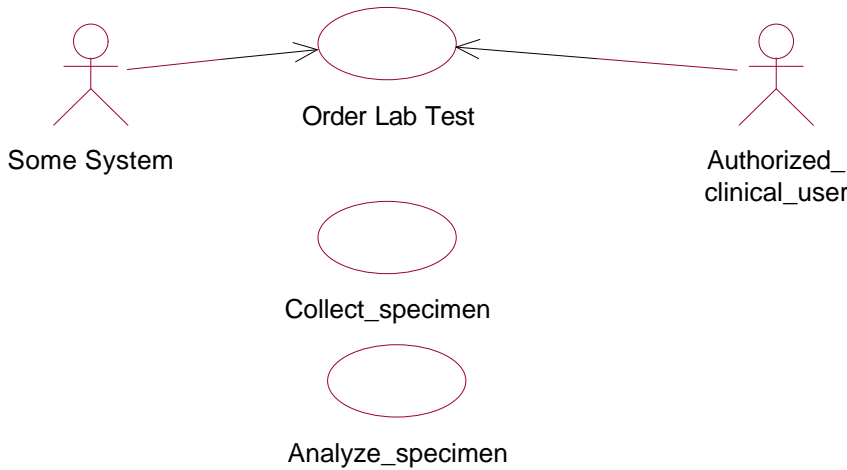


Figure 5-11. The Figure represents a hypothetical Use Case diagram for a subsystem called "Lab Test Execution" which lives within a larger system called "Management of Lab Tests." This system (not shown) has a Use Case named "Perform Lab Test." The decomposition of that Use Case (i.e., the set of use cases that collaborate to fulfill the Use Case) are Use Cases within the "Lab Test Execution" subsystem. The Actors and associations for one of the Use Cases ("Order Lab Test") are shown. The fact that the Use Case is associated with two Actors suggests that the Use Case itself may be further decomposed (Figure 5-12).

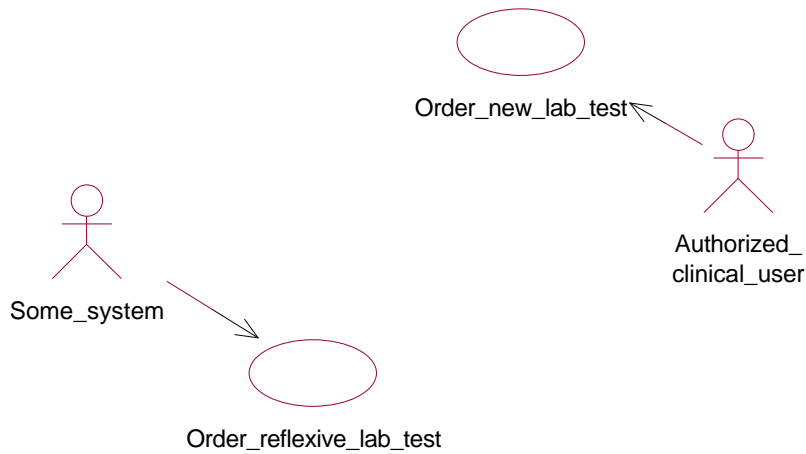


Figure 5-12. The Use Case in Figure 5-11 involving association with two Actors has been decomposed into two distinct System Responsibilities, each being initiated by a single Actor. This decomposition may be represented by either indicating that the two Use Cases in Figure 5-12 reside in a subsystem of the system in which the single Use Case in Figure 5-11 resides, or, by use of a collaboration association between the "parent" Use Case and its "children." In either case, the two Use Cases "Order_new_lab_test" and "Order_reflexive_lab_test" define a more granular architectural layer than the layer defined by their parent "Order_lab_test." NOTE: a collaboration

association is *not* a formal generalization/specialization relationship, and hence, the use of the terms 'parent' and 'child' should not be taken to indicate inheritance in the OO sense of the word.

Unlike many other objects, Actors are non-deterministic in their actions. That is, the explanation for what they do is outside of the scope of inquiry. This limitation is designed to keep focus on the “what” of the business process and to avoid any attempt to begin addressing the “how” prematurely, i.e., designing the messages.

An Actor can be drawn from outside of the application environment (e.g., a person, or organization) or play a part within it as a computer application or system. Clearly, these two are related in the healthcare environment. For example, a nurse may determine the care plan for a patient and then enter it into an application for storage or transmittal to another application. However, the focus for defining Actors shifts during the process of developing the Use Case model from Actors external to the entire healthcare information system to Actors immediately external to the HL7 messaging process (i.e., Application Roles). Initially, it is important to focus on those Actors who initiate the activity that results in a messaging Use Case. These are generally persons or organizations. This makes it easier to focus on the key business rules, and to keep the level of discussion from becoming too detailed. Ultimately, however, it is the precise definition of the Sending and/or Receiving Application Roles that clearly focus the message content of each leaf-level Use Case.

In effect, Use Case analysis involves splitting the healthcare environment into two parts: external users (the Actors) and the internal functions, tasks, and processes that deliver value (Use Cases). The Technical Committee should focus its energy on decomposing the Use Cases to achieve an adequate description of the requirements for messaging and to discover the basic building blocks that form the foundation of the message. These basic building blocks will ultimately become RIM Subject classes, and much of the subsequent MDF analysis centers around them (see Chapter 8).

5.5 Criteria

The following points should be used for quality assurance of the Use Case model. These issues should be considered while the model is being developed, as well as during the review process.

- Responsibility for a particular Use Case model as specified in a Project Scope Definition has been assigned by the HL7 Technical Steering Committee (TSC) to the appropriate Technical Committee within the HL7 Working Group. The assignment of responsibility should be based on the charter of the Technical Committee, and should further be dependent on an assessment by the TSC as whether the specified Technical Committee has the resources to develop the Use Case model.
- The Project Scope Statement to be covered by the Use Case model has been written by the selected Technical Committee and has been approved by TSC, i.e., the Project Scope Statement has been written down for use as a reference document.
- The Project Scope Statement is used as an ongoing "reality check" during the development of the Use Case model, i.e., the System Boundaries and System Responsibilities outlined by the Scope Statement should not change over the course of the development of the Use Case model which delineates and elucidates those Boundaries and Responsibilities.
- In particular, the QA point stated in the previous bullet will functionally mean that a) Use Cases defined to be part of the Use Case Model do not fall outside the scope of the Project Scope Statement; and b) the functional requirements for messaging that fall under the aegis of the Technical Committee are fully identified and specified in the Use Case model.
- For each identified Use Case, the associated documentation is both clear and complete. This is of particular importance when one considers the application of Use Cases as primary communication

tools about the Version3.x Messaging Framework. In particular, documentation of leaf-level Use Cases should include clear statements as to the Trigger Event (state change in Subject Class) as well as the identification of the Sending and Receiving Application Roles and Receiver Responsibilities for each documented message depicted in the Interaction diagram associated with the Use Case diagram.

1. If computer systems and/or stand-alone applications are listed as Actors for a particular high-level Use Case, and if the Technical Committee believes that the system Actor has specific behavior that distinguishes it from a generic "Some System" Actor, the Actor definition should not unreasonably define the scope of a particular healthcare application (although the Actor definition may define a "role" of the specific application.) Note that this restriction should very seldom, if ever, come into play if the Technical Committee follows the guidelines for identifying and naming Application Roles as specified in Chapter 8.

6. Information Model

6.1 Overview

The Information model defines all the information from which the data content of HL7 messages are drawn. It follows object-oriented modeling techniques, where the information is organized into classes that have attributes and that maintain associations with other classes. The information model forms a shared view of the information domain used across all HL7 messages independent of message structure. Thus, the information model provides a means for discovering and reconciling differences in data definition.

6.1.1 Information Model Components

The information model consists of the following components:

- classes, their attributes, and relationships between the classes;
- state-transition models for some classes;
- data types and constraints.

The information model components are defined in the “meta-model” (see Chapter 13). Most of the information model specification is maintained in a relational data base repository according to the meta-model. A textual expression of the information model can be produced by executing a query directly against the database or by exporting to a word-processing tool.

Large portions of the information model have graphical representations in the Unified Modeling Language (UML). This includes the class diagram, the state-transition diagram, and the data type diagram. Those graphical representations are views into the respective information model component, which, in turn, is stored in the repository. The graphical expressions of the information model can be generated from the repository to be viewed with a UML-based modeling tool. The modeling tool can also be used to capture model components for eventual import into the repository. The repository contains all of the same information found in the graphical diagrams plus additional descriptive data for which there is no graphical representation.

Some of the information model is specified informally in descriptive text and accompanying documents. Short pieces of descriptive text can be maintained in both the repository and the modeling tool. Larger pieces of text, that may include figures and text-tables, are maintained outside of the repository. Notably the complete normative data type specification exists in textual form, while the repository only captures the major formal aspects of the data types.

6.1.2 Information Model Notation and Meta-Model

The information model notation and underlying meta-model is based largely on the Unified Modeling Language (UML), a modeling language that unifies the object-oriented modeling methods of Grady Booch, Jim Rumbaugh, Ivar Jacobson, and others. The UML is a rich, mainly graphical, means of expressing object-oriented concepts. It is expected to dominate the object modeling paradigm for some time. To obtain more information about UML see <http://www.rational.com/uml/> or the book *UML Distilled* by Martin Fowler (ISBN 0-201-32563-2).

6.1.3 Types of Information Models

The information modeling process recognizes three different types of information models. Each of the model types uses the same notation and has the same underlying meta-model. The models differ from each other based on their information content, scope, and intended use. The three types of information models are:

- **Domain Information Model (DIM)**

The DIM is used to express the information content for the work of a specific Technical Committee, Special Interest Group, or project. The model is developed or refined by the committee during the information modeling stage of message development. The Reference Information Model (RIM) is the starting point for the DIM. The committee uses the DIM to capture potential additions and changes to the RIM, which it may later submit as RIM change proposals for harmonization.

- **Reference Information Model (RIM)**

The RIM is used to express the information content for the collective work of the HL7 Working Group. The RIM is a coherent, shared information model that is the source for the data content of all HL7 messages. The RIM is maintained by a collaborative, consensus building process involving all Technical Committees and Special Interest Groups. Through a process known as model harmonization, DIM model content submitted as RIM change proposals is debated, enhanced, and reconciled by Technical Committee representatives and applied to the RIM.

- **Message Information Model (MIM)**

The MIM is used to express the information content for one or more related messages. A Technical Committee extracts this model from the RIM during the Message Design stage of the message development process. The MIM starts out as a proper subset of the RIM. The Technical Committee may add message specific information constraints. No new information content is added at this point in the message development process and information constraints added at this point are not allowed to relax constraints specified in the RIM.

6.1.4 Information Model Harmonization

An essential characteristic of information modeling in HL7 is the objective of achieving a credible, comprehensive, and internally consistent representation of the information to be exchanged among computerized information systems in the healthcare domain. The model building process is designed to meet these objectives.

The Reference Information Model is the model of record for specification of the information content of HL7 messages. Contributions to the RIM are made by the HL7 Technical Committees and Special Interest Groups and by ANSI accredited Standards Developing Organizations (SDO). No changes are introduced to the RIM without first providing an opportunity for the entire HL7 working group to review and comment on the proposed change. Following a defined comment period, representatives from each Technical Committee review the proposed changes, along with any comments received from the working group. A consensus process is used to determine which of the proposed changes are actually applied to the RIM. This process, referred to as “harmonization,” ensures that the RIM is a shared view of the entire working group.

Credibility in the model stems from the requirement that all proposed changes to the RIM first be reviewed and approved by an HL7 Technical Committee or Special Interest Group. The healthcare

domain expertise is within these working group committees. The comprehensiveness of the model stems from the diversity and breath of domain interest within the Technical Committees and the willingness of HL7 to consider model contributions and critiques from other American National Standards Institute accredited SDOs. Consistency in the model stems from the adoption and continual refinement of modeling style guidelines and standards delineated in the information modeling procedure section of this message development framework.

The primary objectives of the modeling styles included in this framework are to promote consistency, enhance clarity, and instill stability in the model. An aspect of the harmonization process is to evaluate compliance of proposed changes to the style guidelines and standards. Proposed changes to the RIM that are found to be out of compliance with the styles in this MDF trigger one of three actions:

1. The proposed change is revised in such a way as to bring it into compliance;
2. The style guidelines/standards are revised so as to recognize the style variation as an allowed style;
3. The exception is allowed without modification to the style guide, but noted in the model as an exception.

It is the responsibility of the Modeling and Methodology Technical Committee to oversee this model harmonization process and to maintain this message development framework; however, it is the consensus process involving all of the HL7 working group that determines the outcome of the harmonization effort. A Technical Committee or Special Interest Group may appeal the outcome of harmonization to the Technical Steering Committee for final arbitration. Perceived breeches in the harmonization process itself may be appealed as high as the HL7 Board of Directors. Information model harmonization is an essential part of the HL7 message development process and a responsibility of the entire working group.

6.2 Work Products

The HL7 information modeling notation and meta-model is a derivative of UML. Not all of the concepts of UML are used and some HL7 specific extensions have been added. The primary concepts used in HL7 information models are

- Class, Relationship, and Subject Area;
- Attribute, Data type, and Constraint;
- Subject Class, State, and Transition.

Those concepts will be defined in this section.

6.2.1 Static Structure: Classes and Relationships

6.2.1.1 Classes and Objects

A **class** is an abstraction of things or concepts that are subjects of interest in a given application domain. All things or concepts subsumed under a class have the same properties and are subject to and conform to the same rules. Classes are the people, places, roles, things, and events about which information is kept. Classes have a name, description, and sets of attributes, relationships, and states.

The instances of classes are called **objects**. Where classes represent categories of concepts, people and things, objects represent the individual things themselves. There is still a difference between objects and the things they represent. Objects capture all relevant data about things, which are known to the information system, but are not the things themselves. For instance, a particular human being, named

John Doe, may be represented through an object in an information system. That object contains John Doe's demographic or medical data, but the object is still different from John Doe himself.

Subject classes are classes that are a focus of a committee's interest. State-transition models are defined for subject classes. Messages are defined based on a state-transition model for subject classes.

Associative classes are classes that mainly describe a more complex logical association between other classes. Roles, events, and participation classes are special kinds of associative classes. UML defines the concept of an *association class* which can be represented by an associative class.

Substantial classes are classes that represent some more obvious concept of the application domain, rather than being a mere modeling stereotype. Substantial classes tend to also be subject classes. Substantial classes are usually not merely associative classes. A substantial class should have an identifier attribute.

6.2.1.2 Relationships

Classes relate with classes in various ways. Such relationships can be of three types:

- Generalization
- Association
- Aggregation

6.2.1.2.1 Generalizations

A generalization relationship is a connection between classes (as opposed to instances). The Generalization relationship expresses that one class (called the superclass) is a generalization of another class (called the subclass). Inversely, the subclass is a specialization of the superclass. Instances of a subclass are also instances of the superclass. Conceptually, the superclass does not have a separate related instance to the instance of the subclass, although implementation-wise this may be different. Conceptually, generalization relationships are associations between classes, not between instances.

In a generalization relationship the subclass inherits all properties from the superclass, including attributes, relationships, and states. In addition, the subclass has other properties that are special for the subclass.

Each subclass may in turn have subclasses of its own. Thus, superclass/subclass and generalization/specialization are relative concepts. A class can be both a subclass of its superclass and a superclass of its subclasses. The entirety of superclasses and subclasses with a common root superclass is called a generalization hierarchy.

A generalization usually has multiple specializations. However, not all of the conceptual specializations need to be represented in the model. Only those concepts that warrant special properties (e.g., attributes, relationships, states) are modeled as distinguished classes. If all specializations of a class are fully enumerated as subclasses in the model, the superclass could be "abstract." An abstract class is never directly instantiated, but only through one of its specializations.

Conceptual specializations of a class are not fully enumerated in the model, if not all specializations have special properties beyond the properties of the class that is already in the model. In this case, there is usually some classifier attribute to distinguish specializations that exist conceptually but are not modeled as subclasses.

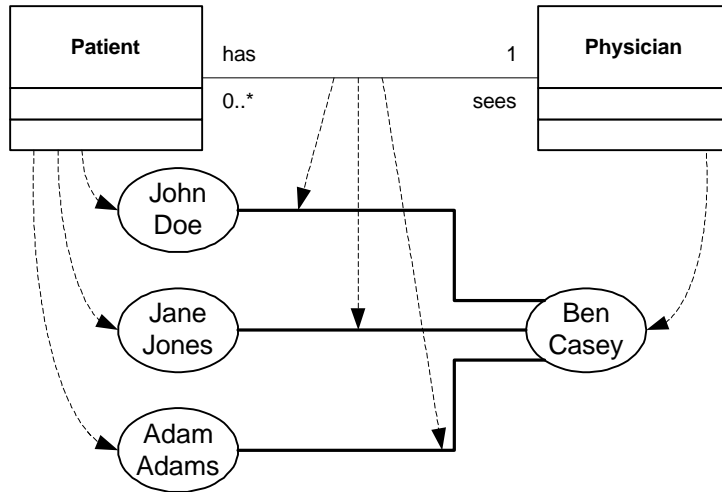


Figure 6-1. One association defined in the information model can have multiple instances. Each instance of an association connects two objects. The number of instances of an association connecting to one object is regulated by the multiplicities.

exactly two objects. The number of instances of an association that can connect to one object is regulated by the multiplicities of the association.

An association multiplicity specifies the minimum and maximum number of objects of each class participating in the association. The multiplicity is documented as a pair of cardinal numbers (i.e., non-negative integers) *min..max*. The lower bound *min* is a non-negative integer, usually zero or one. The upper bound *max* is an integer greater or equal to *min*, usually one, or unlimited, indicated by an asterisk (“*”).¹ Valid multiplicities are listed in the following table:

6.2.1.2.2 Associations

An association defines a connection between objects. The objects may be instances of two different classes or of the same class (reflexive association). Just as classes have instances, associations have instances too. An association instance is a connection between objects and is defined by an association that connects classes.

Associations in the MDF have at least two ends. Each end of the association instance connects with one and only one object. However, one object may be associated with more than one object of the same class by the same association. In this case, multiple association instances exist, each connecting

¹ In other models, the lower case letters “n” or “m” are sometimes used, indicating integer variables. This notation is not defined in the *UML Notation Guide*. The problem with using letters instead of the asterisk is that if the same letter is used in more than once, it may be misunderstood to represent the same integer number everywhere. Furthermore an “n” indicates some boundary specified elsewhere, while the asterisk (“*”) indicates unbounded multiplicity, without the notion of any dependency between such multiplicities.

Multiplicity	Meaning
0..1	minimum zero, maximum one
0.. <i>n</i>	minimum zero, maximum the integer <i>n</i> , where $n > 1$
0..*	minimum zero, maximum unlimited
1	short for "1..1" minimum one, maximum one
1.. <i>n</i>	minimum one, maximum the integer <i>n</i> , where $n > 1$
1..*	minimum one, maximum unlimited
<i>n</i> ₁ .. <i>n</i> ₂	minimum the integer <i>n</i> ₁ , maximum the integer <i>n</i> ₂ , where $n_1 > 1$ and $n_2 > n_1$
<i>n</i> ..*	minimum the integer <i>n</i> , maximum unlimited, where $n \geq 1$

6.2.1.2.3 Composite Aggregations

A composite aggregation is a special flavor of an association between objects. Composite aggregations indicate that objects relate to each other as part and whole. Unlike common associations, a composite aggregation relationship is directed, one end being the whole the other end being the part. The part is conceptually strongly dependent on the whole class, so that the part can not exist without the whole.

6.2.1.2.4 Classes Expressing Complex or Dynamic Associations

Many associations between business objects are of a more complex nature than could be expressed by a common association. For example, an association between two objects might only be valid in a certain time interval. Other associations need to be qualified or classified. This can be expressed with modeling patterns (design patterns). Modeling patterns are building blocks that can be reused in many circumstances. Four of those patterns are defined in the following sub-sections.

6.2.1.2.4.1 Resolving many-to-many Associations

When a model contains associations with multiplicities "0..*" on both ends this is an indication for a complex relationship that is likely to require more information. For example, the relationship between Patient and Physician would be a many-to-many association. In data base design, many-to-many relationships were commonly resolved because there was no other way to technically implement them. In conceptual information modeling, resolving of such relationships is common because it is unlikely that many-to-many associations exist without further qualifications.

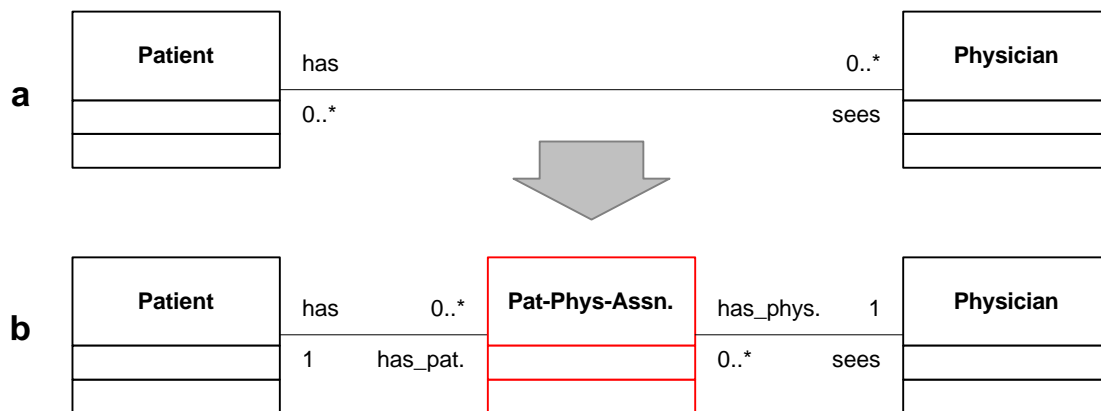


Figure 6-2. Resolving a many-to-many association (a) through an associative intermediary class (b).

Note that the UML concept of an “association class” differs from the “associative class”. An association class does not generally allow resolution of many to many classes, because multiple associations between the same pair of objects are not distinguished as individual associations with an association class. Associative classes are more powerful than association classes. The disadvantage of associative classes is that simple association names are turned into abstract and awkward association names. For instance, where a Physician *sees* a Patient with a many-to-many association, a physician *is_involved_in* many Patient-Physician-Associations, that in turn *has_as_patient* a Patient.

Once an associative class is defined, it is likely that the committee will find specific attributes, such as classifiers, time stamps, etc. to assign to the associative class. The following three design patterns, stakeholder role, event class, and role participation, are special kinds of associative classes.

6.2.1.2.4.2 Stakeholder Roles

Stakeholder is a class representing either a person or an organization that assumes some role in the healthcare industry. A generalization hierarchy with stakeholder as the root superclass and person and organization as the fully enumerated specialization subclasses is used to capture the attributes applicable to stakeholders regardless of the roles they play. The roles assumed by stakeholders are modeled as classes separate from the stakeholder hierarchy. The role classes are connected to the stakeholder, person, or organization class.

6.2.1.2.4.3 Event Classes

An event class represents an activity that occurs at a given location, at a given date and time, for a given duration, involving one or more participants, for a given reason. There are many such “events” in a healthcare information model: encounters, service delivery events, order placement, results reporting, etc.

Event classes often have such properties like effective time interval, reason, location and an association to one or more role or role participation classes.

6.2.1.2.4.4 Role Participation

A role participation class is a class that resolves a many to many association between a role class and an event class. The role participation class reflects the fact that a role can participate in more than one instance of the event and that the event may have more than one instance of the role as a participant.

Attributes of the role participation class include a coded attribute that enumerates the various participation roles the stakeholder role may assume in the associated event.

6.2.1.3 Subject Areas

Subject areas are optional but highly recommended for models of twenty classes or more. The purpose of subject areas is to provide a convenient aggregation of model classes and to partition large models into smaller more manageable subsets.

There are six types of subject areas in the Reference Information Model:

1. *Subject Classes* - This is a single subject area containing the classes which have been declared as subject classes. Subject classes are those classes for which a Technical Committee has elected to construct state transition diagrams. This subject area is named "Subject_classes."
2. *Stewardship* - These subject areas are used to identify the HL7 Technical Committee with stewardship for the classes in the subject area. The subject area name is prefixed with "STW" followed by the committee identifier and name of the steward committee.
3. *Committee Interest* - These subject areas are used to identify a set of classes in which a Technical Committee has a declared interest. The subject area name is prefixed with "COI" followed by the committee identifier and name of the committee expressing interest.
4. *Committee specified* - These are subject areas established by Technical Committees. These subject areas are prefixed with "DIM" followed by the committee identifier followed by a name assigned by the Technical Committee.
5. *RIM* - Subject areas established by the Modeling and Methodology Technical Committee are prefixed with RIM. These subject areas help to partition the Reference Information Model into logical partitions for ease of comprehension.
6. *Data type* - There is a single data type subject area in the RIM. It contains the specification for all data types. The subject area name is prefixed with DTM.

6.2.2 Information Content: Attributes, Values, and Constraints

6.2.2.1 Attributes

Class attributes are the core components of the information model. The attributes are the source for all the information content of HL7. Besides the common attributes there are three special kinds of attributes in the information model:

- identifier attributes
- classifier attributes
- state attributes

6.2.2.1.1 Identifier Attributes

Identifier attributes can be used to identify an instance of a class. Sometimes more than one attribute may be needed to identify an instance of a class. The identifier attributes always have a value. The values of

all identifier attributes together should be unique among all instances of the class. Since identity is static, values of identifier attributes should never change.

However, in the practice of information systems maintaining identity of objects is a problem. For example, consider such attributes as *name*, *date-of-birth*, and *mother's-maiden-name* for a person. Those attributes, are quite meaningful in the real world, but they are neither unique nor static. Names can change, demographic information can get misunderstood or mistyped, and may need to be corrected later. Other identifiers, such as the Social Security Number (SSN) tend to be more stable, but still can be reported wrong, mistyped, or can be misassigned.

If object identifiers shall be truly unique and never changing they must be computer-generated in the information system that first creates an instance of an object. While these identifiers fulfill all criteria to stability and uniqueness, they are completely synthetic and thus can not be guessed or found through partial matches. As soon as such identifiers are being printed and reentered manually, they become unreliable again.

Thus, there are strong and weak identifiers. Strong identifiers being computer generated and never manually entered. A special data type, the Technical Instance Identifier (TII), is designed to be used for strong identifiers.

Weak identifiers are identifiers from less reliable external sources (e.g., SSN, accession number). Attribute sets such as {*name*, *date-of-birth*, *mother's-maiden-name*} are also weak identifiers. Weak identifiers are essentially like common queries (selections). Common queries may return more than one result, which may be different when the same query is executed at different times.

6.2.2.1.2 Classifier Attributes

Classifier attributes can be used in generalization hierarchies that are not fully enumerated. The classifier attribute is placed in the superclass and contains a code value declaring the subclass type. There may be multiple classifier attributes in a superclass, indicating multiple independent generalization hierarchies. In a fully enumerated specialization hierarchy, classifier attributes are not needed only in order to specify the specialization class in a message. Classifier attributes should not be used if the subclasses are fully enumerated in the information model.

6.2.2.1.3 State Attributes

A state attribute is used in subject classes. It contains a value that concisely indicates the current state of the class. A subject class must have only one state attribute. The state attribute must be assigned the data type "set of code value" that allows multiple state flags to be specified.

6.2.2.2 Data Types

Data types are the fundamental constituents of all meaning that can ever be expressed in an attribute. Without data types the meaning of the signs we communicate for an attribute is not defined at all. Data types can be understood as principal value sets for attributes. A particular instance of meaning is expressed by selecting one of all possible values of the data type. But a data type is more than just an enumerated set of values (most data types have infinite value sets). Data types define specific relations and structures among the values in their value set. Data types define everything you can do with one of its values, and everything you can know about such a value.

For instance, consider an attribute "Patient_problem.rank" as of data type integer number, you don't know what the "rank" actually is. It could be a code (*marginal*, *important*, *urgent*, *fulminant*), it could be just text ("not very important," "minor problem," "patient didn't even notice"), or it could be a floating point number between 0 and 1. Only when a data type has been declared for the attribute the expectations to that attribute are clearly standardized. For instance, if rank is defined as an integer number it is clear

that the values are discrete and ordered so one can ask for lower and higher ranks. You can also expect that integer arithmetic operations (+, −, mul, div) are defined, although those would be meaningless for an ordinal ranking. If rank is defined as a floating point number, it has similar properties. In addition, with floating point ranks, one can insert an additional rank between every two ranks.

Data types are often believed to be essentially constraints or restrictions of allowable values. However, this is not quite true. First of all, data types enable meaningful communication by defining semantic fields and “words” by which we can refer to meanings. Nevertheless, because data types are essentially sets of values, once a data type is defined one can define other data types as subsets of the first data type. This is called sub-typing or constraining. For example, once we have defined a data type for floating point values, we can define a data type for probabilities that is essentially a constrained sub-type of floating point number with upper and lower boundaries 0 and 1 respectively.

6.2.2.2.1 Meaning versus Representation

Values of data types have both meaning and form (or representation). One distinguished value can be represented in different forms depending on the Implementable Technology Specification (ITS). The ITS defines all aspects of representation of a data type. On the application layer only the semantic properties of a data type exist. Thus, information models may never be developed with any consideration of representational aspects of data types. For example, we tend to think of data values as having a “length” property. For character strings or list-type collections the concept of “length” does indeed exist. However, most other data types have no notion of length whatsoever. Length, however, is generally a valid concept only on the level of representations of values as characters or bits.

We think of some data types as basic (or fundamental) such as character string, integer number, or Boolean. Other data types appear more complex, such as address, code value, or an instance identifier. We think of complex data types as being comprised of components, each of which is assigned a data type which can be either complex or basic. The nesting of complex data types can be visualized as a tree structure with basic data types at the leaves. However, we must be aware that the semantic construction of data types may be different from the representational construction. That way, a data type appearing semantically basic may well have components in its representation and vice versa.

For example, a floating point value (i.e., an approximation to a real number) may be considered as having two aspects: (1) a value and (2) a precision (e.g., number of significant decimal digits). A character representation of such a number will not need to contain the explicit precision component, since the precision can be encoded in the character representation. A binary floating point representation (e.g., IEEE 754) will need a separate precision component.

6.2.2.2.2 Overview of defined data types

The following table tries to give a short overview of the defined data types that are commonly assigned to attributes. This table is neither complete nor detailed enough to provide anything more than a coarse overview. The complete and detailed definition of HL7 data types is found in the *HL7 Version 3 Data Type Specification* (currently under development, see <http://aurora.rg.iupui.edu/v3dt>). The RIM also contains a special subject area where data types are represented as information model classes.

Name	Symbol	Description	V2.3
Boolean	BL	The Boolean type stands for the values of two-valued logic. A Boolean value can be either true or false.	ID
Character String	ST	Used when the appearance of text does not bear meaning, which is true for formalized text and all kinds of names. Do not use this data type for attributes intended to contain free text. Scarcely any attribute will be declared directly as a Character String.	ST

Encapsulated Data	ED	Can convey any data that is primarily meant to be shown to human beings for interpretation. Whenever attributes should contain text entered and shown to users, the Encapsulated Data type should be used, not the plain character string type. Encapsulated Data can be any kind of text, whether unformatted or formatted written language or other multi-media data. Instead of the data itself, an ED may contain only a reference (URL).	TX, FT, ED, RP
Instance Identifier	II	Used to uniquely identify some individual entity, a piece of data or a real world entity. Examples are medical record number, placer and filler order id, service catalog item number, etc.	ID, IS, CE, HD, EI
Telecommunication Address	TEL	A telephone number or e-mail address specified as a URL. In addition this type contains a time specification when that address is to be used, plus a code describing the kind of situations and requirements that would suggest that address to be used (e.g., work, home, pager, answering machine, etc.)	TN, XTN
Code Value	CV	Exactly one symbol in a code system. The meaning of the symbol is defined exclusively and completely by the code system that the symbol is from. Used primarily for technical concepts, concepts which is crucial to HL7 operations, and concepts which are defined or adopted under the discretion of HL7.	ID, CE
Concept Descriptor	CD	A descriptor for a real world ("natural") concept, such as a finding, a diagnosis, or of any semantic field, that is not under the sole discretion of HL7. A given concept may be expressed in multiple terms where each term is a translation of some other term, or is a (re-)encoding of the original human readable text, that can also be sent in this data type. This data type is suitable for multi-axial code systems.	CE
Integer Number	INT	Integer numbers are precise numbers that are results of counting and enumerating. Integer numbers are discrete, the set of integers is infinite but countable. No arbitrary limit is imposed on the range of integer numbers. Two special integer values are defined for positive and negative infinity.	NM
Real Number	REAL	Fractional numbers as approximations to real numbers. Fractional numbers occur whenever quantities of the real world are measured or estimated, or where quantities are the result of calculations based on other floating point numbers. This type preserves the precision in terms of significant digits.	NM
Physical Quantity	PQ	A dimensioned quantity expressing the result of a measurement. Consists of a floating point value and a physical unit. Physical Quantities should be preferred instead of two attributes expressing a number and a unit separately. Physical quantities are often constrained to a certain dimension. This can be by specifying some unit representing the dimension (e.g., m, kg, s, kcal/d, etc.)	CQ
Monetary Amount	MO	The amount of money in some currency. Consists of a value and a denomination (e.g., U.S.\$, Pound sterling, Euro, Indian Rupee).	MO
Point in Time	TS	A scalar defining a point on axis of natural time.	TS
General Timing Specification	GTS	A data type used to specify the timing of events. Every event spans one time interval (<i>occurrence interval</i>), i.e., a continuous range of natural time between a start-point and an end-point in time. A repeating event is timed through a sequence of such occurrence intervals. Such timings are often specified not directly as a sequence of intervals but as a rule, e.g., "every other day (Mo – Fr) between 8:00 and 17:00 for 10 minutes." GTS is treated as a primitive data type (like TS) where there is a special syntax for specifying time of day, day of week, repetition pattern, etc.	TQ
Ratio	RTO	A ratio quantity is a quantity that comes about through division of any numerator quantity with any denominator quantity (except zero). Ratios occur in laboratory medicine as "titers", i.e., the maximal dissolutions at which an analyte can still be detected. Other Ratios are price expressions, such as dollar per gram.	SN
Postal and Residential Address	AD	The main use of such declared data is to be printed on mailing labels (postal address,) or to allow a person to physically visit a location (residential address). The difference between postal and residential address is whether or not there is just a post box.	AD, XAD
Person Name	PN	Used for one full name of a natural person. Names usually consist of several name parts that can be classified as given, family, nickname etc. This data type is intended to be used only in the Person_name class. Instead of directly using this data type for an attribute of another class, one should consider drawing an association to the Person_name class.	PN, XPN

Organization Name	ON	Used to name an organization. Similar but simpler than the name of a natural person.	XON
<i>Any type</i>	ANY	Used where data of any data type can be sent. This data type must not be used except in very special cases where it needs extensive documentation as to how this can be used interoperable. Currently there is only one use for the ANY data type (i.e., the Observation.value). An attribute of ANY data types can be constrained in subordinate models (r-MIM, HMD).	ANY

Generic (Parameterized) Data Types

Set Collection	SET(<i>t</i>)	A collection of values of any type T without a specifying an order among the elements.	
List Collection	LIST(<i>t</i>)	An ordered set of values of any type T.	
Bag Collection	BAG(<i>t</i>)	An unordered set of values of any type T where each value can occur more than once (rare).	
Interval	IVL(<i>t</i>)	Ranges (intervals) of values of type T. An interval is a set of consecutive values of any ordered data type, such as, integer, floating point, point in time, physical quantity, monetary amount, and ratio.) Intervals should be preferred instead of two attributes expressing a start and an end separately.	SN, XNM
Uncertain value using probabilities	UVP(<i>t</i>)	A nominal value with a probability number indicating the level of certainty for the value to apply in the given context.	
Parametric probability distribution	PPD(<i>t</i>)	A probability distribution used to indicate certainty (accuracy) of a quantitative value. Allows specifying a distribution type and applicable parameters. All distribution types have the parameters mean and standard distribution. The mean is the value that would be reported if no probability distribution were available.	

Note that some data types that existed in HL7 Version 2 no longer exist in Version 3. Many of the old composite types, such as CN, contain multiple concepts, and are now represented more explicitly in the information model as either attributes or classes. Other types, such as ID, IS, and CE, received a more rigorous definition so that an automatic 1:1 mapping is often not possible. Other types, such as PN have been promoted to an information model class.

6.2.2.2.3 Generic Data Types and Collections

Generic data types are incomplete type definitions that a committee can reuse freely in order to construct complex types. Generic data types have one or more parameters that must be bound to another data type to yield a complete data type. The most common use of generic data types is to build collections of values of the same type. There are four kinds of collections:

1. A **set** is a collection of elements where the order of elements is irrelevant and where all elements are unique. The number of distinct elements in a set is called the “cardinality” of the set.
2. A **list** (or sequence) is an ordered collection of elements. The elements need not to be distinct in a sequence, i.e., the same value can occur more than once. The number of elements in a list is called the “length” of the list.
3. A **bag** is an unordered collection of values where each distinct value can occur multiple times. The total number of elements in the set is called its “size” and the number of distinct elements is called its “cardinality.”
4. An **interval** (or range) is a continuous subset of any ordered data type.

A committee uses a generic type by assigning specific types to the parameters of a generic type. For instance, if the committee wants to use a *set of character strings* (ST), it would take the generic type SET(*t*) and bind the parameter *t* to type ST. This is expressed as SET(ST). While SET(*t*) was a generic, incomplete type, SET(ST) is a fully functional data type. For another example, many committees will need time ranges, so they will take the *interval* generic type IVL(*t*) with the data type *point in time* (PT) to construct the data type IVL(PT).

In HL7 Version 2 and prior revisions of the MDF there was a notion of attributes that could “repeat.” This notion of “repeatability” is now obsolete. Whether an attribute has a single value or multiple values depends entirely on the data type. An attribute declared as an ST always has at most one single value. If multiple values are needed, SET<ST> or LIST<ST> is the right data type to choose.

6.2.2.2.4 Missing Information, Defaults, and Null-Values

Information may not be mentioned in a message for a variety of reasons. It may not be available at the sending application. It may not be mentioned because it does not pertain to the transaction. It may not be mentioned because of privacy and confidentiality policies. It may not be mentioned because it would be redundant information. For this reason, all the domains of all data types are automatically extended by a set of so called “null values.” Null values may have many different connotations and different classifications of null values exist. The following table shows the crucial flavors of null values that will be distinguished by HL7. In addition to those other flavors may be defined elsewhere.

Name	Meaning
<i>not present</i>	Value is not present in a given message. This concept exists only in messages, an application program will never deal with <i>not present</i> values. As soon as a message is processed by a receiver it is resolved into a <i>default</i> value, which can be another null value.
no information	This is the default null value. It simply says that there is no information whatsoever in the particular message where the <i>no information</i> value occurs. The information may or may not be available at the sender's system, it may or may not be applicable, known. The receiver can not interpret the “no information” value any further.
not applicable	The attribute does not apply at all.
unknown	A value is applicable but is unknown to the sending system.
other	A value exists and is known but is not an element of the domain. Note that <i>other</i> is a specific flavor of a null value, it is not a “not otherwise specified” null value.

There is no notion of update semantics contained in null values. None of the null values will necessarily take precedence over the other when it comes to update the receiver's data base.

If a value is not present in a message, it will be replaced by a default value specified for a particular attribute or component. The no-information value is the default if no other default is specified. Default values can be specified at any level of the information model (i.e., data types, RIM, MIM), in the HMD, in the ITS and even in a particular message instance if the message definition or the ITS provides for constructs or rules to set a default dynamically. Default values can be overridden. The default value that is closest to implementation has precedence (message element instance > message instance > ITS > HMD > MIM > RIM > data types). Default values comply to constraints, i.e., defaults are allowed values. However, if not explicitly stated otherwise, null-values are always allowed.

6.2.2.3 Constraints

Constraints narrow down the set of possible values that an attribute can take on. Constraints include vocabulary domain constraints (e.g., this attribute must be a LOINC code), range constraints (e.g., this attribute must be a floating point number between 0 and 1) and more. While the term “constraint” has the connotation of restricting and limiting, the objective in defining constraints is more to provide guidance in the proper use of a class or attribute. The important point about constraints is to suggest legal values rather than to penalize illegal values.

Before defining constraints we must introduce the term *domain*. A **domain** is a set of possible values for a variable (i.e., attribute, component, etc.). The set of all values defined by a data type is called the domain of the data type.

Constraints defined on attributes are logical predicates that must be true for the value of an attribute in every instance of a class. With a predicate P one can check if a given value x for an attribute conforms to the constraint ($P(x) = true$) or not ($P(x) = false$). Such a constraint predicate, in theory, generates a subset D' of the domain D of the data type assigned to an attribute: $D' = \{ x \in D / P(x) \}$. Given such a constrained domain D' (a sub-domain), testing for conformance of value x to the constraint is the same as testing whether that value is a member of the sub-domain ($x \in D'$).

The two sides of a constraint, i.e., being a predicate or a sub-domain are equivalent in theory. However, in practice both views of constraints are useful in different circumstances. For instance, when the only way to specify the sub-domain is to enumerate it, there is no use to a predicate. However, if the sub-domain is huge or infinite, the predicate is the better choice for a constraint. The sub-domain can also be derived through the set operations union, intersection, and difference. Operational predicates (e.g., test for equality) and sets (e.g., test for membership) can be used together.

A **vocabulary domain** is a constraint applicable to code values. Vocabulary domains define the set of possible values for a code value. Hence, vocabulary domain constraints are largely defined in terms of sets and subsets rather than using predicates. The data types code value and concept descriptor without a domain specification have an infinite domain that includes all concepts that man can ever conceive of. In other words, code values are useless without some guidance provided by constraints to their value set.

Specification of constraints involves the following items:

1. names for predefined domains, such as data type names and names for vocabulary domains;
2. literal expressions for values of data types;
3. operations and their names or operator symbols defined for data types involved;
4. the name of the variable that is being constrained;
5. references to other variables used to condition the constraint on those other variables;
6. set operations (e.g., union, difference), relations (e.g., element, subset), and quantifiers (e.g., for-all, it-exists);
7. logic operators (e.g., and, or, not) including those to express “branching” (implies, case);

There is currently no standard syntax for constraint specification mandated by the MDF. Any constraint expression language must support the above functionality. Candidates for constraint specification languages include the OMG/UML *Object Constraint Language* (OCL), the traditional notation of set theory and predicate calculus, and natural language expressions. Another notation of constraints is a “pattern.” A pattern is basically an instance notation with variables. Prolog is an example for a programming language based on such patterns.

Constraints may be specified in the RIM, MIM, or HMD. If specified in the RIM, the constraint is relevant for an attribute in all messages containing the attribute. If specified in the MIM, the constraint is specific to all of the messages derived from that MIM. Constraints may also be specified within the HMD where they can be made specific to a particular set of message structures. Constraints specified in a higher level (e.g., the RIM) may be further constrained in a lower level (e.g., MIM or HMD). However, the subordinate constraint must conform to the constraint on the higher level. Higher level constraints can not be undone on a lower level.

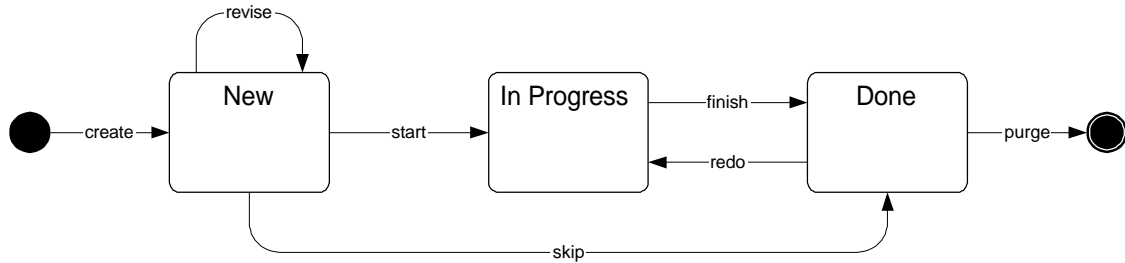


Figure 6-3. A simple state-transition model representing the life-cycle of an activity.

6.2.3 Dynamic Behavior: States and Transitions

Class states and state transitions are defined for subject classes. A subject class is a class the committee designates as the central focus of a collection of messages. A class state is a named condition of a class that can be tested by examination of the class attributes and associations. A state transition is a change in the state of a class by virtue of a change in its attributes or associations such that the condition for the before state are no longer true and the condition of the after state are true.

Although the state of a class is determined by all current values and association instances of an object, states are explicitly accounted in a state attribute of the subject class. Thus the current state of the class can be tested for by examining only the value of the state attribute. Whether other values and association instances of the object or related object conform to the definition of the state becomes a matter of constraint testing. Such constraint declarations for a state are not required, formal methods to express those constraints are being evaluated for use in the MDF. Until a specific method has been selected, constraints can be recorded as natural language clauses.

Each class has only one state-transition model that accounts for all the relevant states of its objects. A state transition model consists of states and transitions. Each state can be thought of as a Boolean variable, indicating whether or not this state is effective. Transitions are directed connections between states. Transitions define all the allowable state changes. A state-transition model defines a “state-machine” (automaton) or “life-cycle” for all objects of a class.

The term “state” has two slightly different meanings. An object is always in exactly one state at a time. However the state of the object can be expressed by multiple partial states being effective in the state machine. In other words, a state-machine may have multiple partial states effective at the same time, but the multiple partial states can be summarized to one joint state of the state-machine. Note that conceptually a joint state is of the same nature as a partial state. The qualifiers “partial” and “joint” can be used to disambiguate the scope of the term “state”.

Figure 6-3 shows the basic features of state-transition modeling in UML. States are rounded boxes and transitions are arrows. Transitions link two states. The semantics of transitions “start” is that if an object is in state “New” it may at some point in time go into state “In Progress”. But it is undefined in the given model at what time the state changes. There would be a trigger event associated with transition “start”. When that event occurs, the associated transition is triggered, it “fires.”

Initial and final pseudo-states are shown as black bullets. Pseudo-states are not proper states because no object can ever exist in such a state. For instance, as soon as the “purge” transition fires, the object ceases to exist, it will never enter the final pseudo state. Likewise, before the “create” transition fires there is no object that could have a state.

Figure 6-3 also shows one fundamental arrangement of states and transitions: a sequence. Although there are branches that allow a transition not only from “New” over “In Progress” to “Done” but also directly

from “New” to “Done,” and although there is a step back from “Done” to “In Progress,” the state-machine will be in only one partial state at a time. Therefore, looking back on the life-cycle history, there will be one enumerable sequence of partial states the object went through, so that we can call this arrangement sequential. In such a sequential state-transition model there is no difference between partial state and joint state.

In Figure 6-4, by contrast, more than one partial states can be effective at the same time. The two states “New” and “In Progress” are now nested within the state “Not Done.” This means that whenever the object is in one of those two sub-states it is also in the enclosing super-state. When a transition fires that ends directly in a sub-state (e.g., “redo”), the sub-state and the super-state are both entered at the same time. Likewise does a transition that links from an inner state to an outer state (“finish” and “skip”) cause both the inner state and its super-state to be left at the same time.

In Figure 6-3 as well as in Figure 6-4 one can still write down the course of an object through its life-cycle as a sequence of states. And the “normal” course of the object would be the same in both. The “abort” transition shows the ability to leave all sub-states in “Not Done” and head to the final pseudo-state preemptively. Whichever sub-states the object is in will be left as the “abort” transition fires.

The life-cycle of objects complying to Figure 6-5 is no longer a sequence of partial states: the state “Held” can be entered and exited independently from the other two states within their common super-state “Not Done.” It is a concurrent state. In UML, concurrent states are depicted by “tiling” of a common enclosing super-state, i.e., by dividing the super-state using a dashed lines into concurrent regions. The transitions of the concurrent regions within the “tiled” super-state can occur independently. Transitions that cross the dashed line may synchronize the otherwise concurrent processes. Without synchronization, there is no notion of “same time,” “before” or “after” between the concurrent regions.

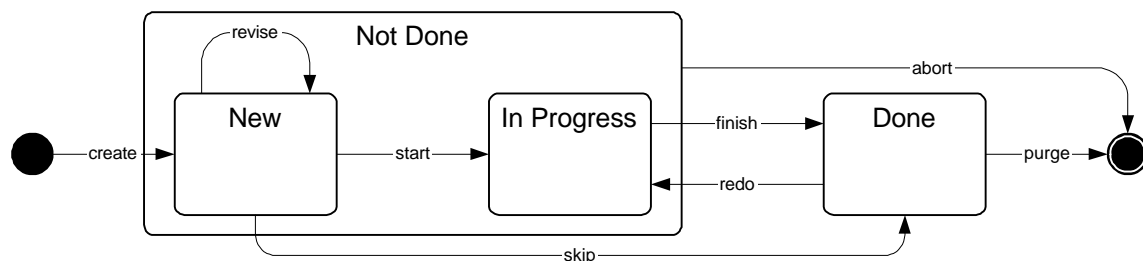


Figure 6-4. State-transition model with nested states and preemption: the activity can be terminated prematurely, if it is not done yet.

Note that the important aspect of concurrent state-machines is not that the objects can be in more than one partial state simultaneously. With nested states, an object can be in multiple states at the same time too. Thus, speaking of “concurrent states” is not the point. The point is that transitions can fire independently from each other. Therefore it is better to speak of “concurrent transitions” or “concurrent state-machines.”

Transitions within different concurrent regions may fire independently, but still, the regions are not completely independent. All concurrent regions within the same super-state are co-dependent on that super-state. When the state-machine in Figure 6-5 leaves the state “Not Done,” either through its normal transition “finish” or through the preemptive transition “abort,” all internal states are left at the same time. Thus, if the object happened to be in state “Held” when the “abort” transition fires, the state “Held” is left as well as all the other nested states of “Not Done.”

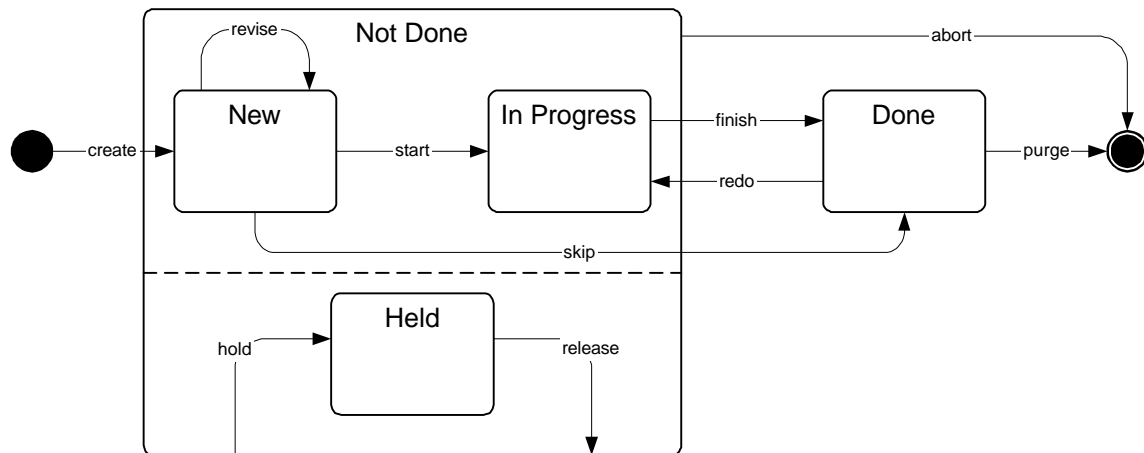


Figure 6-5. State-transition model with parallel states. Regardless of the sub-states above the dashed line, the state “held” can be entered and left independently. When “Note Done” is left, all of the sub-states in “Note Done” will be left too, including “Held.”

Such concurrent inner state machines may have initial and final pseudo-states (shown through black bullets) as well. This means that whenever the enclosing state is entered the initial transition of all concurrent state-machines fires. However, from that point on the concurrent state machines run in parallel without any implicit synchronization.

Note also the special configuration of the state “Held” and its adjacent transitions “hold” and “release” with respect to the state “Not Done.” The transitions link the super-state with one of its sub-states. The semantics of this configuration is implicitly defined by the general rules of our state-transition modeling: (1) The state “Held” can be entered only if the object is in state “Not Done” before transition “hold” fires. But (2) the super-state “Not Done” is not left because the new state “Hold” is one of its sub-states.

States and state transitions of a generalization class are inherited by its specialization classes. Specialization classes may have states and associated state transitions that differ from those of its generalization class. However, the specialized state-machine must be a refinement of the generalized state machine. This means, the general state machine must still be a valid view of the specialized state-machine. For example, the state-machine of Figure 6-5 is a refinement of Figure 6-4. Thus, Figure 6-5 could be from a class “Holdable_activity” that is a specialization of “Activity.” “Refinement” is a relationship between state-transition models just like “specialization” is a relationship between classes. Nested states and concurrent states are used to achieve refinement.

6.3 Procedure

The information modeling procedure is comprised of three activities:

1. Construction/Refinement of a Domain Information Model
2. Update/Harmonization of the Reference Information Model
3. Construction of the Message Information Model

Each of these activities, and their procedure steps, is described below.

6.3.1 Construction/Refinement of a Domain Information Model

Inputs:

- Most recent version of the RIM
- Use Case models for the domain of interest
- HL7 message standard specification and data dictionary
- External information models or data dictionaries

Procedure Steps:

- Define information structural framework (classes, relationships, and subject areas)
- Define information content and constraints (attributes, data types, and constraint)
- Define class behavior (subject classes, states, and state transitions)

Outputs:

- Domain Information Model
- Candidate change proposals for the RIM or MDF
- Proposed data types, attribute types, and constraints

6.3.1.1 Define/Refine Static Structure

This step is driven primarily by an analysis of the relevant Use Cases for the domain of interest. The DIM is initiated by making a copy of the most recent version of the RIM. The selected Use Cases are examined and the DIM reviewed to identify the classes and relationships of interest to the Use Case.

In determining the classes of interest to a Use Case, the committee will seek to identify classes whose attributes would be accessed or altered by the Use Case. Also identified would be the classes whose instances (objects) are created or deleted by the Use Case. And finally, any relationships between objects that would be established, transferred, or removed by the Use Case are also identified.

6.3.1.1.1 Classes

Abstractions for real-world things and major concepts of a committee's application domain are candidates for classes, for instance, people, places, roles, things, and events about which information is kept.

Classes do not necessarily represent "tables" or "relations" in existing information systems and such tables in existing information systems need not necessarily be represented as classes in the RIM. Classes should be defined for conceptual reasons, not primarily because of a certain implementation. However, in practice, there is a dependency between the conceptual classes and implementations. Since not all RIM classes correspond to some distinct entity within existing information systems, not all information systems can identify and distinguish instances of such classes.

For example, a person's address may conceptually be modeled as a class "Location" that is associated with the class "Person." If two persons live at the same address, one would like only one Location object to be associated with both Person objects. However, this requires information systems to actually identify Locations as distinguished concepts, which many existing systems can not do. Those considerations should not discourage a committee to do what it believes captures the concepts best, since the decision whether or not to define a class should not depend on what current systems can or can not do. However,

the assumptions that are made about a class can not ignore current realities. Thus, in the example, one can not expect that Location objects be tracked individually by information systems, and one can not expect that information systems could know about all persons associated with the same physical location.

All new classes must either be associated with a class already in the DIM or participate in a chain of classes and relationships that eventually lead to a class already in the DIM. All classes must have a description which clearly identifies what the class is, and which provides instance qualification criteria and examples as needed.

The singular forms of nouns are to be used for class names. The use of conjunctions, such as “or” and “and” in class names is to be avoided. A class name may be composed of multiple words separated by “_” or “-”, but only to increase clarity or ensure uniqueness. Superfluous words should be omitted. Classes that capture a relationship between two classes should have a name that indicates the nature of the relationship in a meaningful way. Alternatively, associative classes can take on the name of the two associated classes followed by the suffix -Assn.

6.3.1.1.2 Generalizations

Generalization relationships are used because of two different reasons. It may be that a committee discovers a specialization of some class that has additional properties not applicable to the class that is already in the model. On the other hand, a committee may discover that some existing classes in the model are conceptually related and share common properties, so it may define a new class as the generalization of the existing classes.

Through the generalization relationship, subclasses inherit properties from their superclass. Those properties include attributes, relationships, and states. However, the primary rationale for using the generalization relationship must not merely be inheritance of properties if there is no conceptual relationship between the classes. A subclass must be a conceptual subcategory of a superclass. The generalization relationship should not be used simply to provide for inheritance among classes without conceptual links.

A simple test that can discover improper use of the generalization relationship: Construct an English sentence of the form: “A *<subclass>* is a kind of *<superclass>*.” If this “does not sound right” one should investigate the problem. A construct that fails this “read it loud” test must be supported by a written rationale.

The generalizations and specializations should be built because of classes having distinguished properties in the model. Thus, a specialization should be defined because of additional properties beyond those inherited from the superclass. Specializations should not be defined in order to visualize a complete taxonomy. Likewise, generalizations should not be defined because two classes have a conceivable common generalization if the generalization is not warranted by specific information needs. For example, one would not create a superclass “Thing” to every class representing a concrete body in the real world if there are no common attributes or relationships that all those “Things” would share in common.

If the conceptual specializations of a superclass are not fully enumerated, because their properties are not distinguishable in the model, a classifier attribute must be placed in the superclass to explicitly declare the conceptual specialization. If, however, the specializations are fully enumerated, a classifier attribute should not be defined.

Care must be taken when using generalization hierarchies to ensure that the attributes and associations inherited by the specializations are appropriate for all specializations subordinate to the superclass generalization. If the inheritance is appropriate for some but not all of the specializations, consideration should be given to modifying the hierarchy by adding intermediate superclasses to correct the inheritance

or by moving the attributes and connections down the inheritance hierarchy to the appropriate specializations.

State attributes and state-transition model are also subject to inheritance. This requires the special attention of the technical committee. Generally, the state-transition model of a sub-class must be a refinement of the state-transition model of its super-class, as will be discussed below.

6.3.1.1.3 Associations

The committee defines associations if it discovers that relationships between class instances exist. Associations are defined for a certain purpose, which is expressed in the association names on each end. Every association must be named on each end and a multiplicity must be specified for each.

Each association name is a short verb phrase in indicative mood and present tense, written from the perspective of the class on that end of the association. The association names capture the nature of the association between the source and destination classes. The association names are in lowercase letters with underscores “_” used to separate the words.² Indicative mood means that verbs must have the form “is,” “has,” “does,” instead of “may be,” “can have,” or “might do.” The fact that a relationship is potential (“may” instead of “is”) is represented in the multiplicities, not in the verb form. Present tense means that verbs must have the form “is,” “has,” “does,” instead of “has been,” “had,” or “will do.” The model should be independent from any specific temporal perspective. If temporal aspects need to be distinguished an associative class with a time range attribute (attribute type “_tmr”) should be used.

The multiplicity is documented as a pair of cardinal numbers (non-negative integers) *min..max*. The lower boundary *min* is a non-negative integer, usually zero or one. The upper boundary *max* is an integer greater or equal to *min*, usually one, or an asterisk (“*”) indicating an unlimited upper boundary.

In textual representation an association is defined as:

```
⟨class1⟩ :: ⟨association_name⟩ ( ⟨multiplicity1⟩ ) :: ⟨class2⟩ :: ⟨association_inverse_name⟩ ( ⟨multiplicity2⟩ )
```

For example:

```
Patient :: is_involved_in (1..*) :: Patient_encounter :: involves (1)
```

```
⟨class1⟩ = Patient
```

```
⟨association_name⟩ = is_involved_in
```

```
⟨multiplicity1⟩ = 1..*
```

```
⟨class2⟩ = Patient_encounter
```

```
⟨association_inverse_name⟩ = involves
```

```
⟨multiplicity2⟩ = 1
```

² UML has both association names and role names. UML association names are rendered somewhere half way between both ends. The names at each ends are called “role name” in UML. The MDF modeling style uses the role names instead of a UML association name to name associations.

Although there are infinitely many possible multiplicities, only a few are normally used. Special names are attributed to certain pairings of multiplicities, as follows:

Mult ₁	Mult ₂	Name	Usage Note
0..1	0..1	Fully optional one to one	1. In reflexive associations when the goal is to model <i>linear chains</i> (without branches). Truly non-branching chains are not very common. 2. In associations between two different classes this type is extremely rare , it would be appropriate only for an <i>exclusive relationship of a reciprocal pair of instances that do not always find each other</i> . Pot and cover, key and lock, husband and wife are examples that <i>might</i> be modeled this way <i>if the relationship is truly exclusive</i> . However, the same cover can be used for many pots, one lock has usually many keys, and marriage does not always last forever.
0..1	0..*	Fully optional one to many	1. In reflexive associations to model <i>tree structures</i> with single roots. 2. Bundles of such associations may be used for <i>ad hoc</i> choices that are not better modeled through specializations, where the multiplicities would be 1—0..*. 3. Appropriate for weak aggregations (e.g., Account and Transactions).
0..1	1	Optional one to one	1. Mandatory for all stakeholder role associations. 2. Similar to roles, for sets of additional attributes that together are optional for an instance (e.g., Service vs. Service with Authorization). Consider specialization if eventual existence of the association is predetermined. 3. For events with additional information that can happen only once in a life time (e.g., birth, death).
0..1	1..*	Optional one to mandatory many	1. Extremely rare. 2. If such multiplicities occur in a bundle of associations, the optional end 0..1 may express an ad hoc choice. Consider introducing specializations to turn 0..1 into 1. Even then it is a rare association. See 1—1..* below.
0..*	0..*	Fully optional many to many	1. In reflexive associations if the goal is to model <i>network structures</i> or polyhierarchies. 2. For both reflexive and inter-class relationships, consider resolution through an associative class . In the rare case where the associative class remains without other properties this multiplicity type may still be appropriate.
0..*	1	Optional one to many	This is the predominant type of multiplicities, about 50% of all associations in the RIM are of this type.
0..*	1..*	Optional many to many	Extremely rare. Consider resolution through an associative class. Even then it is a rare configuration. See 1—1..* below.
1	1	Mandatory one to one	Deprecated , combine the so associated classes into one class.
1	1..*	Mandatory one to many	1. Rare. 1..* multiplicities are always suspect for a hidden business rule constraint. The model should represent logical rules not business rules. Consider relaxing to 1..* to 0..*. 2. A logical rule could exist for a role class and the class representing the defining function or activity of that role. For example, engaging in a Guarantor_contract is the defining function of a Guarantor (role). Make sure this is a logically defining function not just a business rule.
1..*	1..*	Mandatory many to many	Extremely rare. Consider resolution through an associative class. Even then it is a rare configuration. See 1—1..* above.
n ₁ ..n ₂	n ₃ ..n ₄	Specific multiplicities other than 0, 1, and *.	1. Extremely Rare , consider relaxing specificity. The RIM should contain logical constraints not business rules. 2. Sometimes a multiplicity of 2 or 3 might occur for very specific reasons. E.g., multiplicity 2 could be used for things that naturally occur only in pairs and if such symmetry is important for the model. 3. Multiplicities above 3 occur virtually never. Even as a business rule the constraint is probably better expressed using qualitative than quantitative rules.

If a multiplicity is used in spite of being marked as rare, a rationale must be provided in the description of the association. A written rationale is especially required if the multiplicities are used for any other than the reasons specified. Deprecated associations should be revised. If a committee believes the deprecated association is correct, it should approach the Modeling and Methodology committee with this issue.

Associations that begin and end with the same class are called **reflexive** (or recursive) and must have a minimum multiplicity of zero on both ends (to allow recursion to terminate). Recursion may be established transitively through generalization relationships. In such a case there may be specializations not involved in the recursion. If such alternative specializations exist, the multiplicities on such a transitively recursive association may have a low bound greater than zero at the end attached to the superclass.

Multiplicities in the RIM represent **logical constraints**. Logical constraints are applicable to all meaningful utterances (i.e., instances of the model). Business rules usually imply further constraints. The RIM should not be restricted to **business rules** that may not apply everywhere in the world or at all times. Business rules are subject to change, but the RIM should not be invalidated through change of legal or regulatory context.

For instance, a business rule would be that a patient needs to have an encounter and a billing account covered by an insurance, a guarantor, or a sufficient credit, before a health care service can be delivered. Such a rule, however, (besides being ethically problematic) would not necessarily hold for all institutions (e.g., a charitable organization). Such business rules are often hidden in multiplicities with a low bound of greater than zero. If such multiplicities occur, the committee should make sure there is a logical rationale for it, not just a business rule.

Because associations have a specific purpose, it may well be that there are two **parallel associations** linking the same two classes. Those associations should not automatically be merged into one just because they are parallel.

Not all classes that may be related must have associations connecting them directly. A conceptual association can transitively span multiple classes. However, if objects for the intermediate classes do not always exist when there is a relationship between the end classes, a direct association may be the right choice.

UML has no construct dedicated to *ad hoc* choices. Nonetheless such choice constructs do occur. One way to model choices is through specializations. For example, if a service target can be either a patient or a specimen, the service target could be modeled as having two specializations: `Patient_service_target` and `Specimen_service_target`. However, modeling this way may violate the rule that generalization hierarchies are to represent conceptual relationships not just utilization of inheritance. Missing attributes in the subclasses may indicate the lack of conceptual rationale for the construct.

Such choices that do not conceptually warrant subclasses, may be modeled through bundles of associations that all have multiplicities 0..1 at the distal end. The meaning of such a bundle of optional associations is a “one-of” choice. “One-of” means that the associations are not truly optional but that one and only one such association must be instantiated. This is a constraint that must be made clear at least in the description of the proximal class. It must also be expressed by giving identical names to each association in the bundle. The graphical model can further arrange the bundle so that it is visually clear and further enhanced by a comment text at the proximal end of the bundle stating the words “ONE-OF.”

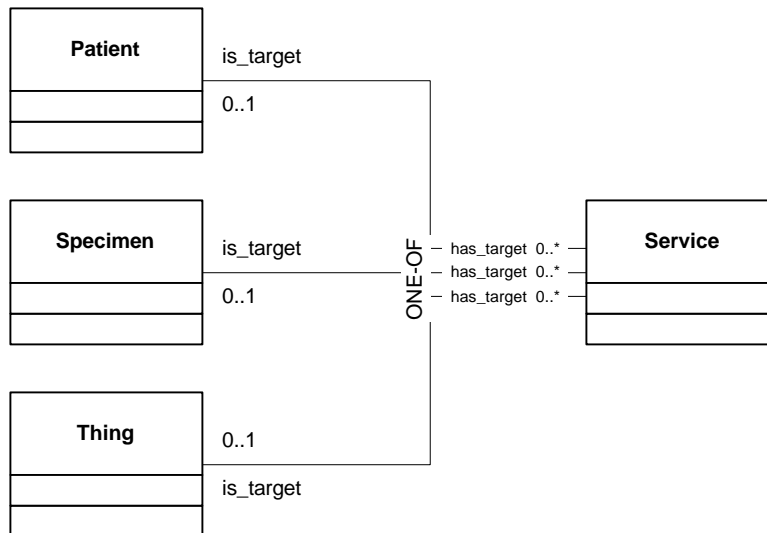


Figure 6-6. Visualizing the ONE-OF constraint on a bundle of outgoing associations representing a choice.

6.3.1.1.4 Composite Aggregation

The composite aggregation expresses some strong conceptual dependency of a part to the whole, so that the part can not exist without the whole. The “part” class is part of one and only one “whole” class.

Composition aggregations are defined as:

```

<whole_class> ::
has_as_part (n1..n2) ::
<part_class> ::
is_part_of (1)

```

Oftentimes there is no clear-cut distinction whether objects relate as whole/part or as

normal instance connections. Care must be taken when using the composite aggregation to ensure that the nature of the relationship is truly a strong conceptual dependency. Composite aggregations are very rare in conceptual information models.

In no way may the composite aggregation be used to reflect implementation considerations. For instance, if a one-to-many association is implemented as an array, it appears as if the class on the many side is part of the class at the one side. However, this is an implementation dependency not a conceptual dependency, and does not warrant the use of a composite aggregation. When in doubt use a common association instead.

6.3.1.1.5 Classes Representing Complex Conceptual Associations

Many-to-many associations should be resolved using associative classes.

```

<class1> :: <association_name> (0..*) :: <class2> :: <association_inverse_name> (0..*)

```

could become

```

<class1> :: <association_name> (0..*) :: <class1>-<class2>-Assn :: <some_name> (1)

```

```

<class2> :: <association_inverse_name> (0..*) :: <class1>-<class2>-Assn :: <some_inverse_name> (1)

```

The committee might consider giving a less formal and more descriptive name to the associative class. For example, a many to many association between a “Thing” and a “Location” might be called “Presence.”

Associative classes often have attributes, such as classifiers, time stamps, etc. However, the committee should be conservative with assigning identifier attributes to the associative class. Associative classes may reveal a very important conceptual modeling construct but they may also be mere products of formal rules. If given identifier attributes, committees promote such associative classes to “substantial” classes, and in doing so, suggest, even mandate, that HL7 compliant systems must distinguish instances of such classes.

If the committee, when considering an associative class to resolve a many-to-many association, does not find any attributes to assign to the associative class, it may elect to leave the many-to-many association instead. For instance, an association

`Service_location :: belongs_to (0..*) :: Service_location_group :: contains (1..*) :: Service_location`

might be resolved using an associative class “Service_location_group_membership.” The long name already suggests this is a fairly abstract minor concept. If the committee does not find any use of identifying, classifying, or time-stamping individual group memberships, the committee should go ahead with the many-to-many association. Classes should be defined for conceptual reasons, not for purely formal reasons. However, since conceptual entities might be discovered through consequently executing formal rules, the committee should go through the considerations outlined here and document its findings and decisions.

Role, event, and participation are special associative classes that should be used as described below.

Classes that represent roles assumed by stakeholders, persons, or organizations must be associated with the class Stakeholder, Person, or Organization with an association named:

`<role_class> :: is_a_role_of (1) :: (Stakeholder | Person | Organization) :: takes_on_role_of (0..1)`

Classes that represent the participation of a role class in an event must be associated with the role class using an association named

`<participation_class> :: is_a_participation_of (1) :: <role_class> :: participates_as (0..*)`

and the Participation class must be connected to an event class with the following association:

`<participation_class> :: is_a_participant_in (1) :: <event_class> :: has_as_a_participant (n1..n2)`

Associative classes represent associations with attributes. Committees should note the difference between a class that exists because it reflects some more or less obvious concept of their application domain, or whether a class is merely a modeling stereotype such as an associative class. Committees should try to avoid connecting two associative classes directly to each other. Such constructs can often be simplified. The role participation is an exception. Role participation is an associative class between two associative classes (role and event). However, role and event are often implemented as “substantial” classes in information systems.

6.3.1.1.6 Subject Areas

Subject areas are optional but highly recommended for models of twenty classes or more. The purpose of subject areas is to provide a convenient aggregation of model classes and to partition large models into smaller more manageable subsets.

A Technical Committee can establish subject areas. These subject areas are prefixed with “DIM” followed by the committee identifier followed by a name assigned by the Technical Committee.

All subject areas must have a description consistent with its purpose. New classes must be assigned to stewardship and RIM subject areas. A new class may also be assigned to COI, DIM, and the subject class subject areas.

6.3.1.2 Define Information Content and Constraints

This step is driven primarily by an analysis of relevant Use Cases. In this step the attributes for classes affected or referenced by the Use Cases are identified and defined. For each attribute, the associated data

type and constraints are also specified. Although captured in the information model, specification of a data type and vocabulary domain for an attribute can be delayed until creation of the HMD. External information models or data dictionaries, and the HL7 v2 message standard specification and data dictionary are potential sources for information needed to identify and define attributes, data types, and domains.

6.3.1.2.1 Attributes

When analyzing Use Cases, the committee should first consider what information would need to be communicated between information systems as a result of the activity described by the Use Case. Next, the committee will identify what class or set of classes in the domain information model is the most appropriate source for that information. The attributes of the candidate classes are examined. If the attributes needed to convey the required information are not present in those classes, they are added.

Attributes have a name, description, and an associated data type and constraint specification.

An attribute name is comprised of one or more words. The final word in the attribute name is called the “attribute type.” To add clarity or to create uniqueness. One or more qualifier words should precede the attribute type. The use of attribute type words in attribute names aids in creating uniformity in the names, helps to avoid unintentional redundancy, and adds clarity to the model.

The class name should not be repeated in the attribute name itself. The scope of the attribute name is the class, so attribute names need to be unique only within the class where they are defined. The order of attributes in a class has no significance in the information model. However, some attributes are usually more essential to the meaning of the class than others and a certain order of the attributes from high to low importance can improve comprehension of the model.

Attribute types used in attribute names must be selected from those defined or approved by the Modeling and Methodology Technical Committee. When selecting an attribute type for an attribute the committee must select, from the list of attribute types, the one with an intended usage consistent with the characteristics of the information represented by the attribute. Attribute types are not data types. A one-to-one correspondence between attribute types and data types is not required. Attribute types are merely suffixes of the attribute names that indicate an intention of what the attribute is about.

The following table lists the attribute types and provides usage notes and suggested data types for each one. In the suggested data type column, a boldface suggestion is the default, i.e., a data type usually chosen for the attribute types. If the default suggestion is underlined, the selection must be voted on in the RIM Harmonization.

Short Name	Full Name	Usage Notes	Suggested Data Types
ADDR	Address	For attributes representing the location at which an organization, person, or item may be found or reached.	AD
CD	Code	For attributes representing some concept. Note that names of individual things are not considered concepts.	<u>CV</u> , CD
COM	Communication address	For attributes representing communication addresses, such as telephones, fax, pagers, e-mail, Web-sites and other devices and their respective protocols. See also PHON.	TEL
DESC	Description	For attributes representing a statement used to describe something.	ED
DTTM	Date and Time	For attributes representing a point in time at which an event happened or will happen. Levels of precision and variation are part of this concept and should usually not be specified in separate attributes.	TS

EXPR	Formal expression	For attributes representing formalized text that is to be evaluated primarily by computers. An attribute named "constraint_txt" is most likely such a formal expression and should be renamed to "constraint_expr".	ST
FRC	Fraction	For attributes that represent a fraction or proportion. The former attribute type PCT for "percentage" is superceded by FRC and is no longer permitted. See also QTY.	REAL
ID	Identifier	For attributes that serve to identify some instance of an information model class. Note that real world Identifiers (e.g., SSN) exist not as attributes but as an association to a special information model class. The attribute type "id" without a prefix is reserved to be the main instance identifier of the class.	II
IND	Indicator	For attributes representing a specific condition as true or false.	BL
NBR	Number	For attributes representing dimensionless numbers. Note that there is a big conceptual difference between integer numbers and floating point numbers. See also QTY.	INT, REAL
NM	Name	For attributes that represent a name by which an instance of the class is known.	ST, PN, ON
PHON	Phone	For attributes representing telephone number of a telecommunication device. See also COM.	TEL
QTY	Quantity	For attributes representing a quantity. The nature of the quantity must be further specified through the choice of data type and through additional constraints. For physical quantities (including elapsed time) the PQ data type must be used. For monetary amounts the MO data type must be used. Parallel unit attributes are not permitted in these cases. Counted objects are not physical quantities and the count nouns are not units of measure. See section 6.3.1.2.1.4.	PQ, MO, REAL, INT, RTO
TMR	Time Range	A range of time between a start and an end time having a duration. The range may be infinite or undefined on either side.	IVN<PT>
TIME	General Timing	A range of time between a start and an end time having a duration. The range may be infinite or undefined on either side.	GTS
TXT	Text	For attributes representing non-descriptive, non-naming text not targeted to human interpretation. Formal expressions evaluated by computers should use the EXPR attribute type instead.	ED
VALUE	Value	For an attribute (e.g., Observation.value) that represents a value whose data type is determined dynamically and is not predefined by the static class diagram.	ANY

If the information represented by an attribute does not fit into any of the defined attributes types the committee should bring the item to the attention of the Modeling and Methodology Technical Committee as a proposed addition or revision.

Care must be taken when selecting classes for attributes. The attribute must represent a fact about the class to which it is added. Before making a final determination, the committee should review the description and direct associations of the class. For example, when considering the Use Case "REGISTER PATIENT," the committee might consider the name of the patient to be a relevant piece of information that needs to be represented in the model. The committee might consider adding the attribute "NAME" to the class "PATIENT." However, after reviewing the description and associations of the Patient class the committee would discover that the Patient class is modeled as a role of a Person and that Name is actually an attribute of Person not Patient. Persons have names regardless of the roles they take on. Careful consideration of attribute placement will minimize the amount of unnecessary redundancy in the model.

An attribute may be added to the information model without specifying a data type or attribute value constraint. However, a data type must be specified in the RIM prior to an attribute's inclusion in an HMD. The same data type must be used for an attribute in all messages containing the attribute. An attribute value constraint may also be specified in the RIM.

6.3.1.2.1.1 Identifier Attributes

Attributes whose sole purpose it is to identify an instance of a class must have the attribute type suffix “_id.” Attributes that do not identify an instance of a class must not use the attribute type suffix “_id.” For instance, “observation identifier” to capture, e.g., a LOINC code may not use the “_id” attribute since it is not an identifier for an instance but rather a classifier for a kind of observation.

The attribute name “id” may be used only for truly unique and stable instance identifiers. Such identifiers must be assigned the data type Technical Instance Identifier (TII) and the use and maintenance of such attribute is governed by HL7 rules. This include the rule that strong identifiers must not be routinely communicated via humans. For example, a medical record number that is typed in at some terminal can not be a strong identifier. Weak identifiers (e.g., accession number) may use the “_id” attribute type suffix, but must have at least one qualifying word.

Whether or not to provide for a strong identifier in a class is an artful decision made by the technical committee. A strong identifier means that information system must be able to match the identifier with a particular instance of a class. Strong identifiers should be assigned to those classes that are commonly recognized entities in most existing information systems. Classes that correspond to commonly identified entities in the real world (substantial classes) are candidates for strong identifiers. Conversely, such classes that are merely modeling artifacts, should not be assigned a separate strong identifier.

Substantial classes are classes that represent some more obvious concept of the application domain, rather than being a mere modeling stereotype. Substantial classes tend to also be subject classes, i.e., to have a state-transition model. Substantial classes are usually not merely associative classes. Roles, and events, though being associative classes, can be regarded as substantial classes. Service_event and Encounter are so important in health care that it is not even clear they are associative classes.

Role classes are often taken for substantial classes in existing information systems. Many existing information systems do not distinguish a Person from a Patient. A committee should be aware that assigning strong identifier attributes to both the Patient class and the Person class may lead to problems with systems that do not distinguish them. In that case, the committee has two options. It can either be aware that the Patient class is still essentially an associative class, regarded as a non-substantial, and thus not being identified independently from patient's also being persons. Conversely the committee could argue the Person not to be a substantial class and so deciding to give Patient, Provider and other Stakeholder roles a strong identifier attribute. If two classes are as tightly connected as role classes (1 to 0..1) only one of both classes should be given a strong identifier attribute.

The following rules should be obeyed if no particular reason suggests otherwise: **1.)** In a generalization hierarchy, a strong identifier attribute is assigned only to the highest level super class (the root of the hierarchy). **2.)** Classes that have a direct or indirect mandatory to-one association to a substantial class, should not themselves have their own strong identifier. In particular, role classes should not have a strong identifier. Also, participation classes should not have strong identifiers.

In a generalization hierarchy, at most one strong identifier should be defined for the root superclass. Role classes, participation classes, must not be assigned a strong instance identifier.

6.3.1.2.1.2 Classifier Attributes

Classifier attributes can be used in generalization hierarchies that are not fully enumerated. The classifier attribute is placed in the superclass and contains a value declaring the subclass type. There may be multiple classifier attributes in a superclass, indicating multiple independent generalization hierarchies. Classifier attributes should not be used if the subclasses are fully enumerated in the information model.

Classifier attributes must have the attribute type “_cd.” Classifier attributes should be named “type_cd” for consistency. The code domain of such classifier attributes must be completely defined and the data type Code Value (CV) must be used.

Classifier attributes must have a specified vocabulary domain. If there is no preexisting vocabulary domain to refer to the committee must at least enumerate 3 to 5 values to illustrate the concept. For many classifier attributes there is no pre-existing reusable vocabulary domain. The committee defining the attribute has the responsibility to closely define the vocabulary domain. The committee is encouraged to approach the vocabulary Technical Committee for advice, but ultimately only the committee that defines and maintains an attribute can know enough about its intended use to be able to specify the allowed values for that attribute.

6.3.1.2.1.3 State Attributes

State attributes are used in subject classes. It contains a value declaring the current state of the class. A subject class must have only one state attribute. The state attribute is named “status_cd” and assigned the data type SET(CV), that allows multiple flags for partial states to be specified. By making a class a subject class, there is nothing else to decide for the technical committee: every subject class must have one and only one state attribute called “status_cd” of data type SET(CV).

6.3.1.2.1.4 Quantitative Attributes

Quantitative attributes require some further thoughts to make sure they are modeled correctly. Some definitions are useful to discuss about the nature of a quantitative attribute.

Counting, estimating, and measuring. *Counting* is a primitive act of using integer numbers to state the quantity of discrete objects. *Estimating* (and averaging) is the expression of counts when counting is not actually and practically possible. Estimates can be fractional numbers, which does not mean that some objects actually occur fractionated (e.g., if the average class size is 185.5 students, there is not actually a half student in each class.)

Measuring is the process of quantitatively assessing phenomena. A measurement process involves

- a) a standard phenomenon (the unit, e.g., 1 pound)
- b) a procedure definition (operationalization) how to compare the actual phenomenon with the standard phenomenon, (e.g., using a balance instrument).
- c) a set of postulates, assumed laws of nature, which inform the operationalization (e.g., relationship between mass and gravitational forces, the laws of levers, moment of force, etc.).

Note: historically measuring is older than counting. Early pre-ancient cultures could measure the quantity of soldiers but could not actually count them. They measured by literally "filling" soldiers into circles of some defined size.

Kinds of quantities. Quantities can be grouped into *kinds of quantities*. All quantities of the same kind are comparable. In measuring, different units can be used to measure the same kind of quantity, thus, some units can be compared with each other and form a *comparability systems* (6). In **physical quantities**, there are different units representing the same kind of quantity with different but constant magnitudes. Hence the term *kind of quantity* is used mostly with respect to physical quantities. Physical dimensionality and kinds of quantities are not the same structures (e.g., a Newton-meter (1 Nm) is one unit and one physical dimension ($L^1M^1T^2$) that measures two different kinds of quantities, Work and moment of force).

The "money" kind of quantity. In monetary quantities there are also different units representing the same kind of quantity with different magnitudes. The magnitudes of currency units are, however, not constant but highly volatile. Therefore, even though all monetary quantities measure the same kind of quantity (money, material wealth,) comparability is only given at a certain point in time (e.g., it is a big difference if your account with 5% interest is denominated in U.S. Dollar, Euro, or Indian Rupee).

"Count" kinds of quantities. Any quantity has a kind of quantity. For counted quantities (including estimated counts) the "number of *counted objects*" is the kind of quantity (e.g., number of students, number of cars). Hence, since all conceivable objects can be counted, these objects extend the set of kind of quantities *ad infinitum*.

Units vs. count nouns. The concept of units exists only in comparability systems (measurement systems). Despite their similar grammar in most human languages, count nouns (e.g., tablets, cars, slots) are not units. Count nouns, do not compare to anything other than themselves and arbitrary collections of themselves. Count nouns are nothing but a natural language construct (that appears quite differently, e.g., in Japanese) count nouns are *not* conceptually part of the quantity.

The following categories exist for quantities (Note the distinction between *kind of quantity and category of quantity*.)

- (1) Ordinal value (e.g., New York Heart Association Heart Failure Index, Dukes Classification of the Colon Carcinoma, etc.). Ordinal values are nothing but ordered, you can say NYHA 4 is worse than NYHA 2, but NYHA 4 it is not (just) "twice as bad" as NYHA 2. Ordinal values are rare in the RIM; if they occur, the committee decides how best to reflect them as data types. Options are:
 - Code value (CV), e.g., for Dukes A, B, and C;
 - Integer (INT), e.g., for a ranking attribute of a collection, prioritized list, or for the typical scale {−, 0, +, ++, +++};
 - Floating point (FLT), e.g., for a ranking with the ability to insert between any two elements;
 - Ratio of INT, or FLT, e.g., for NYHA {1/4, 2/4, 3/4, 4/4} or loudness of heart murmurs in auscultation {1/6, 2/6, 3/6, 4/6, 5/6}.
- (2) Counted objects – cardinal numbers (i.e., non-negative integers,) data type INT ≥ 0.
- (3) Estimations for counted objects – non-negative real numbers, data type FLT ≥ 0.
- (4) Counts (2) or estimated counts (3) allowing for deficits, types INT or FLT respectively, including negative numbers.
- (5) Ratio scales – real numbers (e.g., probabilities,) data type FLT.

(6) Quantities of *comparability systems*

- (a) Physical quantities (real number and unit of measure,) data type PQ. Further distinctions that exist (but are usually ignored) are:
 - (i) Ratio scale vs. difference scale quantities;
 - (ii) Additive (extensive) quantities vs. non-additive (intensive) quantities.
- (b) Monetary quantities (real number and currency unit, always ratio scaled but with volatile exchange rates,) data type MO.
- (c) Calendar date/time, data type PT.

In general, the QTY attribute type shall be used when an attribute describes a quantity. The nature of the quantity must be further determined through the attribute definition, through the choice of a data type, and through further constraints. If the committee has not yet determined the nature of the quantity, the QTY attribute type is appropriate. No data type can be assigned unless the nature of the quantity is determined. Determining the nature of a quantity affects name, definition, and data type of an attribute.

The NBR attribute type (short for “number”) shall be used when the attribute is a numeric ordinal (1), a counted or estimated amount of objects (2-4) or a dimensionless numeric scale (e.g., probability, risk, odds ratio, etc.). (5.) If the dimensionless scale is constrained to the interval [0,1] and denotes a fraction or proportion, the attribute type FRC (fraction) and data type FPN is to be used. The former attribute type PCT for “percentage” is sloppy terminology and no longer allowed.

By default use FPN as the data type for NBR attributes, since for most numbers there are use cases where only estimates are known, and estimates may be fractional even if the actual numbers are integers. Only if fractions, estimates and averages can be positively ruled out, a committee may use the INT data type instead.

After the nature of a quantitative attribute is determined, the QTY attribute type shall be used only for quantities of category (6)(a) and (b), physical and monetary quantities.

Time duration is a physical quantity that can be constrained to the dimension of time (see section 6.3.1.2.2). Distinguishing *time duration* from *point in time* (calendar date/time) is usually easy. Point in time attributes have attribute type DTTM and data type PT. Note also that point in time is often used to form a range or interval (e.g., *begin_dttm / end_dttm*). Such time periods shall be modeled as one attribute of attribute type *time range* (TMR) and data type *Interval of Point_in_time* (IVL<PT>.)

The attribute name must indicate the kind of quantity. The attribute name must not refer to a specific unit of measure.

{<qualifier words>}_<kind of quantity>_q**ty**

For example, use the kind of quantity “volume,” not “liters” or “pints” (“blood_volume_qty” not “blood_pints_qty”). Use the kind of quantity “duration,” not “seconds” or “minutes” (“procedure_duration_qty” not “procedure_minutes_qty.”) It is wrong by all standards to use physical units in plural form. Practically, suggesting a particular unit of measure in the name only leads to confusion in an inter-cultural context (metric vs. customary units). Committees must not constrain physical quantities to any particular unit of measure (e.g., if milliliter are allowed, liter, and U.S. fluid pints, etc. would also be allowed).

For counted objects, the kind of quantity is formed as

{{qualifier words}}_{count noun}_nbr

Count nouns may appear in plural form if this is closest to the natural language form. For example, “family_members_nbr,” “remaining_refills_nbr.”

If counted objects are abstract, the committee may consider a separate attribute that mentions the concrete objects counted (the count noun) as a coded (nominal) value. For example, the abstraction of {*tablet, capsule, suppository, shot*} is a *administration unit* or *application unit*. **An administration unit or application unit is a nominal thing, a count noun, not a unit of measure.** In the pharmacy, the concrete application unit should be mentioned in a separate attribute (e.g., called “dose_form_cd”). This is not because count nouns are somehow part of the counted quantity, but because *dose form* is an important concept in pharmacy information.

In hospital administration it is a common business rule *not* to measure inpatient stay duration as elapsed time, but to count the number of days in various incomparable, incompatible, and non-reproducible ways (e.g., number of high-noons passed, number of midnights passed, number of full days rounded off, etc.). In general a committee should avoid reflecting the idiosyncrasies of business rules directly in the Information Model, since this leads to problems when trying to apply HL7 work products in other “cultural” contexts (e.g., internationally). Instead of simply using a business concept believed to be well understood by some people, a committee should try to further analyze the concept to distill a concept invariant in a worldwide and timeless extent.

However, a committee may reflect concepts originating in national regulations (e.g., HIPAA, UB92, etc.) as RIM attributes. Such borrowed attributes should indicate their origin and purpose in their names and descriptions. If possible, attributes related to one regulation and purpose should be bundled into some class specific to the regulation. Ideally, this special class (e.g., UB92 data) might be a subclass of an invariant concept, but it may also be attached by an optional one to one association (e.g., to the Encounter class).

If business rules require phenomena that are usually measured to be somehow counted (e.g., length of stay,) the attribute name may exceptionally contain the unit name in the attribute name. Especially if the attribute name is borrowed from some regulation, the usual style may be overruled. The attribute type NBR shall be used for such counted attributes rather than QTY. The data type of such NBR attributes shall be INT if counting is strict or FLT if estimates/averages are needed.

Since such count attributes are business-rule dependent and are not proper measurements, another attribute of attribute type QTY and data type PQ may need to be added, to allow communicating a proper measurement value that is independent from such business rules.

Example: Borrowed attribute “HCFA_inpatient_days_nbr,” data type INT. Another attribute, “Encounter.duration_qty” of type PQ (constrained to time,) would carry a proper measurement value of the length of stay that is independent from counting rules. Note that committees must still properly define such quantitative attributes, e.g., concerning the question whether leave of absence time would be counted in to such a value or not. If leave of absence is disregarded, the attribute would not be needed at all, since it would be calculated from an “Encounter.tmr : IVL<PT>” attribute that has more information.

6.3.1.2.2 Constraints

Constraints narrow down the set of possible values that an attribute can take on. Constraints include vocabulary domain constraints (e.g., this attribute’s code value must be from table XYZ), range constraints (e.g., this attribute must be a floating point number between 0 and 1) and more. Before constraints can be

specified, data type must be assigned, since constraints are either predicates on data type values or subsets of the data type's domain.

Constraints in the information model should primarily define logical constraints. Logical constraints enforce meaningful utterances. Committees should define constraints conservatively. Constraints can implement business rules too; however, a committee should be aware that business rules may differ between institutions and especially between countries. The difference between a logical constraint and a business rule constraint is that if a logical constraint is violated, the utterance is meaningless. Conversely, when a business rule is violated, the utterance may still be meaningful, it just does not conform to the business rule.

In the absence of a formal constraint specification language, committees should formulate their constraints in natural language. A committee may invent its own stylized notation, but must declare the meaning of the symbols used. Conventional mathematical symbols and the UML *Object Constraint Language* (OCL) may also be used.

The constraints can be specified through predicates or through specification of a sub-domain of the data type. Examples for a predicate statements would be:

Example 1:

Natural language:

“If for an Encounter the status_cd includes *discharged* (meaning that the Encounter is in discharge state) then the discharge_dttm must have a value.”

Mathematic formalism:

$\forall e \in \text{Encounter} : \text{discharged} \in e.\text{status_cd} \rightarrow e.\text{discharge_dttm} \neq \text{null}$

Object Constraint Language:

```
context e : Encounter inv:
if e.status_cd->includes( discharged )
then
    e.discharge_dttm <> null
endif
```

Example 2:

Natural language:

“The discharge_dttm of an Encounter must be later than its admission_dttm.”

Mathematic formalism:

$\forall e \in \text{Encounter} : e.\text{discharge_dttm} \neq \text{null} \rightarrow e.\text{discharge_dttm} > e.\text{admission_dttm}$

Object Constraint Language:

```
context e : Encounter inv:
if e.discharge_date <> null
then
    e.discharge_dttm.laterThan(e.admission_dttm)
endif
```

Vocabulary domain constraints can be expressed either through predicate statements on the code_system component of the code value:

Example 3:

Natural language:

“For a Result the type_cd must be a LOINC code”

Mathematic formalism:

$\forall r \in \text{Result} : r.\text{type_cd} \in \text{LOINC}$.

Object Constraint Language:

```
context r : Result inv:  
r.type_cd.code_system = 'LN'
```

Note that the above example may be a business rule that should not be specified in the RIM in order to not restrict the usability to a particular clinical vocabulary that may not be accepted everywhere. Vocabulary domain constraints are handled in Chapter 7.

In the HMD vocabulary constraints are defined in a different column than common constraints.

A specific kind of constraint is often used for physical quantities (data type PQ). Physical quantities often need to be constraining to a certain dimension, such as time. Constraints to a particular dimension are expressed by naming one unit of that physical dimension as a representative for the dimension. Note that physical quantities *must not* be restricted to only that particular unit (e.g., only kg but not mg, or only second but not minute, etc.). All units in the same physical dimension must be permitted. This type of constraint is most often used for time duration, that is a PQ constrained to the dimension of time, represented by the unit second or any other unit of time.

Example 4:

Natural language:

“For an Order the duration_qty must be a physical quantity in the dimension of time.”

Mathematic formalism:

$\forall o \in \text{Order} : o.\text{duration_qty} \sim 1 \text{ s}$.

Object Constraint Language:

```
context o : Order inv:  
o.duration_cd.comparesTo( PhysicalQuantity(1, 's') )
```

Attribute value constraints specified in the RIM apply to an attribute in all messages containing that attribute. Additional constraints may also be specified in the MIM and the HMD. Constraints specified in the MIM only apply to the messages derived from that MIM. Additional constraints must still conform to constraints specified on higher levels. For instance, the HMD can not undo a constraint specified in the RIM. If committees find themselves inclined to weaken constraints specified elsewhere, they must take steps to resolve the problem on a higher level.

6.3.1.2.3 Default values

A committee may designate a default value for an attribute in the RIM, MIM, or HMD. Defaults must comply to the domain of that attribute. Defaults may be specified for components of composite data types where the data type is used for an attribute. Defaults may override other defaults with the default specification closest to the implementation taking precedence. The default of all default is the null value “no information.”

Defaults should be specified in the information model for logical reasons. A default should not be specified to capture the common case given certain business rules. A default is appropriate where deviation from the default is the exception from a logical point of view.

6.3.1.3 Define Class Behavior

Identifying class states and state transitions specifies behavioral characteristics of a class. Class states and state transitions are defined for subject classes. A subject class is a class the committee designates as the central focus of a collection of messages. A committee can declare a subject class at any time.

To develop a state model the Use Cases that affect the subject class are analyzed. Approximately each Use Case represents a transition. For each use case, the committee discovers important pre-conditions and post-conditions based upon attributes and associations of the subject class. For example, analysis of the Use Cases:

- plan activity
- revise activity
- begin activity
- abort activity
- finalize activity

suggest that Activity would be a good candidate for a subject class. If all the above use cases are transitions, then at each end of a transition there will be a state. The question is, which of those states are identical? In other words: how do the transitions connect to each other? The committee can now try to identify the order of those transitions, and their dependencies. The committee can discuss the evolving state model by asking questions such as

“Can an activity begin without there being a plan?”

—and depending on the committee’s understanding about activities, it may decide that all activities should be planned. Therefore, the committee would decide to order “plan activity” and “begin activity” to be arranged serially with a common state. The committee decides to name this state “New.” The post-state of “begin activity” could be called “In Progress.”

The next question could be

“When can an activity be revised?”

—and the committee may decide that only activities that are not yet in execution could be revised. Since pre- and post-state of a revised activity is not different from an activity that was originally planned that way, and since a planned activity might be revised multiple times, the committee decides to make the revision a recursive transition over the “New” state.

The committee might now agree that the successful completion of an activity is a noteworthy state reached through the “finalize activity” transition. It would further determine that an aborted activity is in a different state than a finalized activity.

Then the committee could ask

“What activities can be aborted?”

—and decides that an activity can be aborted while in the “New” state or in the “In Progress” state. The committee now has a choice: it could model the Use Case “abort activity” as two transitions, one starting at “New” and the other at “In Progress.” In order to decide whether those are really two different transitions the committee would discuss the consequences of aborting activities out of either state. One conclusion might be that it doesn’t matter, i.e., that an “activity aborted” is one state. Since the committee decides that the pre-state of the “abort” transition does not matter, it could define a super-state for an activity that is either “New” or “In Progress.” And so on, which could result in a state transition model similar to .

Another approach to build a state-transition model is to first identify all the relevant states. From a list of Use Cases states can be identified by building the participle of the verb that labels the Use Case. For instance: “plan activity” → “planned,” “abort activity” → “aborted, ” etc. The committee could then list all the states so discovered in a quadratic matrix. For each cell of the matrix the committee would ask whether a transition from the row’s state to the column’s state of the matrix could be useful. Especially the committee would identify preconditions of Use Cases in terms of the identified states. This approach can also be used to check a state transition model identified by other means.

However, transitions should be defined only where they are conceptually justified, e.g., where there are business processes or logic needs that warrant the transition. Transitions should not be defined for purely formal reasons. Notably, a state transition model where every state can be reached from every other state is wrong modeling. Such a model would not specify anything and could just as well not exist

When all states are discovered, the committee may define constraints for the attributes and associations of the subject class for each one of the states. The description of a state should answer the questions:

- Which attributes must be valued?
- Which attributes must not be valued?
- What domain constraint must be used for valued attributes?
- What associations must be established?
- What associations must not exist?

Attributes and associations from classes other than the subject class may be used in the description of a state for a subject class. However, the chain of associations and intermediate classes must be included also.

Only those states should be defined that are necessary to support the definition of transitions used to define messages. Such states and transitions that are never communicated with other systems do not belong into a model that defines communication between systems. If a committee wants to define messages, it should model the events triggering those messages as transition of one of their subject classes.

Technical considerations of how messaging systems are implemented need not be reflected in the state-transition models of the information model. Especially response messages that merely acknowledge the receipt of other messages need not be modeled, if those messages do not carry relevant application level information.

A state-transition model can be revised later. Especially useful are modifications that extend the model without rendering it incompatible. Such changing is called refinement. Refinement can be used to add backwards compatible extensions but also to reconcile state-transition models of classes related through generalizations. Refinement operations involve defining new states as sub-states of existing states.

If a committee decides to generalize two subject classes in a generalization hierarchy, it can use the inverse to the refinement operation to discover state-transition models that are abstract enough to be brought in conformance with each other. Reconciling state-transition models is a critical step that will help the committee to understand the nature of the generalization relationship better. It may turn out that the generalization hierarchy was not the best choice. If the committee is sure, however, that a generalization hierarchy is justified conceptually, it should consider revising the state-machines in a more radical way to reconcile them.

6.3.2 Update/Harmonization of the Reference Information Model

Changes to the Reference Information Model follow a process designed to maintain accountability for RIM content while inviting opportunities for input from interested Technical Committees, Special Interest Groups, Other ANSI or ISO sponsored Standards Development Organizations and international interests represented by the International Committee or others. The process for external input to the RIM focuses on the privileges granted the submitter. The set of privileges includes the right to submit proposals, the right to attend the harmonization meeting, the right to vote on acceptance of change proposals, and the right to be a steward for classes in the Reference Information Model.

The following table shows the privileges associated with each potential source of input:

Source of Input	Privileges			
	submit RIM change proposals	attend harmonization meeting	vote on change proposals	have stewardship over RIM classes
Technical Committee	yes	yes	yes	Yes
HL7 International Committee	yes	Yes	yes	No
HL7 Special Interest Group	yes	Yes	no	No
ANSI or ISO accredited SDO	yes	Yes	<i>if the SDO's content is derived from the RIM</i>	<i>if the SDO's content is derived from that RIM class and no TC declares stewardship</i>
Others	<i>proposal must be sponsored by an HL7 TC or SIG</i>	No	no	No

Inputs

- Candidate change proposals for the RIM or MDF
- Most recent list of data types, attribute types, and domains
- Most recent version of the RIM

Procedure Steps:

- Prepare RIM change proposal
- Review RIM change proposal with stewards of affected classes

- Participate in RIM change proposal harmonization meeting

Outputs:

- RIM change proposal
- Updated RIM
- RIM change proposal harmonization meeting notes
- Updated MDF

6.3.2.1 Prepare RIM Change Proposal

RIM change proposals are prepared by a Technical Committee or Special Interest Group in an effort to incorporate in the RIM revisions that have been introduced and discussed in the committee and the committee has agreed should be applied to the RIM. Organizations external to HL7 may introduce change proposals and critiques of the RIM by submitted the proposal/critique to the appropriate HL7 technical committee or special interest group for their consideration.

The Modeling and Methodology Technical Committee (MnM) will provide assistance to external organization in preparing their proposal and in identifying the appropriate steward technical committees. The steward technical committee may choose not to sponsor the change proposal, if it does the proposal essentially dies in committee. Changes submitted by ANSI or ISO accredited standards development organizations (SDOs) are unconditionally sponsored by the Modeling and Methodology Technical Committee and may be submitted for harmonization even if rejected by the steward technical committee when the sponsoring SDO finds the technical committee's rationale for rejection to be non persuasive.

All changes made during construction and refinement of the DIM are candidate change proposals for the RIM or MDF. A proposed change to the RIM need not actually be applied to the committee's DIM, but it must have been introduced in committee and approved for submission. RIM change proposals are prepared by or with the assistance of the committee's modeling facilitator.

A change proposal may contain more than one revision to the RIM. Inter-dependent revisions should be submitted on the same change proposal. The sponsoring committee assigns each change proposal a unique identifier. A short name or title is assigned to the proposal and a rationale for the change is documented.

The revisions to the RIM are documented in a spreadsheet. The revisions are linked to the change proposal by including a reference to the proposal's identifier. A separate revision must be documented for each model component affected. Each revision must document the before and after information for the revised model component. The revisions are numbered sequentially within the change proposal for ease of reference. New model components must be fully documented.

6.3.2.2 Review RIM Change Proposal with Stewards of Affected Classes

RIM change proposals must be reviewed with the Technical Committee that has stewardship for the affected class(es). Each class in the RIM is assigned at least one and at most two Technical Committee stewards. Stewardship for new classes is temporally assigned to the committee adding the class. Other Technical Committees may petition for stewardship of a new class. If more that two Technical Committees petition for stewardship of the same class, the Modeling and Methodology (MnM) Technical Committee will assist in resolving the allocation of stewardship. A second steward may not be added to a class without the approval of the current steward. Stewardship for a class can not be transferred between Technical Committees without the consent of both committees and notification of MnM. The steward Technical Committee is given first opportunity to comment on proposed changes to their allocated classes.

For each RIM revision associated with a change proposal, identify the class(es) affected by the revision. For each class identify the steward Technical Committee. If the steward Technical Committee is not the same as the sponsoring committee, the sponsoring committee must arrange to obtain comments regarding the proposed revisions from the steward committee. The steward committee may favor or oppose the change. Comments from the steward committee are captured on the RIM change proposal form.

Once the RIM change proposal has been reviewed with the stewards for the classes affected, the sponsor committee may forward the change to the Modeling and Methodology Technical Committee for posting and tracking. The sponsor committee may be persuaded by the stewards to revise or withdraw the RIM change proposal.

RIM change proposals are posted on the HL7 web-site at least 30 days prior to the scheduled harmonization meeting. A notice is sent via email to the HL7 membership, Technical Committee chairs, and the modeling facilitators. Comments on the change proposal may be submitted directly to the sponsoring committee or to any of the appropriate HL7 list servers (either the steward committee's or the MNM list server). The MnM list is the preferred list for comments regarding proposed changes to the RIM.

6.3.2.3 Participate in RIM Change Proposal Harmonization Meeting

The Modeling and Methodology Technical Committee schedules the RIM change harmonization meeting to occur between working group meetings. In scheduling the meeting, consideration is given to allow sufficient notice to potential attendees, to allow at least 30 days review of posted change proposals, and at least 30 days to apply and document accepted changes to RIM.

Any HL7 member may attend the RIM change proposal harmonization meeting. Modeling facilitators and at least one steward representative per Technical Committee are encouraged to attend the meeting. Representatives from other ANSI or ISO accredited organizations are welcome to attend the harmonization meeting to introduce and discuss change proposals they have submitted.

The co-chairs of the Modeling and Methodology Technical Committee facilitate the meeting. The sponsor of a RIM change proposal should be present to introduce, discuss, and answer questions concerning the proposed change. If the sponsor is not able to be present, it may designate a proxy to introduce the change or a representative from one of the affected steward committees may introduce it; otherwise the proposal is tabled.

Discussion of a proposal may lead the representative from the sponsoring committee to revise, table, or withdraw the proposal. Proposals that are neither tabled nor withdrawn are put to a vote. There is one vote per Technical Committee. To approve a change proposal that is supported by the affected steward committees requires a simple majority. To approve a change proposal that is opposed by an affected steward committee requires a 2/3 majority for approval.

Approved changes are applied to the RIM. Tabled changes may be re-introduced at a subsequent harmonization meeting. Rejected and withdrawn changes are returned to the sponsoring committee with comment. The disposition of all changes is noted on the change proposal. Notes from the harmonization meeting summarize the discussion and disposition of each RIM change proposal.

A RIM change proposal may lead to revision of the MDF. Approved RIM changes which introduce new attribute types or data types will require updates to the MDF, as will approved changes which lead to the addition, revision, or removal of a style guideline.

6.3.3 Construction of the Message Information Model

Inputs:

- Most recent version of the RIM
- Use Case Model
- Interaction Model

Procedure

- Extract model components from the RIM
- Revise association constraints
- Define additional attribute constraints

Outputs:

- A Message Information Model
- Proposed new attribute domain specifications

6.3.3.1 Extract Model Components From the RIM

A Message Information Model (MIM) is a subset of the Reference Information Model (RIM). The MIM is an expression of the information content for one or more related messages. It is a subset of the RIM containing only those model components relevant to the messages to be derived from it.

The subset of the RIM is constructed by first identifying the required classes and associations. Next the required attributes, states, and state transitions are extracted.

6.3.3.2 Revise Association Constraints

The multiplicity on associations in the MIM may be revised from their value in the RIM to increase the strength of constraint (hardening the constraint). The hardened constraint always conforms to the original constraint. Let the multiplicity at one end of an association originally be specified as $n_1..n_2$. A hardened multiplicity is any multiplicity $n'_1..n'_2$ with $n'_1 \geq n_1$ and $n'_2 \leq n_2$. Commonly multiplicities are hardened as follows:

- $0..* \rightarrow 0..1, 1..*, \text{ or } 1..1$;
- $0..1 \rightarrow 1..1$;
- $1..* \rightarrow 1..1$.

As with RIM-level constraints, multiplicities and other constraints in the RIM should be logical and necessary for the meaning and proper function of the messages. Constraints should not restrict the use of models and messages to certain business rules that might be different at other places or other times.

6.3.3.3 Define Additional Attribute Constraints

Attributes may be specified as mandatory in the MIM even if not specified as mandatory in the RIM. A mandatory attribute in the MIM indicates that the HMD must use that attribute in all messages in which the class is used. If an attribute is mandatory in the RIM then it must also be mandatory in the MIM. Attributes in the MIM must have the same data type as in the RIM. This includes its being or not being a

collection (set). A type *t* specified in the RIM may not become a SET<*t*> specified in the MIM and vice versa.

Constraints may be specified for attributes in the MIM. If specified in the RIM, a constraint applies for an attribute in all messages containing the attribute. If specified in the MIM, a constraint applies to all of the messages derived from that MIM. Constraints may also be specified within the HMD where they can be made specific to a particular message.

Vocabulary domains are used to specify constraints on the allowable values of a coded attribute. If an attribute in the MIM needs to be constrained more stringently than it is in the RIM, and that constraint is consistent for all messages drawn from the MIM, then a vocabulary domain may be specified within the MIM for that attribute. As with any other constraints, a vocabulary domain specified for the MIM must conform with (i.e., must be a subset of) a domain specified in the RIM.

6.4 Summary of Information Model Style Guidelines

This section of the HL7 Message Development Framework is a summary of stated guidelines that have been explained in this chapter. This concise summary provides fast guidance for the development of information models for use within HL7. It can be used as a checklist to test for compliance to the rules. The objective sought in creation of this section is to minimize style variation among the information models developed in the Technical Committees so as to facilitate the harmonization of committee models into the HL7 Reference Information Model.

The need for these style guidelines stems from the premise that information modeling permits an information concept to be correctly represented in a multiple ways. By adopting the modeling style outlined in this document it is hoped that Technical Committees will produce data models which express information concepts in a consistent manner. The process of harmonizing the Technical Committees' Domain Information Models into the Reference Information Model could then focus on conceptualization conflicts between DIMs and not be encumbered by style differences.

6.4.1 Classes

1. All new classes must either be associated with a class already in the DIM or participate in a chain of classes and relationships that eventually lead to a class already in the DIM.
2. All classes must have a description which clearly identifies what the class is, and which provides instance qualification criteria and examples as needed.
3. The singular forms of nouns are to be used for class names.
4. The use of conjunctions, such as "or" and "and" in class names is to be avoided.
5. A class name may be composed of multiple words separated by an underscore ("_").
6. Class names must be concise; multiple words should only be used only if needed for clarity.
7. Classes that capture an association between two classes should have a name that expresses the nature of the association in a meaningful way.
8. Alternatively, associative classes can take on the name of the two associated classes followed by the suffix "_Assn".

6.4.2 Relationship

6.4.2.1 Generalizations

1. A subclass must be conceptual subcategories of the superclass. Specialization should not be used simply to utilize inheritance. Generalization constructs that fail the “read-it-loud” test must be justified by a written rationale.
2. A subclass should have additional properties (i.e., attributes, relationships, state-transition models) specified beyond those inherited from the superclass.
3. Generalization hierarchies with more than four levels should be avoided.
4. If all possible specializations of a superclass are represented (fully enumerated) in the information model, the specialization should not be reported in a classifying attribute.
5. If the subclasses are not fully enumerated, a classifier attribute must be placed in the superclass to explicitly declare the conceptual (but not modeled) specialization.
6. Inheritance can not be revoked. All properties (i.e., attributes, relationships, state-transition models) must be valid for all subclasses and their descendents. If this violates the conceptual intention of the modelers the model must be revised.

6.4.2.2 Associations and Associative Classes

1. Associations must be defined for a specific purpose that is expressed in the association names (UML role names) written on each end of the association.
2. Each association name shall be a short verb phrase in indicative mode and present tense, written from the perspective of the class on that end of the association.
3. Association names shall be in lowercase letters with underscores (“_”) separating the words.
4. A multiplicity must be specified for each end of an association.
5. Valid association multiplicities are “ $n_1..n_2$ ” where n_1 and n_2 are cardinal numbers (non-negative integers) and $n_1 \leq n_2$. Where $n_1 = n_2$ the short form “ n_1 ” is used. An asterisk (“*”) indicates an unbound multiplicity.
6. Common multiplicities are “0..1”, “0..*”, “1”, and “1..*”. Any other multiplicity requires written justification.
7. Multiplicity configurations should be checked against the table of section 6.3.1.1.3. (p. 6-22). Deviations from the suggestion require a documented rationale. Deprecated multiplicities should be revised.
8. Reflexive (or recursive) associations (i.e., where both ends connect to the same class) must have a minimum multiplicity of zero on both ends. Transitive recursions through a generalization hierarchy may have minimum multiplicity of 1 at the end of the superclass if an alternative subclass can terminate the recursion.
9. Classes that represent roles assumed by stakeholders, persons, or organizations must connect to the stakeholder hierarchy through an association with names and multiplicities “is_a_role_of (1)” and “takes_on_role_of (0..1).”
10. Classes that represent the participation of a role class in an event must use the association names “is_a_participation_of (1..1),” “participates_as (0..*),” “is_a_participant_in (1),” “has_as_a_participant (any,any).”
11. Composition aggregations must have the names “has_as_part($n_1..n_2$),” “is_part_of (1).”

12. Composite aggregation represents a strong conceptual dependency of the part to the whole and must not be used because of implementation considerations. Composite aggregations that are conceptually justified are rare. When in doubt use a common association.

6.4.3 Attributes, Data Types, Constraints, and Defaults

1. An attribute name is comprised of one or more qualifier words followed by an attribute type suffix.
2. The scope of attribute names is the class. The class name must not be repeated in the attribute name itself.
3. Attribute types must be selected from those defined by the Modeling and Methodology Technical Committee.
4. Only the attribute types description (“descr”,) identifier (“id”,) and name (“nm”,) may be used without a qualifier word.
5. An identifier attribute is used to identify a unique occurrence of a class.
6. Classifier attributes, named “type_cd,” are used in generalization hierarchies that are not fully enumerated. The classifier attribute is placed in the superclass and contains a value declaring the conceptual (but not modeled) subclass.
7. A state attribute is used in subject classes. It is called “status_cd” and of type SET(CV), where each code identifies an active state.
8. Quantitative attributes are of one of the categories shown in Section 6.3.1.2.1.4.
9. Numeric ordinals, counts, estimations/averages of counts, and dimensionless ratio scales shall have attribute type suffix “_nbr” and, generally, shall be of type FPN. Only if estimation and other reasons for choosing the FPN data type are positively ruled out, the INT data type shall be used.
10. Dimensionless ratio scales that represent a fraction or proportion shall have the attribute type suffix “_fre” and data type FPN constrained to the interval [0;1]. The former attribute type “_pct” and other intervals (e.g., [0;100]) are no longer permitted.
11. Attributes representing counted values or estimates or average of counts, shall indicate the kinds of objects counted in their name, that ends in the attribute type suffix “_nbr.” Count nouns are not units of measure. An additional coded attribute should be defined when the things counted need to be specified dynamically.
12. Point in time attributes (calendar date/time) are of attribute type “_dtm” and data type “PT.” Time periods shall be represented by a single attribute of attribute type suffix “_tmr” and data type “IVL(PT).” Parallel attributes to represent begin and end date/time are no longer permitted.
13. Physical quantities and monetary amounts shall have the attribute type suffix “_qty” and data types PQ or MO respectively. Such attributes must show their kind of quantity in the attribute name. Units of measure may not appear in the attribute name. Constraints to any one particular unit of measure are forbidden.
14. Attributes borrowed from common business rules or regulations not under the influence of HL7 may have names and properties overruling this style guide. Particularly, if usually measured values are counted using some special rule, the attribute name may contain the particular unit counted, should

have attribute type “_nbr,” and may be of data type INT. The origin and limited use of such attributes is to be clearly marked in the attribute’s class, name, and definition.

15. An attribute may be added to the information model without specifying the data type and domain. However, a data type must be specified for attributes in the RIM prior to their inclusion in an HMD.
16. Data types assigned to attributes must be selected from those defined or approved by the Control/Query Technical Committee.
17. The same data type must be used for an attribute in all messages containing the attribute.
18. Constraints apply for all derivatives of a model (i.e., data types > RIM > DIM > MIM > HMD > message type).
19. Constraints must express assertions mandated by the conceptual logic of the modeled application domain. Constraints should not limit the applicability of the model to a particular set of business rules.
20. Defaults can be set in any model. Defaults must be allowed values for the attribute in that model, i.e., defaults must conform to any constraints. Defaults may override defaults set in higher level models (i.e., data types > RIM > DIM > MIM > HMD > message type > message instance).

6.4.4 States and State Transitions

1. Each state transition should be associated with a leaf level use case. More than one transition may be associated with the same use case.
2. Transitions should not be defined for purely formal reasons. Notably, a state transition model where every state can be reached from every other state is wrong modeling. Such a model would not specify anything and could just as well not exist
3. States should have a description, that should answer the questions:
 - Which attributes must be valued?
 - Which attributes must not be valued?
 - What vocabulary domain constraint must be used for valued attributes?
 - What associations must be established?
 - What associations must not exist?
4. Attributes and associations from classed other than the subject class may be used in the description of a state for a subject class.
5. State transition models are subject to inheritance. If specialized state-transition models must be refinements of the super class’ state-transition model. This may require revising the state-transition model of the superclass.

6.4.5 Prepare RIM Change Proposal

1. RIM change proposals must have been introduced, discussed and approved by the sponsoring committee.
2. RIM change proposals are prepared by or with the assistance of the committee’s modeling facilitator.
3. A change proposal may contain more than one revision to the RIM. Inter-dependent revisions should be submitted on the same change proposal.

4. The sponsoring committee assigns each change proposal a unique identifier.
5. A short name or title is assigned to the proposal and a rationale for the change is documented.
6. The revisions to the RIM are documented in a spreadsheet.
7. The revisions are linked to the change proposal by including a reference to the proposal's identifier.
8. A separate revision must be documented for each model component affected.
9. Each revision must document the before and after information for the revised model component.
10. The revisions are numbered sequentially within the change proposal for ease of reference.
11. New model components must be fully documented.

6.4.6 Review RIM Change Proposal with Stewards of Affected Classes

1. RIM change proposals must be reviewed with the technical committee that has stewardship for any affected class.
2. Each class in the RIM is assigned at least one and at most two technical committee stewards.
3. Stewardship for new classes is temporally assigned to the committee adding the class.
4. Other technical committees may petition for stewardship of a new class.
5. If more than two technical committees petition for stewardship of the same class, the Modeling and Methodology (MnM) technical committee will assist in resolving the allocation of stewardship.
6. A second steward may not be added to class without the approval of the current steward.
7. Stewardship for a class can not be transferred between technical committees without the consent of both committees and notification of MnM.
8. The steward technical committee is given first opportunity to comment on proposed changes to their allocated classes.
9. If the steward technical committee is not the same as the sponsoring committee, the sponsoring committee must arrange to obtain comments regarding the proposed revisions from the steward committee.
10. The steward committee may favor or oppose the change.
11. Comments from the steward committee are captured on the RIM change proposal form.
12. Once the RIM change proposal has been reviewed with the stewards for the classes affected, the sponsor committee may forward the change to the Modeling and Methodology technical committee for posting and tracking.
13. The sponsor committee may be persuaded by the stewards to revise or withdraw the RIM change proposal.
14. RIM change proposals are posted on the HL7 web site at least 30 days prior to the scheduled harmonization meeting.
15. A notice is sent via email to the HL7 membership, technical committee chairs, and the modeling facilitators.
16. Comments on the change proposal may be submitted directly to the sponsoring committee or to any of the HL7 e-mail list servers.
17. The MnM list is the preferred list for comments regarding proposed changes to the RIM.

6.4.7 Participate in RIM Change Proposal Harmonization Meeting

1. The Modeling and Methodology technical committee schedules the RIM change harmonization meeting to occur between working group meetings.
2. In scheduling the meeting, consideration is given to allowing sufficient notice to potential attendees, to allow at least 30 days review of posted change proposals, and at least 30 days to apply and document accepted changes to RIM.
3. Any HL7 member may attend the RIM change proposal harmonization meeting.
4. Modeling facilitators and at least one steward representative per technical committee are encouraged to attend the meeting.
5. The co-chairs of the Modeling and Methodology technical committee facilitate the meeting.
6. The sponsor of a RIM change proposal should be present to introduce, discuss, and answer questions concerning the proposed change.
7. If the sponsor is not able to be present, it may designate a proxy to introduce the change or a representative from one of the affected steward committees may introduce it, otherwise the proposal is tabled.
8. Discussion of a proposal may lead the representative from the sponsoring committee to revise, table, or withdraw the proposal.
9. Proposals that are neither tabled nor withdrawn are put to a vote.
10. There is one vote per technical committee.
11. To approve a change proposal, which is supported by the affected steward committees, requires a simple majority.
12. To approve a change proposal that is opposed by an affected steward committee requires a 2/3 majority for approval (rounding up).
13. Approved changes are applied to the RIM. Tabled changes may be re-introduced at a subsequent harmonization meeting.
14. Rejected and withdrawn changes are returned to the sponsoring committee with comment.
15. The disposition of all changes is noted on the change proposal.
16. Notes from the harmonization meeting summarize the discussion and disposition of each RIM change proposal.
17. A RIM change proposal may lead to revision of the MDF.
18. Approved RIM changes which introduce new attribute types or data types will require updates to the MDF, as will approved changes which lead to the addition, revision, or removal of a style guideline.

6.4.8 Define Message-set Specific Association Constraints

1. A Message Information Model (MIM) is a subset of the Reference Information Model (RIM). A MIM must conform to the RIM and all of its multiplicities and constraints.
2. All constraints in the RIM may be hardened.
3. Multiplicities are hardened through increasing the lower bound or decreasing the higher bound.
4. Constraints in the MIM apply to all derivatives of that MIM but do not affect the RIM.
5. Attributes may be specified to be “mandatory” in the MIM, which indicates that the HMD must use that attribute in all messages in which the class is used.

6. Data type assignments do reflect into the RIM. The MIM can not change a predefined data type. Especially a MIM can not change a type *t* to a collection of *t* or vice versa.
7. Defaults may be overridden and must conform to all applicable constraints. Especially a default must conform to any hardened constraints. Thus, when constraints are hardened in the MIM or HMD, any default must be reviewed to assure conformance.

7. Associating Vocabulary Domains with Attributes, Elements, and Fields

7.1 Vocabulary Domains

7.1.1 General disclaimer

This chapter represents the current thoughts on domains and domain specifications for use in HL7 messages, and the current design of the domain specification database. It does not represent an immutable final consensus on these issues. This chapter and the domain specification database will be changed and enhanced as appropriate based on actual experience in building and populating the domain specification database.

7.1.2 Introduction

The basic idea of vocabulary domains is intuitively quite simple. A vocabulary domain is the set of allowed values for a coded field. If a message instance contains a marital status field, then the vocabulary domain for that field would consist of the allowed values for that field, such as, *single*, *married*, *divorced*, and *separated*.

The reason for defining a vocabulary domain for coded fields is to strengthen the semantic understanding and computability of coded information that is passed in HL7 messages. The assumption is that if there is a public definition of what values a coded field can take, systems on the receiving end of an interface will have a better idea of how to store and use the data within their systems. The hope is that the coded data can be used in direct patient care, outcomes analysis and research, generation of alerts and reminders, and other kinds of decision support processes.

7.1.3 Vocabulary Domains, and Vocabulary Domain Specifications

Taking the intuitive notion of vocabulary domains as a base, it is, however, important to have a more formal definition of a vocabulary domain. Within the HL7 message framework, a vocabulary domain is the set of all *concepts* that can be taken as valid values in an instance of a coded field or attribute. For example, referring to the RIM, the *Person* class has a coded attribute *gender_cd*. If the *gender_cd* attribute becomes part of a hierarchical message definition (HMD), and a message instance is subsequently created as part of an implemented interface, the *gender_cd* field might have as possible values the concepts *male*, *female*, *other*, and *unknown*. A concept is defined by ISO 1087 as a “unit of thought constituted through abstraction on the basis of characteristics common to a set of objects.”¹ In the roster style of set notation, the Gender vocabulary domain can be represented as $\text{Gender} = \{\text{male, female, other, unknown}\}$. It is important to note that a value domain consists of a set of *concepts*, not a set of *words* or *codes*. In different implementations of an interface, the same concept could be represented using different coding systems. Thus, each concept in a vocabulary domain has a one to many relationship to codes that might be used as representations for the concept in a message instance. The representation of coded values in message instances is explained in the description of the Version 3 data types.

The general meaning of *code system* is a scheme for representing concepts using (usually) short concept identifiers to denote the concepts that are members of the system. Further information about concepts and coding schemes can be found in *Medical Informatics — Categorical structure of systems of concepts — Model for representation of semantics*.² We use the term *code system* within this document as defined in the Object Management Group - Lexicon Query Service document³: *A coding scheme defines and/or describes a set of one or more concept codes. These codes are unique within the namespace of the coding scheme, and are globally unique when coupled with the name of the coding scheme itself.* By this definition, the HL7 Version 2.X tables (taken together) are not a code system because within the namespace of "HL7", "M" could mean

Male or Married. However each individual HL7 table is a code system since within a given HL7 table the code does have a unique meaning. So "M" means only Male within the namespace of the HL7 Gender table and it means only Married within the namespace of the HL7 Marital Status table. This definition is important to remember as the details of the domain specification database are discussed later.

Each coded attribute in the RIM will be associated with one and only one vocabulary domain. The association between a RIM attribute and a vocabulary domain is made via a *vocabulary domain specification*. In other words, each coded RIM attribute will itself have a vocabulary domain specification as a property. A vocabulary domain specification consists of two main parts: the name of the vocabulary domain, and a list of zero or more *vocabulary domain qualifiers*. There are presently only two vocabulary domain qualifiers: *Extensibility* and *Realm*. Currently, the Extensibility qualifier is the only qualifier that can be used in domain specifications. Both the Realm and Extensibility qualifiers can be used in domain constraints, which will be described later. The Extensibility qualifier has two possible values: CNE (coded no exceptions), and CWE (coded with exceptions).

The CWE value for the Extensibility qualifier means that when a coded attribute is sent in a message, local concepts or free text may be sent in place of a standard code if the desired concept is not represented in the standard vocabulary domain. Please see the HL7 Version 3 Data Types Specification for a discussion of the structure of coded data elements in Version 3 messages. Also, "The structure of coded elements in messages" section of this chapter shows examples of how to send local codes or free text as part of a coded value instance. Almost all coded elements in HL7 messages would use the CWE qualifier.

The CNE value for the Extensibility qualifier means that a concept from the specified domain *must* be sent as the value of the coded field in a message. If the field can not be valued from the concepts that exist in the specified domain, the field can not be placed in the message. If a CNE field is required in a message, but the field can not be valued from the concepts that exist in the specified domain, then no valid message can be created. Use of the CNE qualifier is quite rare. It is only used when a given coded field is essential for parsing or understanding subsequent parts of a message.

The other qualifier, *Realm*, can be used in domain constraint statements, but not in domain specifications. The Realm qualifier allows the domain of a coded attribute to be specialized according to the geographical, organizational, or political environment where the HL7 standard is being used. For example, the Realm qualifier would allow the Gender domain to hold a somewhat different value set for HL7 messages when used in Japan versus when the Gender domain is used in HL7 messages in the United States. The meaning of the Realm qualifier will be discussed further in the domain constraint section of this chapter.

All domain qualifiers are members of the VocabularyDomainQualifier domain. The general form of a vocabulary domain specification is: <domain name, list of domain qualifiers>. The value of a domain qualifier is separated from the name of the domain qualifier by a colon (":"). Continuing the example from above, the vocabulary domain specification for gender_cd would be "<Gender, Ext:CWE>."

7.1.4 Validating Vocabulary Domain Specifications and Constraints

Ideally, vocabulary domains would be validated for each coded field each time a message is created or decomposed or interpreted. However, it can be time consuming and resource intensive to validate vocabulary domains in messages. For this reason, it may be that domain checking is only invoked selectively at the times when it has the most value. Times that domain validation could be used include:

- During testing and debugging of an interface.
- During conformance testing of an interface.
- During message creation.
- Upon receipt and decoding of a message.

- Only on the most important coded fields in the message.
- Only when errors or unexpected behaviors occur relative to the contents of a message.
- All of the above.

7.1.5 Vocabulary Domain Constraints

The vocabulary domain specifications stated in the RIM always refer to a complete vocabulary domain. That is, at the RIM level there is no specialization based on realm of use or the context and needs of a specific message. As RIM attributes are specialized to suit a specific message context, the domain of the attribute can be reduced (constrained) to reflect the specialization. A domain that has been constrained to a particular realm and vocabulary is called a *value set*. The vocabulary domain associated with a coded attribute used at any level of specialization below the RIM must be a subset of the vocabulary domain specified for the attribute in the RIM. A vocabulary domain constraint is an expression that states how the sub-domain (value set) was derived from the domain specified in the RIM. Initially the plan was to allow "in place" specialization of domains. That is, a domain constraint expression would be contained directly in the MIM, HMD, or clinical template.¹ However, to make it easier to maintain and reuse domains and value sets, the current plan is to have the MIM, HMD, or clinical template only contain the name of the value set and its list of domain qualifiers. This means that domain constraint expressions are only contained in the Value Set Definition Table. *See the examples in the Expression column of the Value Set Definition Table for further clarification.*

7.1.5.1 Rules for Creating Vocabulary Domain Constraints

The general rule is that a domain can be reduced in scope as it is specialized for a particular use, but its semantic scope can never be expanded. The application of the general rule results in the following specific rules.

- The vocabulary domain of a coded element or attribute used at any level of specialization below the RIM must be a subset of the vocabulary domain specified for the attribute in the RIM. Note that for attributes that are qualified with an extensibility of CWE, local codes *are* an allowed extension to the domain, but this is not intended as an increase in the semantic scope of the attribute. The idea is that if the standard domain for *Hair Color* contained *blonde*, *brown*, and *black*, it would be valid to add a local code for *gray*. However, it would be considered invalid to add a local code that meant *bald*. *Baldness* is not a hair color, it is a state of not having hair. *See also the discussion of the use of local codes later in this chapter.*
- Once the Extensibility qualifier value CNE has been invoked at any level of attribute domain specification (RIM, MIM, MET, CMET, HMD, or clinical template), the CNE qualifier must be retained in any subsequent domain constraints. For example, if a coded attribute has a CNE qualifier in the RIM, any MIM, MET, CMET, HMD, or clinical template must also have a CNE qualifier associated with the domain of that attribute. If a vocabulary domain in the RIM has the Extensibility qualifier value of CWE (coded with exceptions), a subsequent constraint of that attribute's domain can have either the CNE or CWE qualifier.

7.1.6 The Domain Specification Database

Vocabulary domain specifications are only useful if they can be resolved to a list of allowed values for the coded attribute to which they are attached. All vocabulary domain names are resolved against a set of HL7 tables that, taken together, contain the definitions of the various vocabulary domains, or else they point to

¹ Clinical templates are not discussed or defined in this document. For further information about clinical templates, contact Stan Huff.

external vocabularies where the vocabulary domain can be resolved. The assumption is that any process that needs to resolve a vocabulary domain name to the list of concepts that the vocabulary domain contains can do so by reference to the domain specification database, or by reference to external vocabulary tables or services.

There are four primary data tables in the vocabulary domain specification database: the *Value Set Definition Table*, the *Value Set Relationship Table*, the *Source Vocabulary Representation Table*, and the *Observation Identifier to Value Set Linking Table*. A brief description of each table will first be given, and then subsequent sections will give exact structures currently proposed for each of the four types of tables. In addition to the four primary tables, there is a version tracking table that captures audit/management information about edits on any of the tables in the domain specification database. The structure of the version tracking table will be described in association with the description of the Value Set Definition Table.

The *Value Set Definition Table* describes the high level definition of a given vocabulary domain. It holds the name of the vocabulary domain, and shows the relationship of the vocabulary domain to different realms of use, and to the various vocabularies and coding systems that are used to define the vocabulary domain. The structure of this table allows a vocabulary domain (or its value sets) to be defined by recursive reference to other vocabulary domains or value sets.

The *Value Set Relationship Table* records the relationship between HL7 maintained value sets and the concepts that are values within the value set. An HL7 maintained value set can be composed from individual concepts, other value sets, or both.

The *Source Vocabulary Representation Table* shows the relationship between HL7 concepts and codes and descriptions from specific vocabularies. It allows a user of HL7 to see how the concepts in a given domain can be represented within a specific vocabulary/coding system. The structure of this table corresponds roughly to the logical structure of the Version 2.X HL7 vocabulary tables. However, the content of the Source Vocabulary Representation Table can be provided by HL7 or by any vocabulary or terminology provider that wishes to use the HL7 domain specification database as a mechanism for making their vocabulary available for use in HL7 messages.

The *Observation Identifier to Value Set Linking Table* is used to link an observation identifier, like a LOINC code (Logical Observation Identifier Names and Codes), with a value set. This table is used when it is desirable to specify the exact value set that should be associated with a coded observation identifier as used in a service event, assessment, or observation instance.

7.1.6.1 Requirements for the representation of domains

The following requirements for vocabulary domains were used to determine the logical structure of the tables in the vocabulary domain specification database.

- Each vocabulary domain will have a unique, non semantic identifier.
- Each vocabulary domain will have a unique textual name.
- Each vocabulary domain can have a textual description, as well as an edit note.
- There will be version tracking for vocabulary domains.
- Vocabulary domains can be specific to a particular realm of use.

Different realms of use (usually countries, but may be any organizational or political boundary) may have laws or special circumstances that require that the vocabulary domain be unique to that realm of use. This does not mean that vocabulary domains *must* be different in different realms of

use. It is hoped that most vocabulary domains will remain the same even across international boundaries. It is up to HL7 Technical Committee's to approve what realms of use will be allowed within the HL7 standard. When vocabulary domains are different for different realms of use it follows that there could be some loss in interoperability between systems that use realm specific vocabulary domains.

- Value sets (domains that have been specialized by realm and code system) are defined from concepts from a single vocabulary. A corollary to this principle is that defining a value set can not be done without reference to the code system intended to be used in the value set.

There is good evidence that concepts from different coding systems are seldom if ever truly identical, even if the textual descriptions associated with the concepts are the same. The different coding systems may have different definitions, different hierarchies, relationships, or usage rules that result in somewhat different meanings across coding systems. Also, creating vocabulary domains from multiple coding systems would require that interface implementers license multiple coding systems. For these reasons, value sets defined by reference to vocabularies or terminology's external to HL7 will be defined by reference to a single coding system.

- Value sets can be recursively defined.

Many value sets may have a natural hierarchical structure. To simplify creation and maintenance of value sets, value sets may be defined by reference to other value sets. The references can be recursive as long as the result is a directed acyclic graph. That is, a value set can not directly or indirectly refer to itself.

- Set notation will be used to describe how one value set is derived from one or more other value sets.

A UML model of domains and their relationship to coding systems is available as a part of the HL7 RIM.

7.1.6.2 Structure of the Value Set Definition Table

The Value Set Definition Table contains a description of all domains referenced in HL7 specifications. A given domain can be maintained by HL7 or by one or more vocabulary providers external to HL7. If a given domain is owned and maintained by HL7, other value set specifications for that domain that refer to external vocabularies are not allowed.

As noted above, a vocabulary domain that has been specialized in the context of a specific message and placed in the context of a specific Realm and Code System becomes a *value set*. Thus, a domain is the complete set of all concepts that are valid values for an attribute in the RIM, an HMD, a CMET, or a template, while a value set is the subset of concepts from the global domain that are applicable in a given Realm and Code System. A value set is uniquely defined by the combination of a Realm, a Domain Name, and a Code System. Conversely, a domain is the union of all value sets that contain the same domain name.

An example of the Value Set Definition Table is shown below.

HL7 Value Set ID	Domain Name	Realm	Code System	Value Set Description	Expression	Current Status	Vin	Vout
1	Gender	Root	HL7	The gender of a person.	"Gender:Root:HL7"	A	1	
10001	MyGender	Root	HL7	The gender of a person. Does not allow Unknown or Other.	("Gender:Root:HL7" - "O:HL7-0001")	A	1	
60001	ClinicalDiagnosis	USA	SNM3	A clinical diagnosis or syndrome.	ChildrenOf("D*")	A	3	
60002	ClinicalDiagnosis	USA	MED	Diagnosis or syndrome that best explains the patient's symptoms.	SubTypes("1278")	A	3	
70001	BillingDiagnosis	USA	IC9	The billing diagnosis for third party reimbursement purposes.	Any("ICD-9CM")	A	2	
5	Race	Root	HL7	The race of a person.	"Race:Root:HL7"	A	1	
50005	Race	USA	HL7	The race of a person.	"Race:USA:HL7"	A	2	
50006	Race	UK	RC	A person's race.	ChildrenOf("Race")	P	1	
20001	AmericanIndian OrAlaskaNative	USA	HL7	American Indian or Alaska Native Race	"AmericanIndianOrAlaskaNative:USA:HL7"	A	1	
20002	Asian	USA	HL7	Asian Race	"Asian:USA:HL7"	A	1	
20003	BlackOrAfrican American	USA	HL7	Black or African American Race	"BlackOrAfricanAmerican:USA:HL7"	A	1	
20004	NativeHawaiian OrPacificIslander	USA	HL7	Native Hawaiian or Pacific Islander Racer	"NativeHawaiianOrPacificIslander:USA:HL7"	A	1	
20005	WhiteRace	USA	HL7	White Race	"WhiteRace:USA:HL7"	A	1	
30001	AmericanIndian	USA	HL7	American Indian Race	"AmericanIndian:USA:HL7"	A	1	
30002	AlaskaNative	USA	HL7	Alaska Native Race	"AlaskaNative:USA:HL7"	A	1	

Table 7-1. Value Set Definition Table

The Value Set Definition Table has the following columns:

- **HL7 Value Set Identifier** – a unique, sequential number assigned by HL7 that identifies a value set. Each unique combination of a Realm, a Domain Name, and a Code System is given a unique value set identifier. For HL7 tables that exist in Version 2.X, the value set identifier is the same as the Version 2.X table number. The value set identifier has the same meaning across all tables in the domain specification database and is used as a foreign key to allow joins between the tables. The value set identifier comes from the same number space as HL7 concept identifiers. Thus, an HL7 value set will never have the same identifier as an HL7 concept.
- **HL7 Vocabulary Domain Name** – a unique textual name assigned by HL7 for the vocabulary domain. The domain name is retained across all realms. Thus, the vocabulary domain name implies a different set of values depending on the Realm of use and the code system. The name is created using mixed case object oriented style naming, in English, without the use of white space or special punctuation. The name is generally singular. This name is used when the vocabulary domain is used in HL7 specifications or when the domain is referenced by other vocabulary domain definitions. Examples of acceptable names are: Race, AmericanIndian, AlaskaNative, MaritalStatus, Gender, OrderType, PatientType, and AbnormalFlag.
- **Realm** – the realm of use of the value set. Note that a given vocabulary domain will have a new row for each different realm of use and code system. The values for the realm column come from the RealmOfUse vocabulary domain.
- **Code System** – indicates the name of the code system that provides the actual values (concepts) that are contained in the value set. If the code system is *HL7*, then other value sets defined by reference to external vocabularies are not allowed. If the code system is not *HL7* and a given value set can be obtained from more than one vocabulary provider, then there will be a row in the Value Set Definition Table for each unique provider of that value set. If the code system is *HL7*, then the values in the domain can be obtained by examining the HL7 Value Set Relationship Table and the HL7 Source Vocabulary Representation Table. Descriptions of these tables are included below. If

the code system is other than *HL7*, the contents of the domain can be obtained by accessing tables or services provided by the terminology vendor. The values for this column come from the CodeSystem vocabulary domain.

- Value Set Description – a textual description of the value set as it is known within the code system. When possible, the principle upon which concepts are either included or excluded from the domain should be stated.
- Value Set Definition Expression – an expression that defines how the value set is derived from other pre-existing value sets. The expression refers to value sets using a name enclosed in double quotes. The name enclosed in quotes is a concatenation of the domain name, the realm, and the code system. When a value set expression is for the HL7 code system, the expression includes set operators that indicate how a given value set can be derived from pre-existing HL7 maintained value sets. The value sets referenced in the expression can be either primitive or composite. A composite value set is a value set that contains other value sets. The allowed set operators are:

Operator Symbol	Description
+	Union (\cup)
-	Difference (-)
*	Intersection (\cap)

Table 7-2. Set Operators

Parentheses are allowed in the expression when they are needed to create the proper ordering of the operations. If the value set definition is for any system other than HL7, then there must be a valid expression in the expression field that refers to a value set provided by the terminology source named in the code system column of this table. For non-HL7 vocabularies, operators other than the usual set operators are allowed. For example, *ChildrenOf* might be used as a keyword to indicate that all children of a given hierarchical node are included in the value set. It is the responsibility of the given terminology provider to define the operators, keywords, and syntax that are supported by their terminology system, and to state how hierarchical structures (nodes) should be referenced.

- Current Status – the status of the item. The values for Status come from vocabulary domain EditStatus. Some values for status are Proposed, Rejected, Active, Obsolete, and Inactive.
- Vin – the version number of the domain specification database at the time that this entry (row) was added or updated. *See the discussion of the version tracking table below for more details.*
- Vout – the version number of the domain specification database at the time that this entry was modified or deleted. A blank Vout value means that the row continues to exist in the current version of the table. *See the discussion of the version tracking table below for more details*
- Comment (*not shown in the example table*) – a general purpose textual field for recording specific information about the vocabulary domain, or details about the rationale for modifying a particular row in the table.

7.1.6.3 Version Tracking Table

Associated with the domain specification database is a version tracking table. The purpose of the version tracking table is to keep audit information on each editing session that causes additions, deletions, or updates to the domain specification database tables. When used in conjunction with the Vin and Vout fields of the tables, an exact image of the tables for any point in time can be reconstructed. The following discussion will describe the use of the version tracking table.

A row is added to the version tracking table as each new edit session on the domain specification database begins. Each row in the version tracking table contains a version number, the date and time when the edit

session took place, the person actually making the edits, which group or committee was responsible for the edit session (if that is different from the person making the edits), and a reason/comment/summary about what was changed and why. Each time a new row is added to the table, the version number is incremented by 1. Each time a new row is created in any of the domain specification tables, the version number from the version tracking table is entered into the Vin column of the row that was added to the table. Each time a row in any table of the domain specification database is deleted, the current version number is entered into the Vout column. When a row is updated in any table of the domain specification database the modified entry is copied to a new row in the table, the Vout column of the original row is set to the current version number, and the new row's Vin column is set to the current version number.

An entry in the vocabulary domain specification table is considered a part of the current version until its Vout field is set to a version number. So, all items that have a blank or null Vout value are members of the current version. Any items where Vout is set to an actual version number are no longer active in the current version. The exact set of entries that were members of the table at any point in time can be determined by looking for the date of interest in the version tracking table, and then selecting entries in the table that have a Vin value equal to or greater than the version number of the date selected and whose Vout value is less than the value of the selected version.

7.1.6.3.1 Structure of the Version Tracking Table

An example of the version tracking table is shown below.

Version	Date/Time of Edit	Who	For Whom	Comment
1	199904142200	1234 (Dan)	4359 (PAFM)	Created entries for all composite race vocabulary domains
2	199904161000	1234 (Dan)	922 (Vocab TC)	After testing and review, changed the status of the race domain to active.
3	199905181200	8765 (Jim)	2566 (HL7)	Release of Version 2.3.1 of the standard.

Table 7-3. Version Tracking Table for the Vocabulary Domain Specification Database

The version tracking table has the following columns:

- Version – the version number of the edit session. This number is incremented by 1 each time a new edit session takes place. The version number is used as the value of Vin and Vout as appropriate to track which table entries in the domain specification database were added, modified, or deleted during the session.
- Date/Time of Edit – the date and time that the edit session began.
- Who – an identifier of the person who actually edited the database. People are identified by reference to a directory where each person is assigned a unique identifier, and where locating information about the person can be found.
- For whom - an identifier of the person or organization for whom the edit was made. For example, a person may be making edits as authorized by the Vocabulary Technical Committee, or on behalf of the Technical Committee for which they are the facilitator. People and organizations are identified by reference to a directory where each person or organization is assigned a unique identifier.
- Comment – a summary of why the edits were made, and what was done.

7.1.6.4 The Value Set Relationship Table

The Value Set Relationship Table creates relationships between HL7 maintained value sets and their contents. The contents can be either another HL7 value set, or an individual concept. This table also tracks comments and status information on the relationships.

7.1.6.4.1 The Structure of the Value Set Relationship Table

An example of the Value Set Relationship Table is shown below.

HL7 Value Set Id	Value Set Name	Operator	Generality	HL7 Concept Id	HL7 Value Set/Concept Name	Status	Vin	Vout
50005	Race:USA:HL7	Include	Abstract	20001	AmericanIndianOrAlaskaNative:USA:HL7	A	1	
50005	Race:USA:HL7	Include	Specializable	20002	Asian:USA:HL7	A	1	
50005	Race:USA:HL7	Include	Abstract	20003	BlackOrAfricanAmerican:USA:HL7	A	1	
50005	Race:USA:HL7	Include	Abstract	20004	NativeHawaiianOrPacificIslander:USA:HL7	A	1	
50005	Race:USA:HL7	Include	Specializable	20005	WhiteRace:USA:HL7	A	1	
20001	AmericanIndianOrAlaskaNative:USA:HL7	Include	Specializable	30001	AmericanIndian:USA:HL7	A	1	
20001	AmericanIndianOrAlaskaNative:USA:HL7	Include	Specializable	30002	AlaskaNative:USA:HL7	A	1	
1	Gender:Root:HL7	Include	Leaf	40001	Male	A	1	
1	Gender:Root:HL7	Include	Leaf	40002	Female	A	1	
1	Gender:Root:HL7	Include	Leaf	40003	Other	A	1	
1	Gender:Root:HL7	Include	Leaf	40004	Transsexual	A	1	
1	Gender:Root:HL7	Include	Leaf	40005	Unknown	A	1	

Table 7-4. Value Set Relationship Table

The Value Set Relationship Table has the following columns:

- HL7 Value Set Identifier – a unique, sequential number assigned by HL7 that identifies a value set. This number is assigned when a definition for the value set is added to the value set definition table. For HL7 tables that exist in Version 2.X, the value set identifier is the same as the Version 2.X table number. The value set identifier has the same meaning across all tables in the domain specification database and is used as a foreign key to allow joins between the tables.
- HL7 Value Set Name – (*Note: this column is shown for purposes of illustration only. The value set name column only exists in the Value Set Definition Table.*) A unique textual name assigned by HL7 for the value set. The value set name implies a different set of concepts within each realm of use and code system. The name is generally singular. This name is used when the value set is used in HL7 specifications or when the value set is referenced by other vocabulary domain definitions.
- Operator - the name of the relationship that exists between the value set and the concept that is listed in the HL7 Concept Id column. The most common relationship is Include, which means that the domain contains the concept listed in the HL7 Concept Id column. Relationship names come from the ValueSetOperator domain.
- Generality - indicates whether the concept in the HL7 Concept Id column should be interpreted as itself, or whether it should be expanded to include its child concepts, or both. Possible values for this column are *Abstract*, *Specializable*, and *Leaf*. *Leaf* means that only the concept itself is included in the domain. *Abstract* means that only descendants of the concept are included in the domain, and *Specializable* means that the concept itself and its descendants are included in the domain. The values for the Generality column come from the ConceptGenerality domain.
- HL7 Concept Id - the concept identifier of the item that is being included in the value set. The item can be a value set or a terminal (leaf) concept. If the concept is a value set, its definition can be found by reference to the Value Set Definition Table. If the concept is a terminal concept, its meaning can be found by reference to the Source Vocabulary Representation Table.

- HL7 Value Set/Concept Name – (*Note: this column is shown for purposes of illustration only. The domain name column only exists in the Value Set Definition Table.*) A unique textual name assigned by HL7 for the value set.
- Status – the status of the item. The values for Status come from the vocabulary domain EditStatus. Some values for status are Proposed, Rejected, Active, Obsolete, and Inactive.
- Vin – the version number of the domain specification database at the time that this entry was added or updated. *See also the discussion of version tracking table included in the description of the Value Set Definition Table.*
- Vout – the version number of the domain specification database at the time this entry was updated or deleted. A blank Vout value means that the row continues to exist in the current version of the table. *See also the discussion of version tracking table included in the description of the Value Set Definition Table.*
- Comment (*not shown in the example table*) – a general purpose textual field for recording specific information about the value set, or details about the rationale for modifying this particular table entry.

7.1.6.5 The Source Vocabulary Representation Table

This table contains the actual representations (codes and text translations) of concepts that are contained in the Value Set Relationship Table. Though this table is maintained by HL7, it may contain concepts from other vocabulary sources that agree to have their concepts reside in the HL7 domain specification database. The meanings of concepts used in HL7 maintained value sets are defined by the contents of the Source Vocabulary Representation Table, particularly the Description column. That is, the meaning of a given concept identified by an HL7 concept identifier is defined by the row in this table that has the given identifier as the value in the HL7 Concept ID column and has *HL7* as the value of the Code System column.

7.1.6.5.1 The Structure of the Source Vocabulary Representation Table

An example of the Source Vocabulary Representation Table is shown below.

HL7 Value Set Id	HL7 Concept ID	Map Rel	Code Sys	Source Value Set Name	Table ID	Code Sys Vin	Code Sys Vout	Code	ISO Lang	Description	HL7 Status	HL7 Vin	HL7 Vout
1	400001	E	CR	Sex	0220	6		1	en	Male	A	1	
1	400002	E	CR	Sex	0220	6		2	en	Female	A	1	
1	400003	BT	CR	Sex	0220	6		3	en	Other (Hermaphrodite)	A	1	
1	400004	E	CR	Sex	0220	6		4	en	Transsexual	A	1	
1	400005	BT	CR	Sex	0220	6		9	en	Not Stated/Unknown	A	1	
1	400001	E	HL7	Gender	0001	2.3.1		M	en	Male	A	1	
1	400002	E	HL7	Gender	0001	2.3.1		F	en	Female	A	1	
1	400003	E	HL7	Gender	0001	2.3.1		O	en	Other	A	1	
1	400004	E	HL7	Gender	0001	2.3.1		T	en	Transsexual	I	1	
1	400005	E	HL7	Gender	0001	2.3.1		U	en	Unknown	A	1	

Table 7-5. Source Vocabulary Representation Table

The Source Vocabulary Representation Table has the following columns:

- HL7 Value Set Identifier – a unique, sequential number assigned by HL7 that identifies a value set. For HL7 tables that exist in Version 2.X, the value set identifier is the same as the Version 2.X table number. (*Note: this column is shown for purposes of illustration only. The relationship between a value set and an individual concept is defined in the HL7 Value Set Relationship Table. The example shown above could be created by a join between the Value Set Relationship Table and the Source Vocabulary Representation Table.*)
- HL7 Concept Identifier – the unique item identifier assigned by HL7 to this concept. This concept identifier is globally unique to a concept throughout all HL7 tables, and it does not overlap any HL7 value set identifier. That is, if the concept *male* occurred in another vocabulary domain in addition to the Gender domain, it would again have an item identifier of “40001”. If a universal terminology of medicine becomes available, the universal concept identifier from that terminology will be used in place of this HL7 assigned identifier.
- Map Relationship - the closeness or quality of the mapping between the HL7 concept (as represented by the HL7 concept identifier) and the source coding system. The values for the relationship come from the MapRelationship domain and are patterned after the similar relationships used in the UMLS Metathesaurus. Examples of values for map relationship are: exact, broader than, narrower than, etc. Because the HL7 coding system is the master reference for the definition of the concept, the map relationship for HL7 coding system entries will always be Exact.
- Code System – the identifier of the code system from which this item was obtained. This is not a free text field. The values for this column come from the CodeSystem vocabulary domain.
- Source Value Set Name - the name of the value set in the original source which contains this concept.
- Table Identifier - the table number or identifier (if one exists) in the source vocabulary where this concept originated.
- Code System Vin – the version number of the code system at the time that the code was initially created.
- Code System Vout – the version number of the code system at the time the code was retired or deleted.
- Code – the text string used within the code system to identify this concept.
- ISO Language - the language (English, German, French, Italian, etc.) that is used in the description. The language codes come from the vocabulary domain of *Language* and are specified by ISO 639:1988 (E/F) *Code for the representation of names of languages*.⁴
- Description – a textual representation of the meaning of this entry as it is represented in the code system from which it originated.
- HL7 Status – the status of this entry within this table. The values for Status come from the vocabulary domain EditStatus. Some values for status are Proposed, Rejected, Active, Obsolete, and Inactive.
- HL7 Vin – the version number of the domain specification database at the time this entry was added or updated. *See also the discussion of version tracking table included in the description of the Value Set Definition Table.*

- HL7 Vout – the version number of the domain specification database at the time this entry was modified or deleted. A blank Vout value means that the row continues to exist in the current version of the table. *See also the discussion of version tracking table included in the description of the Value Set Definition Table.*
- Comment (not shown in the example table) – a general purpose textual field for recording specific information about this code, or details about the rationale for creating, modifying, or deleting this particular table entry.

7.1.6.6 The Observation Identifier to Value Set Linking Table

Some parts of HL7 messages represent their content using name-value or attribute-value pairs. For example, clinical assessments or laboratory observations are often represented in messages using a pair of data elements: an observation identifier (LOINC code) and an associated value. The LOINC coding system often contains suggestions for the possible values of a coded LOINC observation, but these are only suggestions and are not meant to be comprehensive or normative. The Observation Identifier to Value Set Linking Table, in conjunction with the other domain specification tables, will allow LOINC domains or other LOINC like observation identifiers to be recorded in a formal way, and will allow the links between observation identifiers and domains to be normative in messages as desired.

7.1.6.7 The Structure of the Observation Identifier to Value Set Linking Table

An example of the Observation Identifier to Value Set Linking Table is shown below.

Code Sys	Code Sys Vin	Code Sys Vout	Observation Identifier	Value Set Id	Value Set Name	HL7 Status	HL7 Vin	HL7 Vout
LN	1.0L		11882-8 (Fetal Gender)	1	Gender	A	1	

Table 7-6. Observation Identifier to Value Set Linking Table

The Observation Identifier to Value Set Linking Table has the following columns:

- Code System – the identifier of the code system from which the observation identifier was obtained. This is not a free text field. The values for this column come from the CodeSystem vocabulary domain.
- Code System Vin – the version number of the code system at the time that the observation identifier was initially created.
- Code System Vout – the version number of the code system at the time the observation identifier was retired or deleted.
- Observation Identifier – the observation identifier to which a vocabulary domain is being linked. Currently, the only observation identifiers that have been registered for use in HL7 messages are LOINC codes. When the observation identifier that is being linked is a LOINC code, the item should have a precision of *Nominal*. A given LOINC code should be linked to only one vocabulary domain.
- Code System Version In – the version number of code system from which the observation identifier was selected.
- Value Set Identifier – the unique numeric value set identifier for the value set being linked to the observation identifier.

- HL7 Value Set Name – (*Note: this column is shown for purposes of illustration only. The value set name column only exists in the Value Set Definition Table.*) A unique textual name assigned by HL7 for the value set.
- Status – the status of this entry within this table. The values for Status come from vocabulary domain EditStatus. Some values for status are Proposed, Rejected, Active, Obsolete, and Inactive.
- Vin – the version number of the domain specification database at the time this entry was added or updated. *See also the discussion of version tracking table included in the description of the Value Set Definition Table.*
- Vout – the version number of the domain specification database at the time this entry was modified or deleted. A blank Vout value means that the row continues to exist in the current version of the table. *See also the discussion of version tracking table included in the description of the Value Set Definition Table.*
- Comment (*not shown in the example table*) – a general purpose textual field for recording specific information about the vocabulary domain, or details about the rationale for modifying this particular table entry.

7.1.7 Use of the vocabulary domain specification database

Given the structure of the vocabulary domain definition database, the following steps can be used to determine the set of concepts and codes that are valid values for a given vocabulary domain or domain constraint.

- Look up the domain in the Value Set Definition Table using the domain name, the realm of use, and the code system.
- If the value set is from any source other than HL7, resolve the domain by calling a vocabulary server or service using the domain expression, the realm, code system, and code system version as parameters to the call.
- If the domain is an HL7 domain:
 1. Extract all the value set names from the expression column, and resolve the value set names to value set identifiers by reference to the Value Set Definition Table. A given value set can be found using a combination of the realm of use and the domain name. Use the value set identifier to resolve the domain by looking it up in the Value Set Relationship Table. The value sets may be either primitive or composite. Value sets are resolved by recursively looking up the value sets in the value set relationship table until no further rows with Include or Exclude operators for the value set can be found.
 2. When all value sets have been expanded to their elements, apply the set operations that were specified in the value set definition table to arrive at the complete set of enumerated values for the value set.
 3. If desired, the concepts from the Value Set Relationship Table can be resolved to representations in a particular code system by reference to the vocabulary source representation table. Each concept can be found by using the concept identifier and the code system. If the code that is specific to a particular version of the source is desired, the Vin and Vout columns can be used to find the exact code for a specific version of the source vocabulary.

7.1.8 Summary of vocabulary domains used in the specification of vocabulary domains (meta domains)

The following table summarizes the vocabulary domains that are themselves used in domain specifications or in the vocabulary domain specification database. All of these domains have an Extensibility qualifier of CNE, coded no exceptions.

Domain Name	Description	Sample Values
CodeSystem	The set of registered or known code systems. This table will supercede the current table in Chapter 7 of the HL7 standard.	SNM3 (SNOMED International), LN (LOINC), MEDCIN, ICD-9
ConceptGenerality	Indicates whether a concept itself, or its descendents, or both, should be included in a domain.	Abstract, Specializable, Leaf.
ValueSetOperator	Operations that can be used to associate concepts in a terminology.	Include, Exclude
EditStatus	The status of a domain entry as pertains to its review and incorporation into the HL7 domain specification database.	Proposed, Active, Rejected, Obsolete, Inactive
Extensibility(Ext)	The extensibility of coding determines whether or not exceptions are allowed in the domain of a coded attribute	CWE – coded with exceptions, CNE – coded no exceptions
Language	The human language that is used in textual descriptions or communications. From ISO 639.	English (en), French (fr), German (de), Italian (it), Portuguese (pt), etc.
MapRelationship	The kind of relationship between a concept in the HL7 vocabulary and a concept in an external source vocabulary.	Exact, Broader Than, Narrower Than
RealmOfUse (Realm)	The jurisdiction or realm within which the domain will be used. A realm might be a country, a group of countries, a region of the world, or an organization.	Universal, USA, Europe
VocabularyDomainQualifier	The set of allowed domain qualifiers	RealmOfUse (Realm), Extensibility (Ext)

Table 7-7. Meta-Domains Used in the Domain Specification Database

7.2 The structure of coded elements in messages

All of the details of the structures for the Version 3 data types that relate to coded information are specified in the Version 3 data type discussion. However, it is useful in this chapter of the document to show some examples that help clarify the intended use of each of the coded data types. The coded data types are called *Real World Concept Types* in the data type discussion. There are four kinds of Real World Concept Types: Code Value (CV), Code Phrase (CDPH), Code Term (CDXT), Code Translation (CDXL), and Concept Descriptor (CD). Each type builds from the previous type(s). A Code Value has the following sub-parts:

- value (Character String, conditional)
- code system (OID - Object Identifier, required)
- code system version (Character String, optional)
- print name (Character String, optional)
- replacement (Character String, conditional)

There is a condition on how instances of a Code Value are constructed. The condition is that an instance of a code value will contain either a value or a replacement, but not both. In the examples shown below, an informal notation is used to illustrate how the parts of a coded data type should be instantiated. Parentheses are used to delimit composite elements of the instance such as SETs or SEQUENCES.

The following example shows a typical Code Value that could be used as the value of a Patient Blood Type field in an HL7 message. In this example, the code value is coming from an external vocabulary, SNOMED International Version 3.4. These examples use fictitious OIDs (object identifiers) for the code system. The real OIDs would contain numbers, not text. These examples also assume that code systems would be assigned a unique number space under HL7's root. However, if code systems obtain their own OID root in the future, that OID would be used instead.

```
Patient Blood Type
Code Value (
    :value "F-D1111",
    :code-system "HL7.SMI" -- SMI is defined as a value in the CodeSystem domain
    :code-system-version "3.4"
    :print-name "Blood group A" )
```

The next example shows a Code Value where the code value comes from an HL7 maintained table.

```
Patient Sex
Code Value (
    :value "M",
    :code-system "HL7.HL7Codes.1"-- HL7 is defined as a value in the CodeSystem domain
    :code-system-version "3.0"
    :print-name "Male" )
```

This example shows the proper use of a Code Value when the code value comes from another group, in this case from the DEEDS implementation guide.

```
Tetanus Special Circumstances
Code Value (
    :value "3",
    :code-system "HL7.DEEDS.4.31"-- DEEDS is defined in the CodeSystem domain
```

```
:code-system-version "1.0"  
:print-name "only the year is known" )
```

This example shows how a Code Value should be created if the domain has an Extensibility qualifier of CWE (coded with exceptions), and the desired value is not in the standard domain. Note that the value field is not present, but the code-system and version are present. The meaning that the user wanted to express is entered as free text in the replacement slot.

```
Patient Sex  
Code Value (  
  :code-system "HL7.HL7Codes.1"  
  :code-system-version "3.0"  
  :replacement "Male Pseudo-hermaphrodite" )
```

As previously noted, no code value should be provided if the desired value is not a part of the standard domain. For instance, if male pseudo-hermaphrodite was not in HL7 table 0001, and local codes were ***not*** being used, it would be incorrect to make this instance of the patient sex field.

Incorrect Use of Code Value – both *value* and *replacement* are present at the same time

```
Patient Sex  
Code Value (  
  :value "MPH"  
  :code-system "HL7.HL7Codes.1"  
  :code-system-version "3.0"  
  :replacement "Male Pseudo-hermaphrodite" )
```

If male pseudo-hermaphrodite was not in HL7 table 0001, and local codes from Acme Healthcare (AH) ***were*** being used, and the local code for male pseudo-hermaphrodite was "MPH, then the correct representation would be as follows:

```
Patient Sex  
Code Value (  
  :value "MPH"  
  :code-system "HL7.Affiliate.AH.ADT" -- AH is assigned by HL7 to Acme Healthcare  
  :code-system-version "3.0"  
  :print-name "Male Pseudo-hermaphrodite" )
```

The following example illustrates that code-system-version and print-name are not required.

```
Patient Blood Type  
Code Value (  
  :value "F-D1111",  
  :code-system "HL7.SMI" )
```

- The second kind of coded data type is Code Phrase (CDPH). A code phrase has the following sub-parts
- primary code (Code Term, Required)
- SET OF Role Relation (Optional)

Role Relation is further defined with the following sub-parts:

- name (Code Value, Required)

- inversion_ind (Boolean, Optional)
- value (Code Translation, Required)

Note that the primary code in Code Phrase is of type Code Term. Code Term (CDXT) is defined as a CHOICE between Code Value and Code Phrase. Thus, the definition of Code Term is:

- Code Value **OR** Code Phrase

Note that Code Term is a virtual type, and is only used as a building block for other coded types. Any instance of a Code Term will actually be either a Code Value or a Code Phrase. Also note that since Code Phrase contains *primary code* which is a Code Term, and Code Term is defined as either a Code Value or a Code Phrase, Code Phrase indirectly refers to itself, making Code Phrase a recursive definition.

The following example shows how a Code Phrase could be used to represent a body location description.

Body Location (Example 1)

```
Code Phrase (
:term (Code Term
:primary-code (Code Value
:code-value "Arm"
:code-system "HL7.Affiliate.AH.Clinical" -- AH denotes Acme Healthcare
:print-name "Arm")
:roleRelation (SET (Role Relation
:name (Code Value
:code-value "[has-laterality]"
:code-system "HL7.Affiliate.AH.ADT" )
:value (Code Term
:primary-code (Code Value
:code-value "R"
:code-system "HL7.Affiliate.AH.ADT"
:print-name "Right" ))))))
```

Note that this example uses a local coding system, and that the coding system is redundantly stated in each occurrence of code value. In the next example the redundant code system statements have been removed.

Code Phrase (example 2)

Body Location

```
Code Phrase (
:term (Code Term
:primary-code (Code Value
:code-value "Arm"
:code-system "HL7.Affiliate.AH.Clinical" -- AH denotes Acme Healthcare
:print-name "Arm")
:roleRelation (SET (Role Relation
:name (Code Value
:code-value "[has-laterality]")
:value (Code Term
:primary-code (Code Value
:code-value "R"
:print-name "Right" ))))))
```

Example 3 shows that, as before, the size of a code phrase can be reduced by not showing the print names.

```

Code Phrase (example 3)
Body Location
Code Phrase (
:term (Code Term
:primary-code (Code Value
:code-value "Arm"
:code-system "HL7.Affiliate.AH.Clinical" -- AH denotes Acme Healthcare
:roleRelation (SET (Role Relation
:name (Code Value
:code-value "[has-laterality]")
:value (Code Term
:primary-code (Code Value
:code-value "R" ))))))

```

The final example of code phrase shows that replacement text can be used in any of the code value parts of a code phrase. In this example, *Both* have been used as replacement text.

```

Code Phrase (example 4)
Body Location
Code Phrase (
:term (Code Term
:primary-code (Code Value
:code-value "Arm"
:code-system "HL7.Affiliate.AH.Clinical" -- AH denotes Acme Healthcare
:print-name "Arm")
:roleRelation (SET (Role Relation
:name (Code Value
:code-value "[has-laterality]")
:value (Code Term
:primary-code (Code Value
:replacement "Both" ))))))

```

The fourth type of coded data type is Code Translation (CDXL). It has the subparts listed below. It is used as a component of Concept Descriptor. Examples of Code Translation will be shown in conjunction with examples of Concept Descriptor.

- term (Code Term, required)
- origin (reference to Code Translation, required)
- producer (TII, optional)
- quality (floating point number, optional)
- label (Character String, optional)

The final kind of coded data type Concept Descriptor. It is a set of Code Translation, plus original text. The structure is represented as follows:

- Original Text (Free Text, optional)
- SET OF Code Translation (required)

Examples of code descriptor will be shown by creating a progressively more complex statement about a trivial (but interesting) clinical description of a patient's hair color. In the first example, hair color is simply

represented as a code value from a locally defined coding scheme. Remember, for local codes to be legal, the domain for hair color must have been defined with a qualifier of CWE.

```
Hair Color
Code Value (
  :value "AB"
  :code-system "HL7.Affiliate.AH.Clinical" -- AH denotes Acme Healthcare
  :print-name "ash blond" )
```

Let's assume now, that we represent the same information as a Concept Descriptor. Note that there is no new information in this example, that is, no translation has been added yet. All that has been done is that the previous example has been moved into a Concept Descriptor. The *origin* field is used to indicate the translation from which this translation was derived. Since there was no translation, origin is set to null.

```
Hair Color
Concept Descriptor (
  :set of translations (
    :translation (
      :term (
        :code-value (
          :value "AB"
          :code-system "HL7.Affiliate.AH.Clinical"
          :print-name "ash blond" )
        :origin #null ) ) )
```

In the next example, original text has been added (shown in shaded box). This structure indicates that the original statement about the patient's hair color was collected as free text (ash blonde), and that the text was then encoded by a human or a machine as the local code AB which stands for ash blond. When taken in aggregate, this concept descriptor captures the original text and a first encoding of the hair color field.

```
Hair Color
Concept Descriptor (
  :original text "ash blonde"
  :set of translations (
    :translation (
      :term (
        :code-value (
          :value "AB"
          :code-system "HL7.Affiliate.AH.Clinical"
          :print-name "ash blond" )
        :origin #null
        :quality '1.0' ) ) )
```

Now suppose that the information from the previous example was translated to a new code system, in this case, the (made up) ICHC code system. Now there are two representations of the original text that have been created. The origin field (with a value of "1") in the second translation shows that this translation was created from the translation labeled 1. In other words, the ICHC translation was created by reference to the local AH.Clinical code system code of AB.

Hair Color

Concept Descriptor (

:original text "ash blonde"

:set of translations (

```
:translation (  
  :label "1"  
  :term (  
    :code-value (  
      :value "AB"  
      :code-system "HL7.Affiliate.AH.Clinical"  
      :print-name "ash blonde" ) )  
  :origin #null )
```

```
:translation (  
  :term (  
    :code-value (  
      :value "10.2"  
      :code-system "HL7.Affiliate.ICHC"  
      :print-name "blond, pale" ) )  
  :origin "1"  
  :quality '.85'  
  :producer (TII for Interface Engine) ) ) )
```

The final example shows yet one more translation. This final translation uses the PILS-AVACC code system. Based on the value of origin in the third translation, we can see that the third translation is based on the second translation. This would be an extremely bad practice for real data, but it illustrates that the structure can accurately reflect the ancestry of each translation. The other significant factor is that PILS-AVACC is a multi-axial code system that is more atomic in nature than either the local AH.Clinical code system or the ICHC code system. In the PILS-AVACC code system, ash blonde is represented as two codes, not just one. Thus, the structure can accommodate complex mappings between vocabularies.

Hair Color
 Concept Descriptor (
 :original text "ash blonde"
 :set of translations (

<pre> :translation (:label "1" :term (:code-value (:value "AB" :code-system "HL7.Affiliate.AH.Clinical" :print-name "ash blonde")) :origin #null) </pre>
<pre> :translation (:label "2" :term (:code-value (:value "10.2" :code-system "HL7.Affiliate.ICHC" :print-name "blond, pale")) :origin "1" :quality '.85' :producer (TII for Interface Engine)) </pre>
<pre> :translation (:term (:code-value (:value "B001" :code-system "HL7.Affiliate.PILS-AVACC" :print-name "blond") :code-value (:value "G002" :code-system "HL7.Affiliate.PILS-AVACC" :print-name "slight gray") :code-value (:value "H003" :code-system "HL7.Affiliate.PILS-AVACC" :print-name "homogeneous")) :origin "2" :quality '.60' :producer (TII for Receiving System))) </pre>

7.3 The general process of maintaining domain specifications

- Specifying vocabulary domains, and vocabulary domain harmonization will be patterned after the RIM harmonization process. Vocabulary harmonization will happen in conjunction with RIM harmonization meetings.
- Vocabulary Technical Committee members will be appointed as vocabulary facilitators to the message development Technical Committee's by the co-chairs of the vocabulary Technical Committee. The facilitators must also be approved by the co-chairs of the committees for whom they are providing facilitation. At least two vocabulary facilitators will be assigned to each message development Technical Committee. People with a special interest may also be appointed to be the stewards of specific tables.

- For shared vocabulary domains, a single Technical Committee will be chosen as steward, with all interested parties have an opportunity to provide input, following the harmonization rules adopted for shared RIM objects.
- The message development Technical Committees have the ultimate responsibility to define the vocabulary domains referenced by the RIM objects for which they are the steward.
- The vocabulary facilitators have the responsibility to see that good vocabulary practices are followed, and to physically maintain the vocabulary domain specification database according to the definitions provided by the Technical Committees.
- Any new concepts created by HL7 Technical Committees will be submitted to HL7 registered vocabulary/terminology developers for potential inclusion in the standard terminologies. See below for more details on how vocabularies are registered for use in HL7 messages.
- The Vocabulary Technical Committee will maintain a document that contains a description of the good vocabulary practices to be followed by HL7 Technical Committees for definition and maintenance of vocabulary domains. The guidelines in the document will be approved by the Architectural Review Board and by the Technical Steering Committee.
- All HL7 registered vocabulary/terminology providers are welcome to provide mappings to HL7 domains using their terms and codes. The vocabulary providers are responsible for mapping their concepts to the concepts in the domain. Any proprietary vocabularies used in HL7 specifications or interfaces must be properly licensed according to the licensing policies of the vocabulary provider.

7.3.1.1 Detailed Process for Physically Updating the Domain Tables

- The vocabulary domain specification database will be available from the HL7 Web Site.
- Any HL7 member will be able to access the web page via login and password and review the HL7 tables.
- An *Edit Permissions* table (structure not shown) will be maintained as part of the vocabulary domain specification database. It contains the list of vocabulary facilitators who have been appointed to modify the contents of each vocabulary domain. The designated participants must identify themselves by login and password before editing of the vocabulary domain specification database is allowed. This table should also contain contact information for the facilitators so that people know where to direct their questions or input.
- The co-chairs of the Vocabulary Technical Committee (or persons designated by them) will maintain the Edit Permissions table.
- The person or persons that have been assigned to maintain a particular vocabulary domain will be able to access the domain and make additions, modifications, or deletions to the values of the domain. All changes will first be entered with a status of proposed.
- A vocabulary review committee appointed by the co-chairs of the Vocabulary Technical Committee will review all proposed changes to domains for consistency with good vocabulary practices prior to the changes being submitted for RIM harmonization.
- The co-chairs of the Vocabulary Technical Committee have the responsibility of preparing reports and summaries of changes and additions to domains to be reviewed during vocabulary harmonization meetings. At least one Vocabulary Technical Committee co-chair must participate in the vocabulary harmonization process.

- Based on the conclusions of the RIM harmonization process, proposed changes are either accepted or rejected. The accepted domain values will be marked as active in the domain specification database, and rejected items will be marked as rejected in the database.
- The vocabulary review committee will collaborate with all HL7 registered vocabulary/terminology developers to insure that any new concepts generated by HL7 Technical Committees are submitted for inclusion in the standard terminology's.
- Each time a new version of an HL7 standard is released, the version numbers from the version tracking tables in the domain specification database will be recorded. In this way, the version of the domain specifications that correspond to the given standard release will be a known and fixed set.
- Based on requirements set by the HL7 Board of Directors, HL7 members will be able to download copies of the HL7 domain specification database from the web site. This could either be a complete copy of a particular version (date), or only updates starting at a particular version.

7.4 Good vocabulary practices

The vocabulary Technical Committee has the responsibility to maintain a set of good vocabulary practices used in reviewing domain contents. This document will be developed separately from the 1999 version of the MDF. Specific practices have yet to be approved, but at least one source for such rules exists in the literature.

7.5 Use of external terminology's in HL7 standards

The Vocabulary Technical Committee has been working on strategies for how to use external vocabularies in HL7 domain specifications and messages for over two years. The following principles summarize HL7's general approach:

- HL7 is committed to using existing terminology's, where possible, as values for coded fields in HL7 messages, rather than creating a new terminology.
- HL7 is seeking a solution that allows the use of proprietary vocabularies (SNOMED, Read, MEDCIN, etc.) in a manner that is equitable to all vocabulary creation/maintenance organizations.

Given these goals, the HL7 Vocabulary Technical Committee has concluded that:

- HL7 should **not** choose a single proprietary coding scheme.
- The presumption is that a "market model" will assure responsive maintenance of proprietary terminology's.
- HL7 will properly license any terminologies it uses.
- The primary code in coded fields can be a UMLS CUI (Unified Medical Language System, Concept Unique Identifier) or a proprietary code when properly licensed.
- Coding schemes used by HL7 must be registered with HL7. *See more details below.*
- To be used for a given domain, a terminology must cover all of the concepts defined in the domain specification for that domain.

- To enable interoperability, proprietary codes used in messages must be mapped to a publicly available reference model that supports the desired degree of interoperability.

7.5.1 Process for registering vocabularies for use in HL7

The goal of registration is to make available to HL7 implementers a list of vocabularies that can be used in HL7 messages. The goal is that registration will not subject HL7 to liability regarding antitrust, or other types of liability.

7.5.1.1 Steps of the Registration Process

1. Terminology Systems will self-register.
2. Registration of a terminology system will require *Registration Documentation* to describe how the system is to be used in HL7. Registration Documentation will contain:
 - An itemized description of how the Terminology System meets the “Principles for HL7-compliant terminology’s”
 - An initial list of domain specifications for HL7 v 2.X and v 3.X in tabular form suitable for upload into the domain specification database as described above.
3. A Terminology System Sponsor (acceptable to the Terminology Development Organization being sponsored) will identify themselves to the HL7 Terminology Technical Committee and will commit to the creation and timely update of the HL7 terminology registration documentation.
4. The sponsor will complete the *Registration Documentation* and make the documentation available to HL7 for posting on the HL7 web site.
5. Any HL7 member may ask clarifying questions regarding the principles and the domain constraints proposed in the Registration Documentation. These questions alone or the questions and their subsequent answers will become a permanent part of the registration documents. Questions deemed irrelevant by a majority of the Technical Committee Co-Chairs may be excluded from or edited before inclusion into the Registration Documentation.
6. Completed registration documents are submitted during a regular HL7 meeting. At the next HL7 meeting, allowing for clarifying questions and incorporation of the answers of those questions into the registration documentation, the terminology system shall be considered registered.

7.5.1.2 Obligations upon registration

- Advertisements that a system is HL7 registered must include a prominent disclaimer stating that “HL7 registration does not imply endorsement by the HL7 organization, nor does it imply that a system is necessarily appropriate for use in HL7 messages.”
- Keep the document up to date on a quarterly basis, and move toward comprehensive specification of domain values.

7.5.1.3 Principles for Adoption of HL7-Compliant Terminology’s

The Vocabulary Technical Committee has been working on guidelines and practices for registering vocabularies/terminologies for use in HL7 domain specifications and messages. While the guidelines are stated using words such as “will” and “must” and “shall”, in reality any vocabulary can be registered for use in HL7 messages. However, it is the opinion of the HL7 Vocabulary Technical Committee that terminologies that adhere to the guidelines are more likely to be used. Ultimately, it is up to the

implementers and users of systems to determine which terminology's provide the best cost/benefit ratio. Part of the registration process is a document describing how the given terminology adheres to these principles.

1. The terminology must be compliant with the semantics of the HL7 structures.
2. The terminology provider must be willing to participate in HL7-sanctioned interoperability efforts.
3. There must be an organization committed to the timely maintenance and update of the terminology.
4. Terminology license fees should be proportional to the value they provide to the enterprise and the end user, yet should not be out of proportion with the development and support costs of the terminology itself.
5. The terminology should be comprehensive for the intended domain of use.
6. Regulatory circumstances may require the adoption of specific terminology's. The use of such terminology's, provided that they are consistent with principle #1 above, shall not be considered a deviation from the HL7 standard.

7.5.1.4 Principles for HL7-Sanctioned Terminology Integration Efforts

In addition to the guidelines stated above, the Vocabulary Technical Committee has approved the following principles that should be followed by HL7 and terminology providers when they collaborate to integrate terminology use in HL7 messages and domain specifications. Again, the principles are stated with "must" and "shall" language, but adherence to these principles is strictly voluntary by all parties.

- A source provider must be willing to publish a limited conceptual model sufficient for HL7 implementation, and allow HL7 implementers to implement those or derivative information models within their applications without royalty, and the HL7 organization is free to incorporate those or derivative information models within the HL7 standard.
- Within the limitations stated by an appropriate written agreement, a terminology will be provided by "The Terminology Provider" to any HL7 sanctioned "Terminology Integration Organization" (TIO) efforts that wish to integrate said terminology content with the terminology content of others.
- For the purposes of an integration effort
 1. The TIO is required to implement the terminology provider's system in a way that is approved of by the source provider.
 2. The TIO must distribute and license its integrated products in accordance with its agreement with the terminology provider. Participation with HL7-sanctioned terminology integration efforts does not imply that the terminology provider must relinquish any rights to its intellectual property.
 3. The TIO should enter into a written agreement with the terminology provider that details the assumption of liability pertaining to the accuracy of any resulting interoperability products.

7.5.1.5 Necessary Information for Registered Vocabularies

A certain set of information must be known in order for a vocabulary to be referenced correctly in the domain specification database. This information will be kept in a document available on the HL7 Web Site. For each registered vocabulary it will be noted:

- How versions are named for that vocabulary
- How concepts are referenced in the vocabulary (i.e., which code or identifier is to be used in when the term is used in a domain specification)
- What hierarchies or named subsets are available to reference in vocabulary domain specifications, and how the subset or hierarchical nodes are to be referenced. Because of differences in vocabularies, how subsets are referenced may be different for each registered vocabulary.
- The terminology provider must provide the semantics and syntax for expressions that are created in the *Expression* column of the value set definition table.

7.6 The use of Local Vocabularies in Coded Elements

It is legal to use locally defined concepts and codes in HL7 messages when the standard value set does not contain the needed concept, and when the domain has been qualified with an Extensibility of CWE (coded with exceptions). When local codes are used in conjunction with the standard domain, the complete domain for the field is created by joining the local codes with the standard domain using a union operation.

There are a few rules that govern the use of local codes:

- A local code can not be sent for a concept that is already represented in the standard domain.
- Locally defined code systems will be assigned OIDs (object identifiers) so that they can be distinguished from any other HL7 registered proprietary or external terminology.
- Local codes should be submitted to the HL7 Vocabulary Technical Committee so that they can be incorporated into the standard domain, thereby increasing the interoperability of the communicating systems.

7.7 HL7 Vocabulary and the UMLS Metathesaurus

HL7 has an informal agreement with the United States National Library of Medicine (NLM), that HL7 maintained vocabulary tables will be incorporated into the UMLS Metathesaurus. A previous article⁵ describes the details of the approach used to add HL7 table information into the Metathesaurus. In fact, many of the characteristics of the HL7 domain specification database are based on the design principles of the Metathesaurus tables.

The main reasons for wanting the HL7 tables in the Metathesaurus are:

- It is a neutral environment where cross mapping of various terminologies can occur.
- The Metathesaurus has proved to be a stable and readily accessible archive of concepts and their representations.
- Linking HL7 content to the Metathesaurus content allows HL7 users to take advantage of the richness of the UMLS Knowledge Sources, including links to bibliographic references and the semantic network.

The idea of creating a common repository for vocabulary used by the message standards groups is not new. Dean Bidgood proposed the creation of “a generic message/terminology mapping resource based on the SNOMED (Systematized Nomenclature of Medicine) DICOM Microglossary (SDM) model -- the *Terminology Resource for Message Standards* (TeRMS).”⁶ The SDM is the current vocabulary resource

used in the DICOM standard. The TeRMS database would incorporate many of the features of the SDM, but would accommodate vocabulary content from multiple message standards. The plan was to build the TeRMS database as an extension of the UMLS Metathesaurus. This shared resource would provide the basis for a common understanding of coded-entry concepts as they were used in each of the individual message standards. The vocabulary of any given standard would be a subset of the proposed TeRMS message/terminology mapping resource.

Placing HL7 terms and codes in the Metathesaurus will allow easy cross-referencing between HL7 terms and existing vocabularies, like SNOMED International and LOINC. It also creates linkages to other aspects of the UMLS Knowledge Sources such as bibliographic references, the Semantic Network, the Specialist Lexicon, definitions, and co-occurrence data. These linkages have the potential to make data in HL7 messages much more useful to the parties that send and receive clinical messages. Furthermore, the goals of the TeRMS initiative will be realized if other SDO's also use the Metathesaurus as a repository for their codes. Having all of the codes in a common database will allow comparisons that can lead to greater consistency and interoperability among the various message developers.

While the TeRMS database is not yet a reality, some first steps in that direction have been taken. The 1999 version of the Metathesaurus contains the contents of 20 HL7 maintained vocabulary tables. The inclusion of these tables (domains) in the Metathesaurus was undertaken as an experiment to determine the magnitude of the work effort, and to gain a greater understanding of the process. If the experiment is successful and the content is useful, more of the HL7 vocabulary will be included in future editions of the Metathesaurus.

References

1. Organization IS. International Standard ISO 1087: Terminology--Vocabulary. Geneva, Switzerland: International Standards Organization; 1990.
2. CEN ENV 12264. Medical Informatics — Categorical structure of systems of concepts — Model for representation of semantics. . Brussels: CEN; 1995.
3. Object Management Group. Lexicon Query Service. TC Document CORBAmed 98-03-22. : Object Management Group; 1998.
4. ISO 639. Code for the representation of names of languages. . Geneva, Switzerland: International Standards Organization; 1988.
5. Huff SM, Bidgood WD, Cimino JJ, WE H. A Proposal for Incorporating Health Level Seven (HL7) Vocabulary in the UMLS Metathesaurus. Journal of American Medical Informatics Association, AMIA Annual Fall Symposium Supplement, 1998; 800-4. . 1998.
6. Bidgood WDJ. The SNOMED DICOM Microglossary: A Controlled Terminology Resource for DICOM Coded Entry Data Elements. In: Chute CG, ed. *IMIA Working Group 6*. Jacksonville, FL; 1997.

8. Interaction Model

8.1 Introduction

The Interaction model describes the parties that exchange HL7 messages and the interactions that flow between those parties. It provides concepts that define the messaging behavior required for particular functional roles. This supports a sophisticated view of the requirements for messaging, and makes it possible to structure Version 3 conformance claims.

The Interaction Model provides a place to determine and specify the information flows that the messages will be developed to support. It provides dynamic picture of the HL7 world as contrasted to the Use Case and Information Models. However, construction of this model draws on both the Use Case Model and on the Information Model for its basic material. This model also lays the groundwork for defining meaningful conformance claims for HL7 users. Here are some explanatory points to help in developing the model.

The Interaction Model provides a tool for specification of the information flows that are standardized with the definition of HL7 Version 3 messages. An interaction defines a specific instance for information exchange. It specifies the trigger event and preconditions a message and defines the responsibilities of the message receiver. Technical committees will work on the Interaction Model by following a series of steps. The following steps should be considered:

1. **Identifying interactions**

The committee will determine the immediate scope of its work. It may choose to define the interactions needed to support a particular use case or use cases. It may focus on the interactions needed to support the state transitions of a subject class. It may seek to support the requirements already managed by another form of information exchange such as a group of Version 2 messages.

2. **Defining interactions**

Once the scope has been chosen, the committee will define the interactions needed to support messaging requirements within that scope. This involves specifying the components of the interaction, e.g., trigger event as discussed below.

3. **Validating interactions**

Supporting robust conformance claims is a key goal of the Version 3 process, and it is important to ensure that interaction definitions support the definition of reasonable and appropriate conformance claims.¹

4. **Grouping interactions**

The HL7 repository has the ability to group interactions. This makes it possible to retrieve and manage sets of interactions that a committee feels are linked together. Normally, the grouping principle will reflect the scope set for a particular message development project. Interactions may also be grouped to support conformance claims.

The information in the Interaction Model is needed to guide the development of message content and structure. Each HL7 message specified within a Hierarchical Message Description² will support one or more interactions. This information is also needed for defining and structuring HL7 conformance claims.

¹ Conformance claims are discussed more fully in the body of this chapter, and in Chapter 9 – Conformance Claims and Organizational Practices

² See Chapter 10 – Creating Message Specifications – for a detailed discussion of how to construct messages.

8.1.1 Why build the Interaction Model?

The Interaction Model is a tool for discussing the messaging requirements within a defined functional scope. It provides a link between both the Use Case and Information models and the message definition process. Within the Interaction Model, message developers can discuss the trigger events that cause information to flow, the roles that participate in information exchange, the requirements that must be met for a valid trigger event, the responsibilities of message receivers. In a sense, this is where the rubber of model construction starts to meet the road of information exchange standardization.

It is possible to start message development by constructing a preliminary version of the Interaction Model. However, as the message developer will see below, the final version of the model cannot be created without reference to the HL7 Reference Information Model.

8.2 Elements of the Interaction Model

The diagram shows three of the four key components of an interaction. These are the trigger event, the application role, and the interaction. When you add the fourth, the interaction sequence, you have the essence of the Interaction Model. As can be seen, it presents a highly abstracted view of the communication between systems.

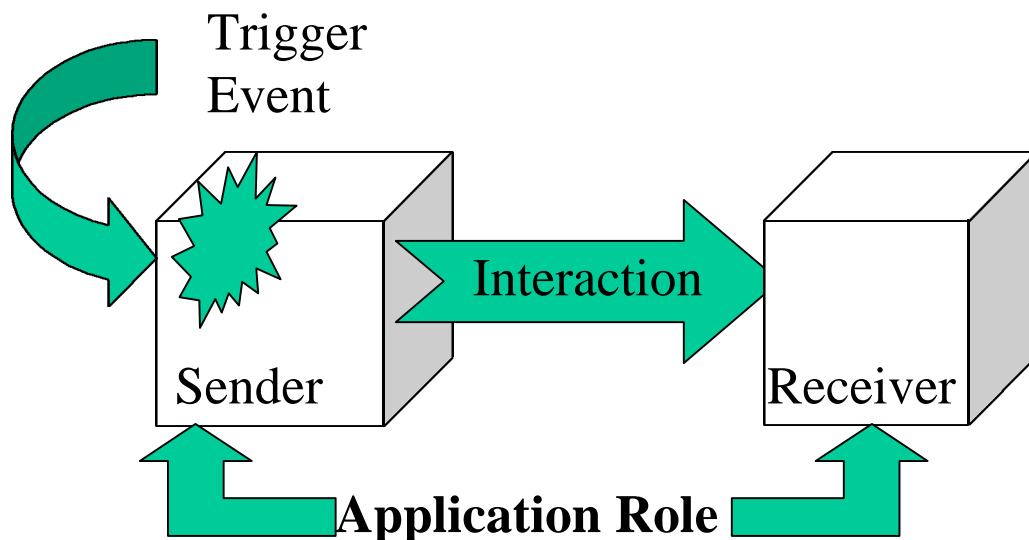


Figure 8-1. Each component is described below. This model is in contrast this to HL7 Version 2 which had trigger events and, implicitly, interactions; but no formal notion of senders and receivers.

8.2.1 Trigger Event

In a general sense, a trigger event is an occurrence in the healthcare domain or within the applications/information systems that support this domain that causes information to be exchanged. This exchange can be between systems, or it can take place within the healthcare domain without reference to system boundaries. For purposes, of HL7 analysis, it is assumed that information is to be exchanged between systems, since otherwise there is no need for a standardized set of HL7 messages to model that information exchange.

Each state transition defined for a subject class represents a perturbation of the class - a change in one or more attribute values. These changes are not spontaneous but are caused by something, and that something is a candidate for definition as a trigger event. Therefore, inspection of the subject class state models within the Information Model provides a source for discovering trigger events. Defining trigger events is a

key step within the HL7 process. Trigger events define the circumstances under which messages are sent. They are just as critical for the HL7 specification as the contents of the messages themselves. Trigger events, with a significant class of exceptions, are discovered by working with the state model for a subject class.³ Each trigger event is tied to at least one state transition for one of the subject classes in the model. In turn, the state transition traces to a leaf-level use case from which the transition stems, and the leaf-level use case defines the context for the trigger event.

Queries represent the significant class of exceptions, and the trigger event for a query is something far more general than a specific trigger event. It is based on the sender's "need to know" about a subject class and the data related to it. Defining the specification of Version 3 queries is still being worked on.

Note, examination of use cases in the Use Case model may lead to identification of potential trigger events, of cases in which data must be exchanged to achieve the objective of the use case. If the Information Model has been properly developed, these potential trigger events will have been also identified as state transitions for a subject class. If they have not been, it is necessary to re-analyze the state model for the class in question. Every trigger event must be linked to a state transition of a subject class.

8.2.2 Application Role

Application roles are abstractions that standardize the roles played by healthcare information system components when they send or receive HL7 messages. The concept of the application role has been created in order to define those features and only those features of the systems sending and receiving HL7 messages that need to be standardized to support message exchange.

The purpose of the application role concept is to find a way for HL7 to discuss senders and receivers of messages without getting involved in issues of application functionality. In a sense, the application role resolves a dilemma for HL7 standards development. On the one hand, HL7 needs both to reflect the variety of ways a single message can be used while still moving towards more precisely defined messages. This reflects one of the major lessons from working with Version 2 messages – you can make a message flexible by making all of its features optional, but you can't implement such a message without resolving that optionality with a detailed implementation agreement. On the other hand, HL7 has no business trying to define how healthcare applications are structured by their creators. The resolution of this dilemma is to find a way to standardize the application's messaging behavior without standardizing the application. The application role is designed to provide this standardization.

The concept of application role, and its function in message standardization is one of the major features that distinguish HL7 Version 3 from previous standards. Here are three ways the application role supports the standards development process. The application role provides:

- **A tool to analyze the relationship between messages and key classes in the RIM**
Applications are defined "as stereotypes of a subject class". That is, when an application role is defined, it is defined as a messaging role of a subject class. For example, if PERSON is a subject class, a potential application role might be "PERSON information broadcaster". Another way to state this point, is that an application role represents a subset of the message behavior of one or more subject classes.
- **A way to define more interoperable messages**
An interaction is defined by the relationship of a trigger event, the sending application role, and the receiving application role. Therefore multiple interactions can be defined for a single trigger event, one for each combination of sender and receiver. This is a way for a technical committee to tailor message definitions to fit specific needs.

³ See Chapter 6 – Information Model – for more discussion of subject classes and their state transitions.

- **A foundation for conformance claims**

An application role can be characterized as the combination of the set of interactions for which it is the sender and the set of interactions for which it is the receiver. The application role can be thought of as a set of responsibilities with regard to interactions. That is, for each interaction where the application role is listed as the sender, it must send the appropriate message when it detects a trigger event. For each interaction where the application role is listed as the receiver, it must be able to receive the appropriate message and perform the receiver responsibility. The application role and its responsibilities form the basis for establishing conformance specifications. As explained in Chapter 9—Conformance Claims and Organizational Practices, a vendor application will claim conformance to an application role. This implies its ability to send and receive the interactions that have been defined for that role.

8.2.2.1 Subject Class Stereotypes and Messaging Modes

In order for application roles to fulfil these functions, application roles are constructed in a specific way. The goal is to focus on the abstract requirements of messaging based on the data that is being managed by and application or system, rather than attempting to define how functions should be bundled together. It is important for application roles to be related to information systems as senders and receivers of messages, not as bundles of function. The reason is for this is to keep HL7 from standardizing or from seeming to standardize application functionality. Such standardization would be both out of scope for HL7 and unwise.

The subject class has been chosen as the starting point for defining application roles. This choice is based on the fact that, within the Information Model, dynamic behavior, as shown by a defined life cycle and state model, is restricted to subject classes. More specifically, application roles should be chosen to indicate specific ways that a subject class is related to messaging. The technical committee has been given considerable leeway in doing this; because HL7 has little history of defining application roles. Once some concrete experience has been gained, it may be possible to be more prescriptive.

Given that an application role characterizes the messaging behavior of a subject class, several approaches have been suggested:

8.2.2.1.1 Messaging modes

The technical committee should start the process of developing application role stereotypes from the perspective that there are three fundamental purposes for message exchange, three basic “messaging modes”. These are:

- **Declarative**

These are messages whose purpose is simply to convey information from one party to another. In Version 2.x these were known as unsolicited updates. For example, a healthcare provider might send a message to transfer a patient from one room and bed to another.

- **Imperative**

These are messages that direct or request a party to do something. For example, a healthcare provider might send a message to a laboratory every time the provider needs the laboratory to perform a test. Note, that even though the message must include information about the test and the patient the test is for, the primary purpose of the message is to request that the test be done.

- **Interrogative**

These are messages that ask for information, that ask a question and expect a response. In Version 2.x these were known as queries. For example, a healthcare provider might send a message to an MPI mediator asking whether the MPI mediator has information about a specific person.

This approach recommends that application roles be defined by using stereotyped names constructed as <subject class> <relationship>. One useful way to construct these stereotypes is by considering roles that are specific to the messaging modes.

- For the declarative mode, the typical relationships are “declarer” and “recipient”. It might be productive to have two types of declarer – the “tightly coupled declarer”, and the “loosely coupled declarer. The notion is that tightly coupled systems share information easily so that short messages can be constructed that would send information for the subject class, and include foreign key references for classes related to the subject class. The sender would assume that, if the receiver did not have all the information needed, it could query the sender for that information. On the other hand, loosely coupled systems would always send a full set of information to avoid having to support the ability to respond to queries if previously transmitted information had been discarded.
- For the imperative model, the typical relationships are “placer” and “filler”
- For the interrogative mode, the typical relationships are “questioner” and “answerer”

8.2.2.1.2 Role Stereotypes

In this case, the source for application roles is still the subject classes within the Information Model. However, the roles are defined in relationship to the subject class according to the way a system relates to or uses a class; that is according to the role the system takes with respect to the class. The following roles have been suggested: manager, tracker, and historian. Note, this approach has been chosen, in place of a broader set of names(e.g., order processing system, hospital information system, laboratory processing system) so as to avoid make judgments about the way in which the functions that need to be supported for healthcare are partitioned within actual implementations.

The potential application roles are suggested in this document and should be used as a first step. If others seem to be needed, they should be added. Subsequently, the Modeling & Methodology Technical Committee will create standard list which will be maintained according to the needs of the technical committee.

8.2.2.1.3 Eclectic Approach

There may be a need to look at other ways of stereotyping the application roles. For example, if a class such as Service that is applicable to a wide range of functional situations is chosen as a subject class, the technical committee may want to use application roles to constrain the applicability of specific messages. For example, would “Microbiology Service Filler” be an appropriate application role. More discussion is needed here.

8.2.3 Interaction

Clearly, interactions are what the Interaction Model is all about. More specifically, an interaction is a single one-way transfer of information from a sending application role to a receiving application role for a single trigger event⁴. Each interaction can be supported by a message definition, a Hierarchical Message Description (for more detail on the HMD see Chapter 10 – Creating Message Specifications). The point of an interaction is not so much the interaction itself – it really just consists of a line on a diagram or an identifier on a list – it is the combination of trigger event, sender application role, receiver application role, and receiver responsibility that the interaction creates.

The reader should note that, within itself, an interaction cannot directly specify a return message. An interaction may, however, establish a responsibility for the receiver of its message, and this responsibility may require that the receiver initiate a particular trigger event/interaction subsequent to the receipt. This, in

⁴ The reader can see from this definition why the section that offers a more detailed description of the interaction follows sections covering trigger events and application roles.

fact, is the final element in definition of the interaction; the determination of whether there is a receiver responsibility for the interaction. That follow-on interaction may have the effect of continuing or completing a transaction that requires two or more linked message exchanges. The receiver responsibility indicates any action that should be carried out by the receiver of a message. Within the current scope of HL7 Version 3, the only receiver responsibilities contemplated are to send an additional interaction or interactions. For example, if we define an interaction that covers communication of clinical orders, the receiver may be required to indicate whether or not it can perform the order. In such cases, it will be useful to refer to the Use Case Model to see if the use case involves a series of steps – perhaps requiring several interactions – for its realization

The decision to define an interaction is an important one for the Technical Committee. There must be an interaction for each trigger event in order to lay the basis for the Hierarchical Message Description that replaces Version 2.3's abstract message. However there can be more than one such interaction. Why would a Technical Committee define multiple interactions for a trigger event? In order to lay the basis for an HMD that more precisely defines the contents of the message (i.e., to reduce optionality).

The decision to define multiple interactions for a trigger event should be weighed carefully by the Technical Committee. At this point, the Modeling & Methodology Committee knows of two situations in which this is a preferred option:

1. If, for a trigger event, there is a substantial difference in the data that is required between tightly coupled and loosely coupled systems, multiple interactions are a reasonable way to express this difference. For example, when an episode is created (e.g., registration or admission) a receiver that is tightly coupled to the sending system will already have access to information about the patient, while one that is loosely coupled will not. This difference may be substantial enough to justify multiple interactions. In this situation, the Technical Committee should use the interaction description to indicate whether the interaction does or does not imply tight coupling between the systems exchanging messages.
2. If there are multiple application roles that can be receivers for the interaction, and IF the data that is needed differs substantially between the roles, multiple interactions related to a single trigger event are appropriate. For example, when a test order is created, there could be different information required if the data is being transmitted to a clinical archive as opposed to a test manager who will perform the test.

In some cases, trigger events may be generic. For example, if Create Encounter is defined and is expected to cover inpatients and outpatients. If the information required is substantially different between the two cases, this is NOT a reason to create multiple interactions for the trigger event. Instead, the proper course is to define more specialized trigger events. As a general principle, if the functional context triggering two interactions is different, then there should be two different trigger events. If the functional context is the same but there is good reason for the interactions to be different, then two interactions for one trigger event are appropriate.

8.2.4 Interaction Sequence

An interaction as an atomic unit of communication may stand on its own. For example, the admission of a patient to a healthcare facility might simply be associated with the need to inform interested parties of the admission, and to pass along some relevant information. This situation implies the existence of a trigger event, it might be called "Admit Patient". This trigger event would be associated with an interaction that transmitted patient and encounter information to the party or parties that needed the information. Ideally,

this functional requirement would be uncovered through working with a use case from the Use Case Model. In this case, the messaging for the trigger event is complete once the first interaction has been sent.⁵

However, there may be situations in which multiple, coupled interactions are associated with a single trigger event. For example, In HL7 Version 2.3 and before, the trigger event, “Doctor orders test” leads to two interactions. First, an order interaction that goes from the placer system to the filler system. Secondly, a response interaction that passes the filler number for the order back to the placer system. More complex chains of interactions may also be indicated. As mentioned above, this indication is best discovered through working with the Use Case Model.

An interaction sequence is a time-ordered sequence of healthcare relevant events as a sequence of interactions. Scenario sequence diagrams (these will be discussed below) are used to document interaction sequences that the modeling committee expects will typically occur in the domain. The scenario sequence diagram shows how a scenario description is represented as a sequence of interactions. Interaction sequences may be constructed to illustrate a storyboard – see Chapter 5, Use Case Model. They may also be constructed to define a series of interactions that must be satisfied as part of a conformance claim.⁶

There is another case in which there is a multiplicity of interactions related to a single trigger event. This occurs if a technical committee defines multiple application roles as the receivers of the interaction associated with a single trigger event. Naturally, this implies definition of multiple interactions for that trigger event (as stated above, an interaction is associated with 1) a trigger event, 2) a sending application role, and 3) a receiving application role) This situation can arise when a technical committee identifies multiple settings in which a trigger event can be detected, and determines that the difference in the settings is sufficiently important to imply multiple interactions. For example, it might be the case that the trigger event “Register Patient” is considered to imply one message when the patient is registered at an acute care facility, and another when the patient is registered at a test facility. This is the kind of question that needs to be determined by the technical committee.

8.3 Diagramming Interactions

Interaction diagrams are constructed to show the interdependencies between interactions, between messages. What is an Interaction diagram? Martin Fowler describes an interaction diagram as a “model that describes how groups of objects collaborate in some behavior. Typically an interaction diagram captures the behavior of a single use case. The diagram shows a number of example objects and the messages that are passed between these objects within the use case.”⁷

The UML (Unified Modeling Language) provides two forms of interaction diagram, and either of these can be used by HL7 developers. Each is discussed below.

8.3.1 Sequence Diagram

The interactions that support a use case or a storyboard defined for a use case can be represented in a scenario sequence diagram. The interaction sequence diagram is a graph that shows application roles as vertical bars and arrows which represent the communication between these application roles, i.e., interactions or messages. Each interaction is shown as an arrow, and the order, from top to bottom, shows the time sequence of the interactions. Therefore, it shows how messages, initiated by a trigger event, flow in order to achieve a specific end.

⁵ This does not include the requirement for a system response that indicates an interaction was received. That response is assumed for all interactions and does not have to be documented.

⁶ Note, the collection of interaction sequences that HL7 may publish does not limit the ways in which HL7 can be applied. The definition of an interaction sequence is illustrative, not mandatory. Other combinations of trigger events, interactions, and application roles that are consistent with the interaction model may also be used.

⁷ UML Distilled, Page 103.

Sequence diagrams are used to show how an interactions are coupled through receiver responsibilities in order to achieve a purpose. Ideally, this purpose will have been documented as a use case or storyboard within a use case.

A special form of sequence diagram shows, for a given application role, all the interactions that it participates in, both as sender and receiver. This becomes a graphical representation of the responsibilities that have been assigned to that application role, and can be used to depict the substance behind the conformance claim that mandates support of that application role.

8.3.2 Collaboration Diagram

A collaboration diagram provides another way to show how multiple interactions, and multiple application roles are inter-related. In a collaboration diagram, application roles are shown as rectangles, and the interactions as arrows between the boxes. If interaction sequence is necessary, it can be indicated by numbering the interactions. The collaboration diagram is useful in showing how multiple application roles, each which might detect trigger events, work together. A technical committee might work with a collaboration diagram at the beginning of its work to try to get a grasp of the scope of work for a potential new project.

8.4 Conformance and the Interaction Model

A crucial feature of HL7 Version 3 is that it lays the basis for reasonable conformance claims. Conformance claims are based on the assertion that an application or system supports an application role. (Of course, it could support multiple roles.) The conformance claim provides a summary representation of the different sender and receiver roles that have been defined for an application role. In other words, it defines the messages which an application role is expected to be able to send and receive. Ability to receive an interaction also includes the ability to carry out the receiver responsibility as defined for the interaction.

Once the set of interactions that is needed for a project or a specific use case has been constructed, conformance claims can be developed by looking at each application role and considering firstly the interactions that the application role is the sender for, and secondly the interactions that the application role is the receiver for. The conformance claim states the responsibility of an application that claims conformance to that role. For more details, see Chapter 9, Conformance Claims and Organizational Practices.

8.5 Building the Interaction Model

8.5.1 Defining Scope

Developing the contents of the Interaction Model is the first step towards building messages since each interaction defined is a potential message, and will become one once the process of HMD (hierarchical message diagram) construction has been completed. Development of individual messages can be laborious, and, it will be important for the technical committee to develop a set of messages that work together. Therefore, it is valuable for the committee to set the appropriate scope for a new message development activity.

In order to start interaction development, it is important to have defined subject classes in the Information Model, and to have constructed the needed state model. It will also be valuable to have a documented, rather than implicit, Use Case Model

8.5.1.1 Use case scope definition

The Use Case Model is a valuable tool for defining the scope of the technical committee's efforts. Once the committee's use cases have been documented, they can be prioritized, and then the committee can seek to develop the set of interactions needed to support an individual use case. When storyboards have been defined for a use case, the committee will find it valuable to verify that it has defined all the interactions needed to support each storyboard.⁸

8.5.1.2 Subject class scope definition

In the final example, interactions are discovered through examination of the state transition models developed for subject classes. Even, if the Use Case Model is that starting point for the analysis, there must be a state transition to support each use case. Starting with the state model, each state transition within a subject class state transition model represents a potential interaction. An interaction should be created for a state transition if the state change needs to be communicated between the party detecting or causing the state change, and the party or parties that have a need to know about that state change. In other words, the state transition is a potential trigger event.

8.5.1.3 Other scope definition

In special cases, there will be other drivers for a technical committee's work on interactions. It might be important to replace a set of Version 2 messages, or to full a particular functional purpose. In this case, it will still be necessary to verify that needed trigger events can be derived from the Information Model as discussed above. It will also be valuable to consider construction of use cases so that the requirements for the interactions can be properly discussed and agreed upon.

8.5.2 Building Interactions

Given the trigger event and other information such as the sender and receiver system, provide a description for the interaction. In some cases, multiple interactions are created for a single trigger event to allow more precise specification of the HMD that will be created to support the interaction. Define receiver responsibility. In some cases, receipt of an interaction implies a specific responsibility for the receiver of an interaction. Document the responsibility for each interaction. These responsibilities will become important when conformance claims are defined. Note that the actual detailed specification of the data transferred in the interaction will come later, in the Hierarchical Message Description. At this point we are describing the act of transferring the information rather than the contents.

An interaction is a transfer of information from sender to receiver that is needed to support a use case from the Use Case Model. Each interaction also indicates the need to have an HL7 message specified that will carry the data needed for the interaction.⁹ Interactions are defined by expanding on the trigger event that is to be supported. That is to say, the trigger event is the starting point for defining an interaction – it defines when messages need to be sent. ” The sender and receiver for the interaction need to be defined, as does any receiver responsibility. Senders and receivers are defined in a specific way, as application roles. The definition process is finished by filling out the other items related to the interaction. The full definition of an interaction includes definition of any action on the part of the message receiver that is needed to make the interaction appropriate, the “receiver responsibility.

To fully construct interactions, the Technical Committee will provide the following items of information about each interaction:

- **Initiating Trigger Event:** The trigger event that triggers or initiates this interaction.

⁸ See Chapter 5 – Use Case Model – for a more in depth discussion of the role of use cases.

⁹ Look in the following section that describes the Message Information Model for detailed information on this process.

- **Sending Application Role:** This is the application role that has responsibilities to recognize the trigger event for the interaction and to cause the appropriate message to be sent.
- **Receiving Application Role:** This is the application role that is responsible for receiving the message involved in this interaction. The receiving role must be prepared to accept the message and to fulfill the receiver responsibility.
- **Message Transferred:** The identifier of the message format for the message that the interaction transfers.
- **Receiver Responsibility:** Definition of a follow-on interaction that the receiver must initiate. This is an optional element in that there may be no follow-on responsibility. If a responsibility exists it might reference a subsequent action on a subject class that will lead to a state transition and thus to a subsequent trigger event and interaction. Thus, transactions can be established through a chain of receiver responsibilities for individual interactions.
- **Interaction Identifier:** An interaction identifier must be established and recorded.

Note that the enumeration of all Interactions in which an application role participates, as sender and as receiver, is the basis for HL7 conformance claims, so it is important to review the designation of interaction senders and receivers to ensure that a coherent set of application roles emerge.

8.5.3 Validating Interactions

Interactions are validated by going back to the Use Case Model, or to the use cases developed from a specific project scope, and verifying the existence of a set of interactions that support the use case. In some cases, the technical committee may want to work with scenario sequence diagrams that are developed to demonstrate support of a use case or use cases. The scenario sequence diagram shows a sequence of interactions that support either a use case, or a scenario that has been documented for a use case. This demonstrates that the requirements of the use case are supported by one or more interactions between designated application roles.

Validating interactions also includes consideration of possible conformance claims. For each application role, the technical committee should look at all the interactions that have that application role as a sender and as a receiver. The technical committee needs to consider whether that set of interactions comprise a reasonable conformance claim. This can be done by first looking at all the interactions that a given application role is expected to send. Is it a coherent list? Does the set of interactions cover all the state transitions of a subject class that an HL7 application should be able to manage? If the interactions relate to multiple subject classes, is it reasonable to link them together within a single conformance claim? After this kind of consideration for the interactions the application role is expected to send, the same should be done for those it is expected to receive. Finally, the technical committee should consider whether the interactions the application role is expected to send and those it is expected to receive fit reasonably together.

What if the model defines different sender roles for a use case but the interaction for each is the same? Perhaps the project team didn't need to define multiple types of sender role. In this case, the project team should generalize the sender type so that only one is needed for the trigger event. Remember, the idea of specifying more than one type of sender for a trigger event is to allow each type to send a different interaction.

8.5.4 Grouping Interactions

Interactions can be grouped in order to easier review related interactions. The capability for grouping is included within the HL7 tools that are documented in their own chapter within this Message Development

Framework. The technical committee should determine its criteria for grouping interactions. However, one key criteria for grouping is sharing of application roles, and the technical committee should group all interactions for a single sender application role.

8.6 Interaction Model Quality Criteria

The following points should be used for quality assurance of the Interaction Model. These issues should be considered while the model is being developed, as well as during the review process.

- All trigger events must be linked to documented subject class state transition.
- Interactions should be fully defined. For each interaction, the trigger event, sender role, receiver role, receiver responsibility (if there is none, it should be stated), and interaction ID should be defined.
- Interaction diagrams should be accurate. Sequence diagrams should show the flow of messages implied by interaction receiver responsibilities, and application roles should be appropriately positioned.

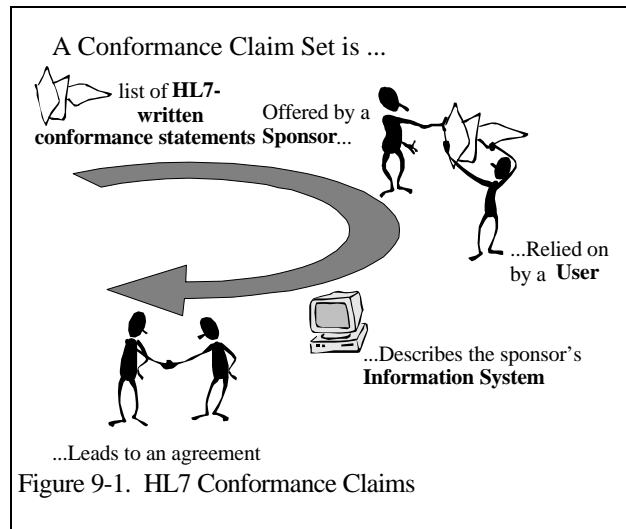
9. Conformance Claims and Organizational Practices

9.1 Introduction

This chapter describes how conformance criteria are stated and used in Version 3.

Most conformance criteria are stated in terms of the performance of a healthcare information system with respect to its interfaces. As such the burden of conformance falls primarily on the developer or installer of the information system. Our goal with respect to these conformance criteria is to provide a concrete definition of conformance that can be the basis of contractual agreements and conformance testing.

In other cases, HL7 states good organizational practices for using HL7. As there is no contractual relationship between HL7 and user organizations, HL7 does not expect to be able to audit or otherwise enforce compliance. Nonetheless, it is extremely valuable to state these principles in a clear and concise way.



9.1.1 Conformance Claims

One of the primary benefits of Version 3 is that conformance criteria are stated in terms that are sufficiently concrete to use in contracts and test the conformance of a system.

The parties that need to communicate about this are the sponsors and users. The *sponsor* of an information system is the vendor, in-house developer, or provider of public domain software. For this chapter, the *user* of a health care information system is the organization that buys or leases the information system or employs an information system for which the software was developed in-house or is in the public domain.

The scope of HL7 is very broad; very few if any systems will conform to all the application roles contemplated for Version 3. For this reason, a general statement such as “system X conforms to HL7” is imprecise. A user wants to know “to what parts of HL7 does it conform”?

In order to be precise, the specifications created by the HL7 technical committees describe entities that constitute individual statements of conformance criteria. The sponsor of a healthcare information system combines them into a conformance claim set by simply publishing the list of individual conformance statements that it claims. This list is called a *conformance claim set*. The user relies on the conformance claim set to know the HL7 capabilities of a sponsor’s system, and its ability to interact with other systems, before making a commitment to purchase or otherwise employ the system under consideration. The user can expect that the sponsor’s system *fully* conforms to each of the statements in its conformance claim set. After a contract is signed, the conformance claim set serves as a basis for resolution of differences in the expected and actual implementation of HL7.

As certification methods are developed for Version 3, there will be explicit tests to validate the conformance of a sponsor’s systems to the statements of conformance criteria.

9.1.2 Good Organizational Practices

In this revision of the MDF, good organizational practices are stated primarily with respect to vocabularies (code sets).

9.2 Conformance Claim Work Products

As described in the introduction, statements of conformance criteria are designed to provide precise descriptions of the HL7 functions a system must support with respect to a specific part of the HL7 specifications. They are defined by HL7 and given an unambiguous identifier.

A conformance claim set is a list of the identifiers of specific HL7 statements of conformance criteria.

There are two kinds of statements of conformance criteria: functional and technical. A functional statement of conformance criteria is, exactly, a commitment to fulfill the responsibilities of a Version 3 application role. Functional statements of conformance criteria are completely specified by functional technical committees as they define interactions relating to an application role. In other words, they are claims to:

- send certain HL7 messages to systems that conform to certain other application roles in response to certain trigger events
- receive certain HL7 messages to systems that conform to certain other application roles, and
- upon receipt of certain messages, perform the receiver responsibilities

Furthermore, the sponsor enumerates all optional conformance message elements associated with a conformance claim, and identifies:

- whether it can send or receive the conformance message element
- if the conformance message element is of data type multimedia-enabled free text, which of the media types it supports. The multimedia-enabled free text data type is described in Chapter 6.

A technical statement of conformance criteria is any other testable claim. This will include, but not be limited to, claims to employ specific modes of transmission of HL7 messages (XML, CORBA, OLE, etc.) In general, the Control Technical Committee writes technical statements of conformance criteria.

9.2.1 Conceptual Model of HL7 Conformance Claims

Figure 9-2 is a conceptual model that illustrates the relationship of these concepts. Each class box belongs to one of three categories:

- conceptual classes are not precisely defined. They are included here solely to illustrate the concepts.
- information that is provided by sponsors of systems in work products that characterize their systems' conformance with HL7
- work products of HL7.

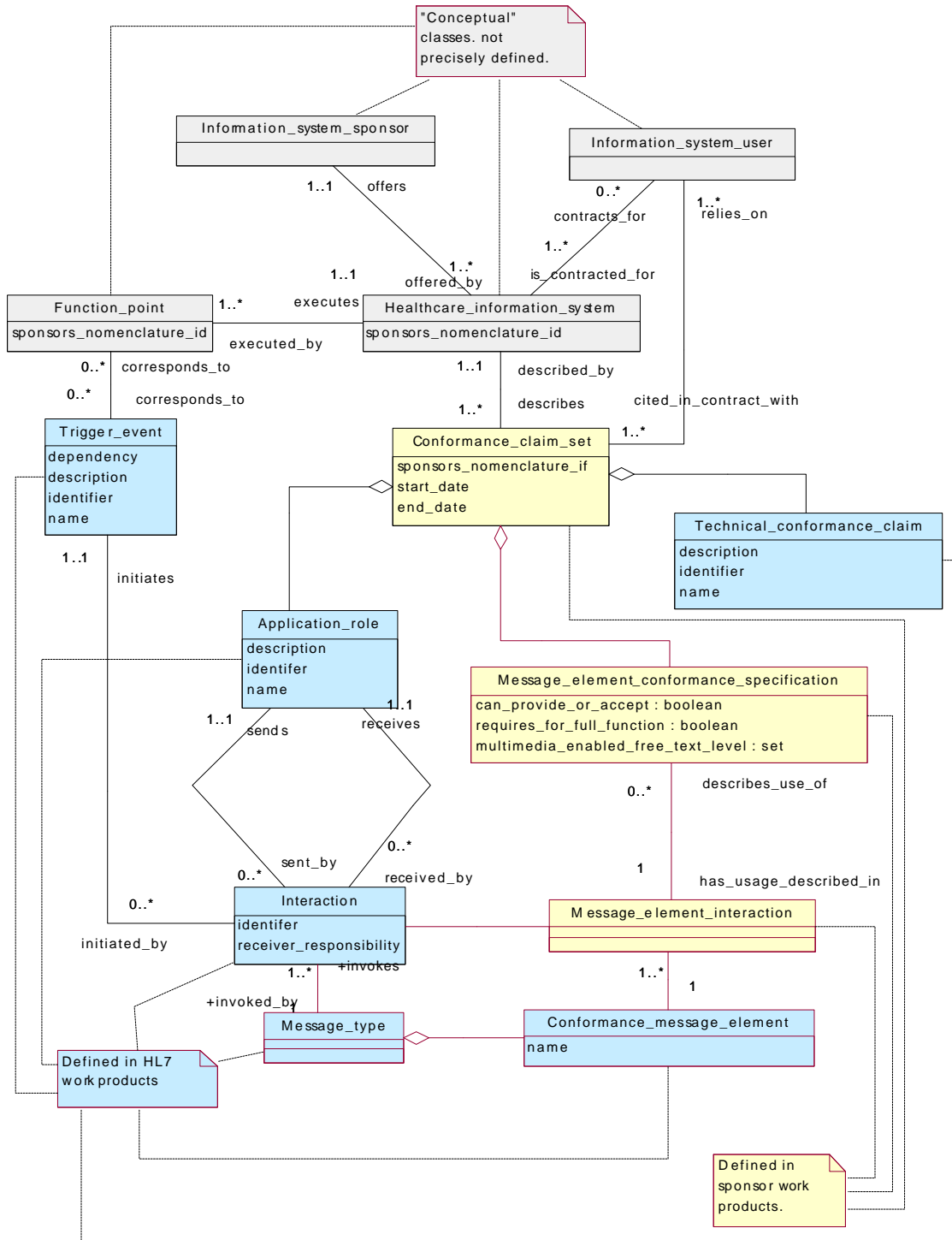


Figure 9-2. Conceptual Model of Conformance Claims

The classes and attributes of the conceptual model are listed below.

9.2.1.1 Classes and Attributes of the Conceptual Model

9.2.1.1.1 Information_system_sponsor

An organization that is a vendor or other developer of an information system. This organization bears the responsibility for the system performing as specified with respect to HL7 interfaces.

9.2.1.1.2 Information_system_user

An organization that purchases or otherwise acquires a healthcare information system. This organization relies on the HL7 interfaces for interoperation among healthcare information systems from one or more sponsors.

9.2.1.1.3 Healthcare_information_system

A healthcare computer application that is a collection of functions that is contracted for, installed, or implemented as a unit.

9.2.1.1.3.1 sponsors_nomenclature_id :

The sponsor's text that identifies the healthcare information system and distinguishes it from its other healthcare information systems.

9.2.1.1.4 Function_point

Any function, user transaction, or other interaction or event in the sponsor's application which, when it occurs, does or may correspond to an HL7 Trigger Event.

9.2.1.1.4.1 sponsors_nomenclature_id :

The sponsor's text that identifies the healthcare information system and distinguishes it from its other healthcare information systems.

9.2.1.1.5 Conformance_claim_set

A document or computer file that describes the conformance of a healthcare information system to HL7. This document is prepared by the information system sponsor. There is no requirement that it use the same conformance claim set each time a specific healthcare information system is proposed or installed. However the conformance claim set provided to a specific information system user is meant to be part of the contract between the sponsor and user.

9.2.1.1.5.1 sponsors_nomenclature_if :

The sponsor's identification of the conformance claim set.

9.2.1.1.5.2 start_date :

The first date where the conformance claim set is valid.

9.2.1.1.5.3 end_date:

The last date where the conformance claim set is valid. (Optional).

9.2.1.1.6 Trigger_event

A trigger event, as defined in the HL7 Version 3 Interaction Model. This is part of the normative HL7 specification.

9.2.1.1.7 Application_role

An application role, as defined in the Version 3 methodology. Application roles are defined by HL7 technical committees as one of the work products of producing a standard specification.

9.2.1.1.8 Interaction

An interaction, as defined in the HL7 Version 3 Interaction Model.

9.2.1.1.9 Technical_conformance_claim

A conformance claim written by an HL7 Technical Committee. A technical conformance claim describes some aspect of the HL7 standard that is different from, and independent of, the functional statement of conformance criteria that are asserted through application roles. The statement is made in natural language text. In order to be a technical statement of conformance criterion it must meet as unambiguous as is possible in natural language and be testable.

Candidates for technical statement of conformance criteria include the Implementable Technology Specifications and special HL7 protocols such as batch or sequence number.

9.2.1.1.9.1 description :

A short phrase that serves as a title for the technical conformance claim.

9.2.1.1.10 Conformance_message_element

A message element that is defined in a Hierarchical Message Definition.

9.2.1.1.11 Message_element_conformance_specification

A declaration by the information sponsor of its performance with respect to a specific conformance message element when it is used within a specific interaction.

9.2.1.1.11.1 can_provide_or_accept : boolean

If true, the healthcare information system can accept and use the conformance message element when it is the receiver of an interaction that contains it, and it can provide data in the conformance message element when it is the sender of an interaction.

9.2.1.1.11.2 requires_for_full_function : boolean

If true, the performance of the healthcare information system is somehow impaired if another healthcare information system does not provide a value for this conformance message element.

9.2.1.1.11.3 multimedia_enabled_free_text_level : set

If the data type of the conformance message element is multimedia-enabled-free-text, then this is a set of the codes for the media descriptor component of the data type that the healthcare information can send or receive. This set must

include the mandatory values for the data type and must be compliant with constraints placed on this conformance message element by within the hierarchical message description for the conformance message element.

9.2.1.1.12 Message_element_interaction

A conformance message element, as it is used in a specific interaction.

9.2.1.1.13 Message_type

A message type, as defined in a Hierarchical Message Description.

9.2.2 Functional Statement of Conformance Criteria

Each application role defined in a release of HL7 Version 3 specifications is a functional statement of conformance criteria. In claiming conformance to an application role, the sponsor is asserting that it fulfills the obligations of an application role. This specifically means that it sends all of the interactions that are required to be sent, and receives all the interactions that the application role is required to receive, and performs the receiver responsibilities associated with them.

It is important to notice that each interaction is associated with one and only one application role. The same message type may be used in response to the same trigger event in another application role, but that interaction will have a different identifier. Likewise the same message type may be used in response to a different trigger event in the same application role, but this will also be a different interaction.

9.2.2.1 Sending Interactions

The sponsor claims that its information system will send an interaction in response to the HL7 trigger event that initiates it. The sponsor also claims that the message will be sent using the message type prescribed by HL7 in Hierarchical Message Description (HMD), the structure of which is described in Chapter 13 of this MDF. The HMD defines the sequence, grouping, repetition and abstract type of message elements in the message. It does not, however, define their physical format or the manner in which it is sent. (These vary according to the application of different Implementable Technology Specifications (ITS) to the HMD. Correspondence to ITSs will be referred to in technical statements of conformance criteria.)

The interpretation of the terms *null*, *not present* and the possible inclusion values in the HMD are critical to determining conformance.

We use the term **conformance message element** to mean any element of a message that is either a component of the message type or a component of any component of the message type, to the entire depth of the tree that is represented in the HMD.

Certain conformance message elements have special treatment, because they correspond to attributes of the Reference Information model. They are called **RIM attribute elements**. The statement of conformance criteria is different for RIM attribute elements.

9.2.2.1.1 Inclusion

Inclusion refers to the conditions under which a message element appears in a message. The term relates directly to the determination of conformance. This section defines how they are used when relating to interactions that are sent by a sponsor's information system.

The possible values for the inclusion of a message element are:

Mandatory. The message element must appear every time the message description is used for an Interaction (subject to the rules of multiplicity and conditionality.) In addition, if the component represents a RIM attribute, it must *not* have any of the null values described in Chapter 6.

Optional. The element need not appear in every message instance; if it appears, and it is a RIM attribute it can have a null value.

9.2.2.1.2 Conformance

Required. The message element must appear every time the message description is used for an Interaction (subject to the rules of multiplicity and conditionality.) Note that all message elements that have an inclusion value of *mandatory* must have a conformance value of *required*.

Not Required. The message element may be populated or used by one system sponsor, but not by another. Each system sponsor is required to state its ability to accept or send the message element as part of its conformance claim.

Not Permitted. This message element may never occur when the message type is used for an interaction. (Having this is an artifact of using a single HMD to describe multiple message types.)

9.2.2.1.3 Mapping of Sender's Information System to the Interaction

In describing a sponsor's information system we define the term "function point" as any system function, user transaction, or other interaction or event which, when it occurs, does or may correspond to an HL7 Trigger Event. It is difficult to be more specific because of the variety of architectures and terminologies used by in various information systems. The relationship between trigger events and function points is many-to-many. More than one function point may represent the same trigger event. For example "discharge AMA" and "patient death" may be handled on different screens although they both represent the same trigger event. It is equally true that a single function point may represent different trigger events. For example, "discharge mother and baby" may represent two unique discharge trigger events.

A claim that an information system corresponds to an application role implies the following statements:

The mappings between data elements in the sender's database and the message must conform to the semantics of the message information model and its specific relationships that are named in the HMD. (This does not imply that the sender's database uses the same names for the data elements or has the same structure as the appropriate HL7 message information model.)

Whenever any function point occurs that corresponds to any HL7 trigger event that initiates an interaction, the system will send all the interactions specified for the application role.

9.2.2.2 Receiving Interactions

A sponsor claiming to conform to an application role also has requirements when receiving interactions.

9.2.2.2.1 Inclusion

In general, the receiving system must map appropriately all message elements of the interaction instance to the corresponding data in the receiving information system. If, however, the message element has a conformance value of *not required* and the sponsor does not claim to support the message element, then this requirement does not apply.

The data so mapped must appear correctly in screens, reports, and other outputs of the system. There is no definite rule specifying the information content of specific screens, reports, and other outputs. (This is beyond the scope of HL7.) However, if a specific data element does occur on a screen, report, or other output, and the corresponding data field has been received in an HL7 message, and it has not been

overridden by subsequent actions of the users of the information system, the value in the last HL7 message should appear.

The mappings between data fields in the message and the receiver's database must conform to the semantics of the message information model and its specific connections that are named in the HMD. (This does not imply that the sender's database uses the same names for the data fields or has the same structure as the appropriate HL7 message information model.)

If the trigger event for which the message is sent corresponds to one or more function points in the receiver's system it should take the actions that correspond to the function point. Suppose, for example, that the function point that corresponds to canceling an order creates consequential updates to the patient account and work list portions of the database of the information system. The receipt of an HL7 message that corresponds to that functional point should trigger the same updates.

Sometimes the receiving information system may choose to recognize the receipt of an HL7 message instance as a distinct function point from the local execution of the same information. (For example, it may choose to defer the transaction for verification.) It is not a failure to conform to have a reasonable but different workflow for the recognition of events through HL7 messages.

9.2.2.2.2 Receiver Responsibilities

Certain interactions include responsibilities of the receiving information systems to initiate other interactions. In claiming that its information system corresponds to an application role, the sponsor is claiming that it performs all receiver responsibilities.

9.2.2.3 Simultaneously Sending and Receiving Interactions

In many cases a sponsor's information system may conform to application roles that it make the information system both the sender and receiver of the interaction. It may implement this transfer of information in any manner at all, not necessarily using HL7.

In certain cases a sponsor's information system must send an interaction both to itself and to another application role that is implemented in another information system. In this case it need not use HL7 for the information transfer to itself, but it must use HL7 interactions for the transfer to the other information system.

9.2.3 Technical Statements of Conformance Criteria

A technical statement of conformance criterion is a statement that an information system corresponds to some aspect of the HL7 standard that is different from, and independent of, the functional statement of conformance criteria that are asserted through application roles. The statement is made in natural language text. In order to be a technical statement of conformance criterion it must meet as unambiguous as is possible in natural language and be testable.

Candidates for technical statement of conformance criteria include the Implementable Technology Specifications and special HL7 protocols such as batch or sequence number.

9.2.3.1 Technical Conformance Criteria with Respect to Vocabularies

The Vocabulary Technical Committee will write one or more of the technical statements of conformance criteria. They will include these principles:

This system has the ability to produce a report of all local codes in use in a specific implementation.

This system has the ability to produce an HL7 message containing a report of all local codes in use in a specific implementation.

This system has the ability to change the code in use for a concept without invalidating system outputs, including reports that include times before and after the change.

This system has the ability to replace the code in use for a concept with several that represent subspecies of the concept without invalidating system outputs, including reports that include times before and after the change.

9.3 Good Organizational Practices

Statements of good organizational practice will be written as a series of statements about the policies of a generic user organization. To the maximum extent practical it should be possible to objectively determine the performance of an organization with respect to the statements.

9.3.1 Good Organizational Practices for Vocabulary

The Vocabulary Technical Committee will write one or more statements of good organizational practice that embody the following principles.

The organization will use standard codes if they exist. If the concept to be sent exists in the domain specification for the field, a code from the domain must be sent. In other words, one can not send a local code for something that exists in the standard domain.

The organization will not use a code from a standard set and give it a new meaning.

Where the organization uses a local code, it will use one that is consistent with the semantic type of national codes assigned to the same data field. In other words, if the field is Eye Color with values of blue, hazel, brown, it would be inappropriate to send a code meaning "Arm". (Note: A violation of this principle can only be detected by human review.)

Where the organization finds the necessity to use local codes, it will submit them to the HL7 for review and addition. HL7 will share these proposed concept additions with HL7 vocabulary source providers.

An organization will promptly begin using new codes when it upgrades its systems.

When an organization has submitted a local code requesting the creation of a standard code, and the responsible standards organization denies the creation of a new standard code, and the denial provides an alternative method to meet the need, the organization will adopt the alternate method.

10. Creating Message Specifications

10.1 Introduction

10.1.1 Overview

Previous sections of the MDF have described the analytic steps that prepare a committee to write the actual message specifications. Here we show how to complete the process, to actually define the structure of HL7 messages. This chapter relies on The HL7 Example Model, which is presented in Figure 10-6, Figure 10-7, and Exhibits 10-1, 10-2 and 10-3, which are at the end of the chapter.

Figure 10-1 reviews the prior steps and looks ahead. Its upper section depicts the entire process of defining a message. Within that section, the light gray background indicates prior steps. The white background shows the steps described in this section. The lowest section describes what a software system does in creating, sending, and receiving messages according to the specifications.

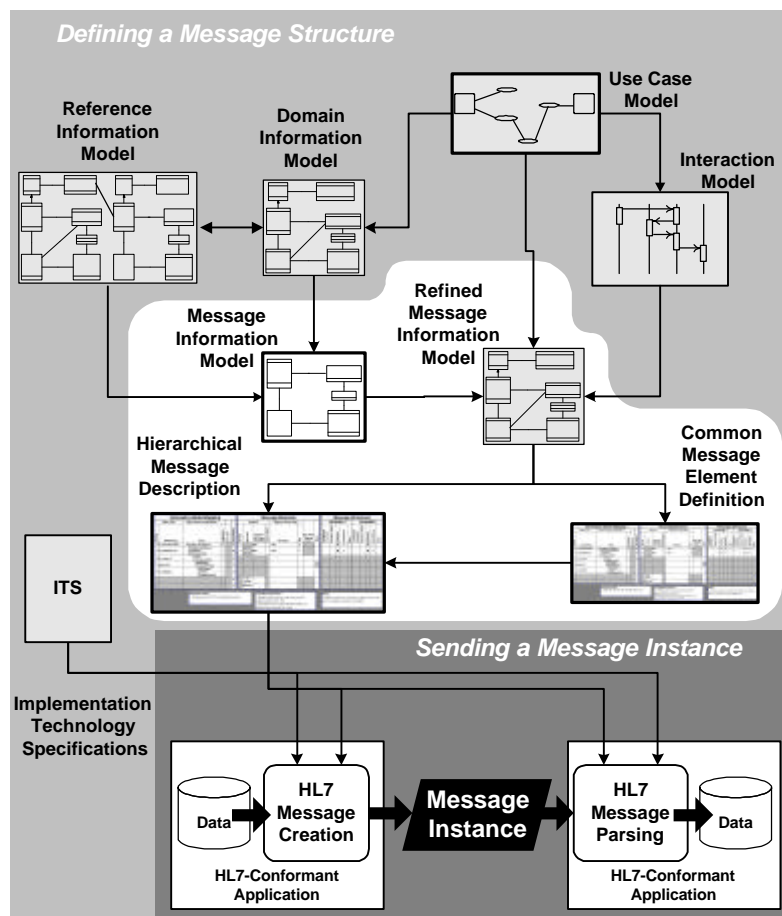


Figure 10-1. Creation of Message Specifications and Messages

In previous steps a committee has built a Use Case Model and a Domain Information Model and reconciled the Domain Information Model with the Reference Information Model. It has also created a **Message Information Model**, which is a subset of the Reference Information Model. In building the Information Models, the committee has designated certain classes as important, and built state diagrams for those classes. The committee has also prepared the Interaction Model, which identifies trigger events, relates

them to state transitions, and provided identifiers for **interactions**—the combination of a sending application role, a receiving application role, and a trigger event.

This chapter describes how a message type will be defined for each of these interactions. The committee will define the message type in the **Hierarchical Message Definitions** (HMD). It is these HMDs that are the normative expression of the standard.

Each Hierarchical Message Definition defines one or more message types. A message type is a recipe for creating a message instance. It specifies what data may appear as elements of the message and the order in which they will appear. Individual instances of a message are validated and decoded by referring to the definition of the message type. The instances may differ from one to the next because some optional elements of the message may not be present, or because elements repeat a different number of times, but each instance will comply with the type definition in the appropriate Hierarchical Message Definition.

Common Message Element Types are a critical part of the methodology. They are used to define message elements that may be re-used. For example, Common Message Element Types may be used to define the appropriate manner for identifying a new patient or a current inpatient, or for including a clinical observation in an administrative message. When the appropriate Technical Committee creates a Common Message Element Type, it achieves two important goals. It saves considerable work for other committees reading and understanding the RIM and designing message elements. Perhaps more importantly, it assures that message elements are designed by the committees that have the expertise in the appropriate subject area.

In a separate effort, the Control/Query Committee will have defined one or more *Implementation Technology Specifications* (ITS). Each ITS describes a different method for packaging the data and sending it from one application to another.

When a message is actually sent, an HL7-conformant application extracts data from its database, organizes it into the logical structure defined by the Hierarchical Message Definition, and formats it according to the rules of the ITS. It then transmits the data to another HL7 conformant application, which decodes the message using the methods of the ITS. The receiver can then apply the logical structure of the Hierarchical Message Definition to map the data to its database.

Some questions that are frequently asked are, “why do we need to create a Hierarchical Message Definition? If we already know the sender, the receiver, the trigger event and the classes, why not just send the data?” There are several important answers to this question.

- The information model contains a group of classes that frequently are interconnected in more than one way. For example there may be associations that lead from Patient to Person directly (this is the person who is the patient) and indirectly (this is the person who is the next of kin of the patient or this is the person who is the primary physician of the patient.)

The computers must be told which of the objects derived from these classes contain the data to be sent. Furthermore it must be told how the related objects can be found through the associations that are defined for the classes.

- The same attributes may not be appropriate for different objects derived from the same class. Although both the patient and the physician associated with a clinical order are people, it is unlikely that we will send the physician’s religion, date of birth or sex, each time we process an order for the patient.

The computers must be told which of the attributes of the objects will be sent.

- Associations may be used with different semantic contexts and have different cardinalities. For example, Figure 10-6 includes the association Person has (0..*) Person_name. The

hypothetical committee that designed the hypothetical message type in Figure 10-3 has decided that the information for each individual healthcare practitioner will be sent with exactly one name, while the information for each patient will have at least one name.

The computers must be told how many objects to send in a manner that is sensitive to the semantic context.

- Finally, to send data over the wire, the computer must organize it sequentially. There are many different ways to sequence the data from a group of objects interconnected by their associations. If the sender and the receiver do not agree exactly on the sequence, communication is frustrated. If one computer sends the information about the attending physician before that of the primary care physician and the other is expecting the opposite order, there will be a problem.

The computers must be told the exact sequence in which to send information.

The Hierarchical Message Definition specifies these choices. It defines a single message type that is used for an interaction without reference to the implementation technology. The Implementation Technology Specification describes how to combine data with the message type in order to create a message instance. This means that a message sent in the format of one implementation technology can be easily transliterated into the format of another.

The term *message* is often used to describe either the message type or the message instance. “The A01 message is used to send the information from the patient administration to all ancillary systems when a patient is admitted.” Or, “this A02 message didn’t have a date of birth.” In order to avoid confusion in this section of the MDF we will refer to the *message type* as the generic description of a message specified in the Hierarchical Message Definition, and *message instance* to speak of a specific message constructed according to the general definition.

10.1.1.1 Work Products

While developing the Hierarchical Message Definitions the Technical Committee will develop or use the following work products.

- A. The **Message Information Model (MIM)** is an information model that is used by a Technical Committee to present only the classes, associations, and attributes that are of interest to a specific set of message types. Its contents are defined in Chapter 6, Information Model.
- B. The **Refined Message Information Model (R-MIM)** represents the semantic constraints associated with a group of message types. It is particularly valuable when a message describes distinct views of objects drawn from the same class. For example, many messages will describe more than one person: the patient, a next of kin, a guarantor, and some providers. The attributes and, perhaps, the associations that will be used for each may vary.

The R-MIM is represented two ways: the **Graphical Refined Message Information Model** is a diagram in the Unified Modeling Language (UML); the **Tabular Refined Message Information Model** is a table that contains more details of the specific constraints that are associated with the views.

The Graphical Refined Message Information Model will represent the different views with distinct UML class icons.

- C. The **Hierarchical Message Definitions (HMDs)** completely define the structure of message types and the mapping to attributes and associations of objects of the classes in the Message Information Model. A Hierarchical Message Definition is a table that is partitioned into four major vertical sections:
 - The *Information Model Mapping* (left side). The columns that make up this section describe classes, attributes, and associations of the Refined Message Information Model.

- The *Message Elements* (middle). The columns that are in this section describe the elements of the message. They are row-wise related to the information model entities of the Information Model Mapping portion of the HMD.
 - The *General Constraints and Defaults* describe limitations on how message elements may be used when generating any of the message types described in the HMD.
 - *Message Type Structures* (right side). The columns that comprise this section are the same as those in the General Constraints and Defaults section. This set of columns is repeated for each message type defined in the HMD. Each message type is linked to one or more interactions in the interaction model. Row-by-row the structure definition tells whether to use the corresponding message element. The columns, when not blank, override the constraint or default specification in the corresponding column of the General Constraints and Defaults section.
- D. The committees may develop and will use **Common Message Element Types (CMET)**. A Common Message Element Type is a group of message elements that may be incorporated by reference into Hierarchical Message Definitions or into the definition of other Common Message Element Types. For example, a CMET may be defined to describe an accredited physician.

10.1.1.2 Procedures for Building a Hierarchical Message Definition

This section gives a brief overview of the procedures that a Technical Committee will use to specify messages by building Hierarchical Message Definitions. Figure 10-2 illustrates the relationship among the work products discussed in this chapter. A Message Information Model may be the basis for one or more Refined Message Information Models. A Refined Message Information Model may be the basis for one or more Hierarchical Message Definitions (or Common Message Element Types).

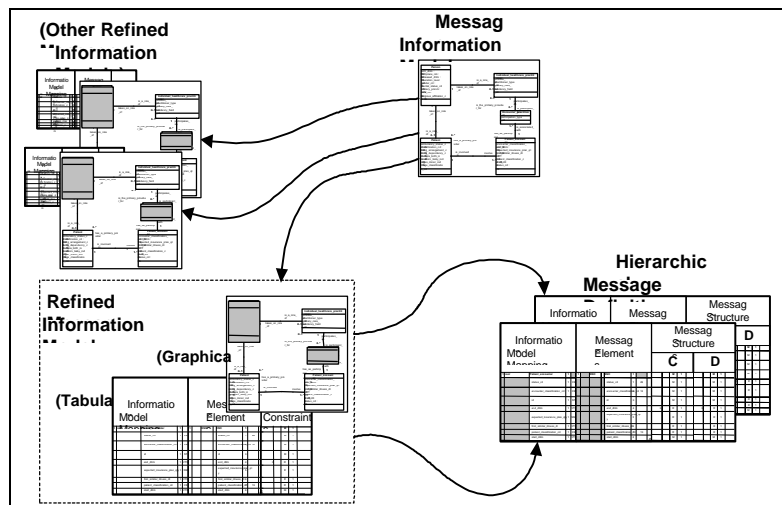


Figure 10-2. Relationship Among Work Products

Figure 10-3 illustrates the relationship between message types, interactions, trigger events, and application roles. In the hypothetical figure, the message trace diagram at the top represents an excerpt from the HL7 Interaction Model. At the bottom is a figure representing a Hierarchical Message Definition. The trigger event, “Admit Patient”, is linked to three interactions which the application role Encounter Manager must initiate (interactions number 3, 4, and 5). Of these, interaction 3 is intended to go from one system taking on the Encounter Manager application role to another that takes on the same role. Interaction 4 is a message intended to go to a system taking on the Encounter Tracker role. Interaction 5 is intended to go to a system taking on the Encounter Archivist role. Interactions 3 and 4 are associated with the identical message type (C). However, Interaction 5 is associated with a different message type (D). Both message types (C) and (D) are defined in the same Hierarchical Message Definition.

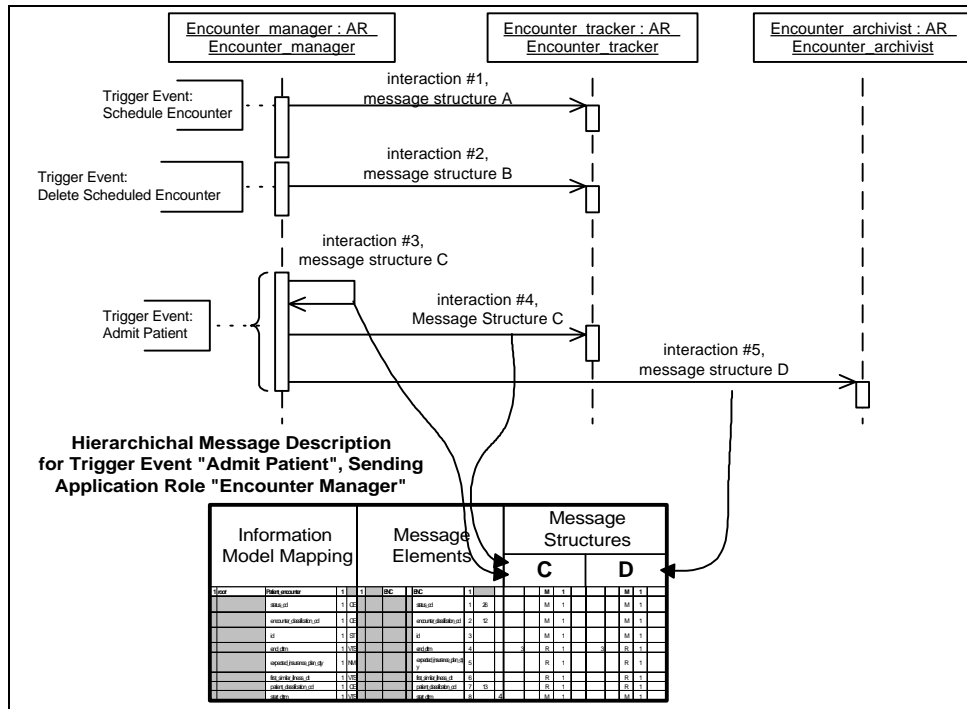


Figure 10-3. Message Types, Interactions, Trigger Events, and Application Roles

10.1.2 Introduction to Version 3 Message Instances

This section provides a brief introduction to version 3 messages by way of an example.

In order to create an example we need to assume an Implementation Technology Specification. This section assumes that the syntax will be XML, and that XML will be used in a certain style. The final Implementation Technology Specification may differ from that used here, but the example will serve to introduce some important concepts.

The following XML terms will be used without being defined here. The reader may refer to any of a multitude of texts for their definition.^{1,2}

- | | |
|--------------------------------|---------------------|
| Document Type Definition (DTD) | Element |
| Attribute | Tag |
| Content Model | empty content model |
| PCDATA | NMTOKEN, NMTOKENS |
| Document type (DOCTYPE) | |

Notes of the form <!-- 3 --> within the message are keyed to comments in this text that follow the example in Figure 10-4.

¹ Bray, Paoli, Sperberg-McQueen, *Extensible Markup Language (XML) 1.0*, W3C Recommendation 10-Feb-98, <http://www.w3c.org>.

² DuCharme, *XML: The Annotated Specification*, Prentice Hall, 1998.

```

<?xml version="1.0"?>
<!DOCTYPE Pt SYSTEM "admitexempl.dtd" [ ]>
<Pt>
  <!-- 1 -->
  <id V="12345" AA="100.12.92.81.5.7" APN="MRN"/> <!-- 2, 3 -->
  <status V="L" S="HL7003" R="3.0"/> <!-- 2 -->
  <isAroleOfPersnAsPt> <!-- 4, 5 -->
    <adminvGendr V="M" S="HL7001" R="3.0" PN="Male"/> <!-- 4 -->
    <brthDttm V="19790924162403-0800"/> <!-- 4, 6 -->
    <phon> <!-- 4 -->
      <_TEL ADR="tel:(358)555-1234" USE="PRN EMR"/> <!--7, 9 -->
    </phon> <!-- 4 -->
    <hasSetPrsnNameForPt> <!--4, 8 -->
      <_PrsnNameForPt>
        <nm>
          <G V="Irma" CLAS="R"/>
          <G V="Corine" CLAS="R"/>
          <F V="Jongeneel" CLAS="R M"/>
          <D V="-"/>
          <F V="de Haas" CLAS="R B"/>
        </nm>
        <purpse V="L" S="HL7005" R="3.0"/>
      </_PrsnNameForPt>
    </PtPrsnName> <!-- 4 -->
  </isAroleOfPersnAsPt> <!-- 2 -->
  <hasAprimryProvdriHCP> <!-- 2 -->
    <phon>
      <_TEL ADR="tel:(358)555-1234" USE="PRN EMR"/> <!-- 8 -->
      <_TEL ADR="tel:(358)555-4321" USE="FAX"/> <!-- 8 -->
    </phon>
    <isRoleOfPersnAsIHCP>
      <hasPrsnNameForIHCP>
        <nm>
          <G V="Bubba" CLAS="R"/>
          <G V="Corine" CLAS="R"/>
          <F V="Jongeneel" CLAS="R M"/>
          <D V="-"/>
          <F V="de Haas" CLAS="R B"/>
        </nm>
      </hasPrsnNameForIHCP>
    </isRoleOfPersnAsIHCP>
  </hasAprimryProvdriHCP>
  <isInvlvdInPtEncntr T="PtEncntr"> <!-- 2 -->
    <id V="12345A23" AA="100.12.92.81.5.7" APN="EID"/>
    <startDttm V="19990924162403-0800"/>
    <status V="L"/>
    <hasAsPartcptntSetEncntrPractnr>
      <_EncntrPractnr>
        <partcptntTyp V="ATT"/>
        <isPartcptntForIHCP>
          <phon>
            <_TEL ADR="tel:(358)555-1234" USE="PRN EMR"/>
          </phon>
          <hasPrsnNameForIHCP>
            <nm>
              <G V="Bubba" CLAS="R"/>
              <G V="Corine" CLAS="R"/>
              <F V="Jongeneel" CLAS="R M"/>
              <D V="-"/>
              <F V="de Haas" CLAS="R B"/>
            </nm>
          </hasPrsnNameForIHCP>
        </isPartcptntForIHCP>
      </_EncntrPractnr>
      <_EncntrPractnr>
        <partcptntType V="CONS"/>
        <isPartcptntForIHCP>
          <phon>
            <_TEL ADR="tel:(358)555-1234" USE="PRN EMR"/> <!-- 8 -->
          </phon>
          <hasPrsnNameForIHCP>
            <nm>
              <G V="Billy-Bob" CLAS="R"/>
              <F V="de Haas" CLAS="R B"/>
            </nm>
          </hasPrsnNameForIHCP>
        </isPartcptntForIHCP>
      </_EncntrPractnr>
    </hasAsPartcptntSetEncntrPractnr>
  </isInvlvdInPtEncntr> <!-- 2 -->
</Pt> <!-- 1 -->

```

Figure 10-4. Example Version 3 message in a possible XML style.

Notes for Figure 10-4.

- 1 The payload of the message is entirely within the document type, Pt (Patient). In HL7 terms, this is the largest **message element**.
- 2 In this example, where many optional parts of the message are not included, the Patient message element directly contains the following additional message elements.

```
Pt (Patient)
  id (id)
  status (status_cd)
  isAroleOfPersnAsPt(Person_as_patient)
  hasAprimryProvdrIHCP
    (has_a_primary_provider_Individual_healthcare_practitioner)
  isInvlvdInPtEncntr (is_involved_in_Patient_encounter)
```

These message elements have **short names** that are abbreviations for their full names. The full names seen so far are the names of classes, attributes, or associations of the RIM, or are constructed by combining the name of an association with the name of the distal class.

The message elements seen so far are represented by XML elements.

- 3 The id field, which is of the **instance identifier** data type, has three components in this example, value (V), assigning authority (AA), and assigning authority print name (APN). In HL7 terms these components are also message elements. These message elements are not represented with XML elements; they are represented with XML attributes.
- 4 Within the isAroleOfPersnAsPt (Person_as_patient) message element we see the following composites, which are also message elements:

```
  admnvGendr (administrative_geneder_cd)
  brthDttm (birth_dttm)
  phon (phon)
  hasSetPrsnNameForPt (has_Set_Person_name as_Patient)
```

- 5 Birth_dttm is of data type TS. This is a primitive data type, so there are no subcomponents. Its value is in the attribute named V.
- 6 Administrative_gender_cd is of type CV. Its subcomponents are shown as XML attributes.
- 7 Phon is of type Set(TEL); the data type permits more than one locator for telecommunications equipment. Sets and lists are usually represented as a message element that contains zero or more identical message elements of the underlying type (in this case TEL). The message element that represents the underlying type is always named `_XXX` where XXX is the name of the type.
- 8 Other examples of collections (sets or lists) include


```
status (status_cd), of type Set(CV)
hasSetPrsnNameForPt
  (has_Set_Person_name as_Patient), of type Set(Person_name)
etc.
```
- 9 Message elements of type TEL have two components, address, of type URL; and use; of type set(ST). The representation of this set is optimized in XML: there is no subordinate subelement name `_ST`. Instead, the set is represented using the XML NMTOKENS feature.

Altogether, the message is a hierarchy of message elements, each element having a name and type. The complete hierarchy is

Message element name	Message element type
Patient	Patient
id	instance identifier

Message element name	Message element type
value	ST
assigning authority	OID
assigning authority print name	ST
status_cd	set(CV)
_CV	CV
value	ST
coding system	ST
coding system version	ST
print name (optional message element missing in example)	ST
is_a_role_of_Person_as_Patient	Person_as_Patient
administrative_geneder_cd	CV
(to conserve space the subcomponents of the CV data type are not repeated, here or for other message elements of type CV)	(subcomponents are all of type ST)
birth_dttm	TS
phon	set(TEL)
_TEL	TEL
address	URI
use	Set(ST)
_ST (this HL7 message element type is represented by making the XML attributge for use be of XML attribute type NMTOKEN)	ST
PtPrsnName (has_Set_Person_name as_Patient)	Set(Person_name)
_PrsnName	Person_name
nm	PN
(various specialized name parts, such as given and family)	(ST)
purpose_cd	CV
has_a_primary_provider_Individual_healthcare_practitioner	Individual_healthcare_practitioner
phon	set(TEL)
_TEL	TEL
(to conserve space the subcomponents of the TEL data type are not repeated, here or for other message elements of type TEL)	
has_Person_name as_Individual_healthcare_practitioner	Person_name
nm	PN
(various specialized name parts, such as given and family)	(ST)
purpose_cd	CV
is_involved_in_Patient_encounter	Patient_encounter <i>or</i> Inpatient_encounter
id	instance identifier
status_cd	set(CV)
_CV	CV
start_dttm	TS
has_as_participant_Set_Encounter_practitioner	Set(Encounter_practitioner)
_Encounter_practitioner	Encounter_practitioner
participation_type_cd	CV
is_participant_for_Individual_healthcare_practitioner	Individual_healthcare_practitioner
phon	set(TEL)
_TEL	TEL
has_Person_name as_Individual_healthcare_practitioner	Person_name
nm	PN
(various specialized name parts, such as given and family)	(ST)
purpose_cd	CV

Figure 10-5. Hierarchy of message elements in the example message.

Note that the is_involved_in_Patient_encounter message element is shown as having one of two message element types. This is because the Inpatient_encounter class is a specialization of Patient_encounter. Some

message instances may transmit information about inpatient encounters, but the particular example shown here does not. So, in the example, `is_involved_in_Patient_encounter` actually has the type `Patient_encounter`.

Based on this example, we observe the following general characteristics of Version 3 messages:

- Every part of the message, from the entire message down to the smallest subcomponent of a subcomponent of a data type, is a message element.
- Every message element has a type.
- Every message element that is an attribute of a RIM has a type that is an HL7 data type.
- Every message element that represents a component of an HL7 data type also has a type that is an HL7 data type.
- Every message that represents a class has a data type that is based on that class or a Common Message Element Type.
- Every message that represents an association has a data type that is based on the distal class of the association or a Common Message Element Type.
- Every message element type is one of these four metatypes: primitive, composite, collection, or choice.
- Primitives have no subordinate components.
- Composites do have heterogeneous subordinate components, each with its own name.
- Collections have repetitions of a homogeneous message element. The name of the repeating message element is derived from its type.
- The previous statement notwithstanding, the XML representation of messages may not always create a separate message element for the item which is repeated in a collection. In some cases it may use the XML `NMTOKENS` attribute type to achieve the same goal.

10.1.2.1 HL7 Interversion Compatibility

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 type 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 type 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 type in use by a specific sender.

- 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 type may be declared obsolescent in one release, with a stated alternative message type. Both the obsolescent message type and its alternative can be used by all systems supporting that release.
 - c) The obsolescent message type may be declared obsolete and dropped when still another HL7 version is issued.
 - d) An obsolescent message type 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 types 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.
- D. There are no restrictions on making changes to the information model if those changes do not impact the message types that were described in a prior version of the Standard.

10.2 Work Products

This section will define the Hierarchical Message Definition and the intermediate work products in detail. Many of the illustrations will be drawn from the HL7 Example Model. Figure 10-6 is its Message Information Model.

10.2.1 Message Information Model

The Message Information Model (MIM) is an information model as described in Chapter 6. It contains classes drawn from the Reference Information Model (RIM). It also contains selected relationships among classes. The subset of the RIM should be adequate to describe the contents of a logically related group of message types. Figure 10-6 is an example of a MIM that is used in this chapter as a basis for many other examples.

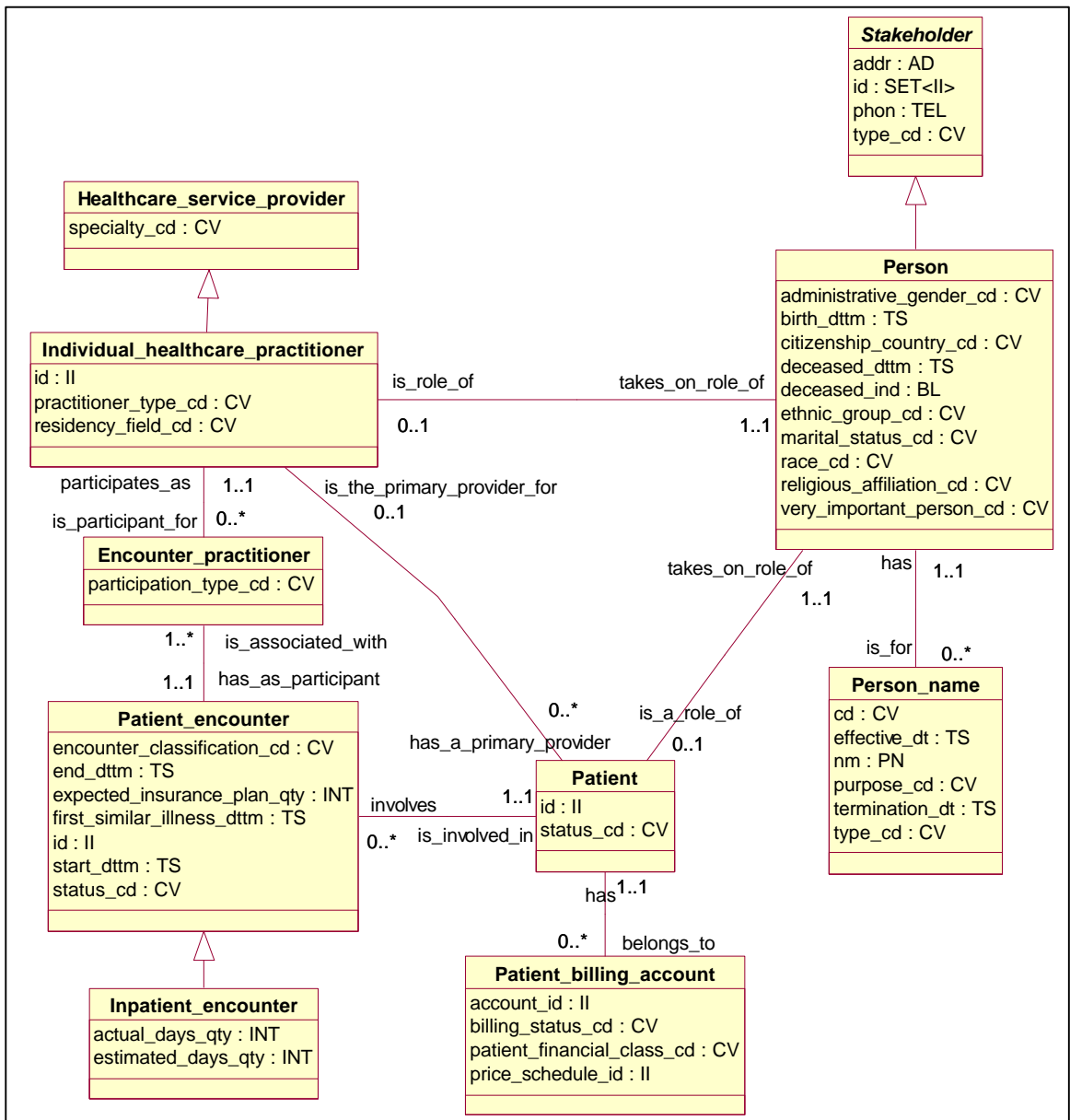


Figure 10-6. Message Information Model from the “Example Model”

10.2.1.1 Recursion

In certain Message Information Models a relationship exists in which an association leads from a class directly or indirectly back to that class. This is a recursive relationship.

Figure 10-7 is an example of a Refined Message Information Model with recursive relationships. The association in the lower left corner,

Organization :: is_a_subdivision_of (0..1) Organization:: has_as_a_subdivision (0..*),

creates the possibility that different message instances may contain a different number of message elements that describe subdivisions of an organization. Depending on how the message is defined it could start with

the most inclusive location (perhaps a specific practice) and follow the `is_a_subdivision_of` association to a more inclusive unit (perhaps the department), follow it again to a still more inclusive location (perhaps a medical center), and follow it again to another more inclusive organizational unit (the health system.) Another order message may have fewer levels, because it is about an organization that consists entirely of a single practice. When the committee is designing the message type, there is no way to predict how many such objects will occur in an instance. Figure 10-8 is snippet of an instance example drawn from Figure 10-7.

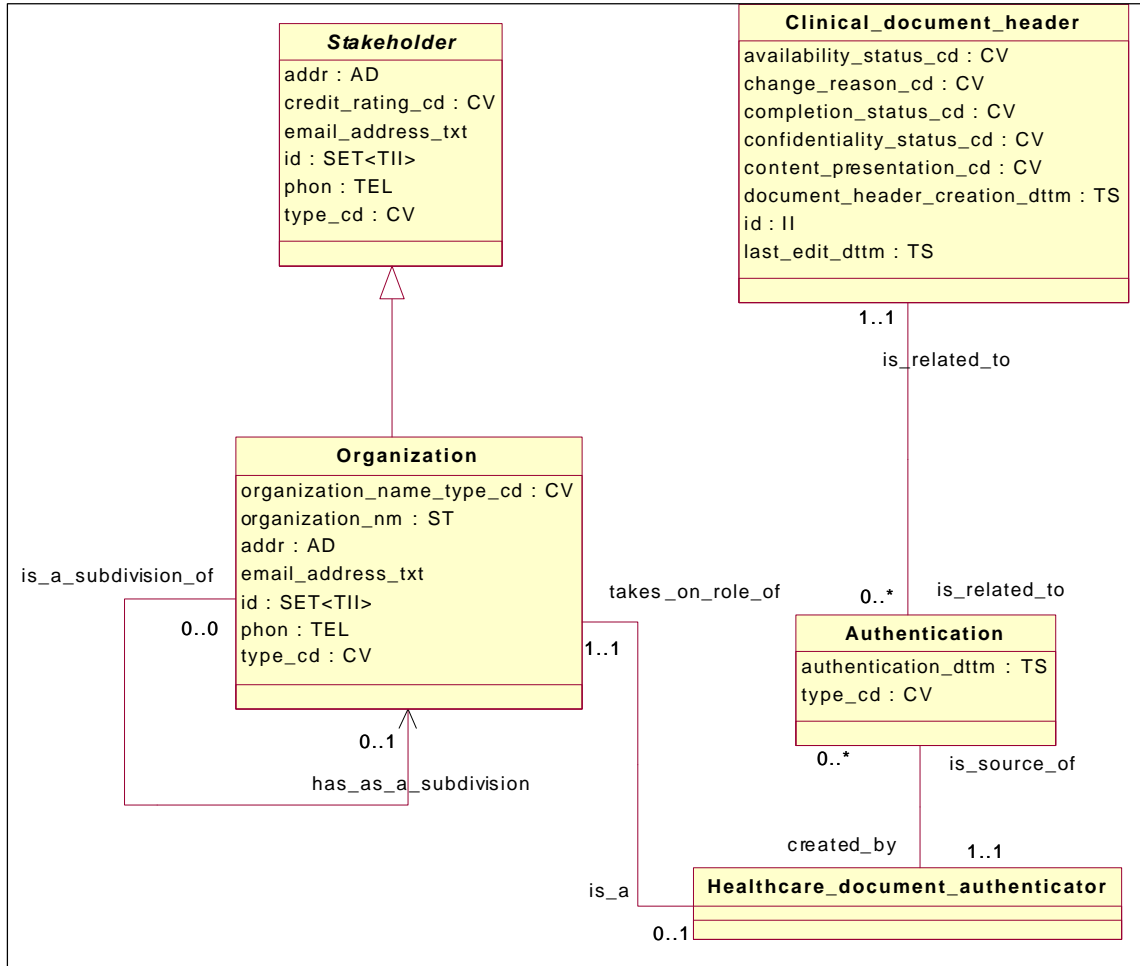


Figure 10-7. Recursion in a Refined Message Information Model.

```

<Healthcare_document_authenticator>
  <is_an_Organization>
    <organization_nm V="leftPinkyclinic"/>
    <is_a_subdivisoon_of_Organization>
      <organization_nm V="leftHandclinic"/>
      <is_a_subdivisoon_of_Organization>
        <organization_nm V="Orthopedics"/>
        <is_a_subdivisoon_of_Organization>
          <organization_nm V="Surgery"/>
        </is_a_subdivisoon_of_Organization>
      </is_a_subdivisoon_of_Organization>
    </is_a_subdivisoon_of_Organization>
  </is_an_Organization>
</Healthcare_document_authenticator>

```

Figure 10-8. Instance example of recursion.

The technical committee must be aware of special message design considerations for recursive messages. These are discussed later in this chapter.

Some recursive relationships are represented by loops of associations that go through intervening class. Figure 10-9 is an example where the loop that creates recursion passes from Master_service through Master_service_relationship and back to Master_service.

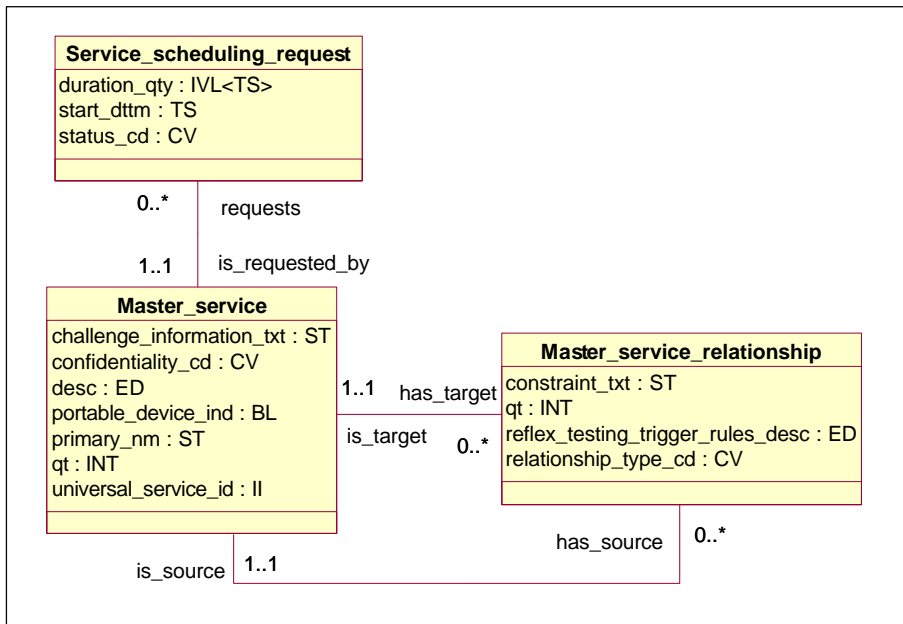


Figure 10-9. Recursion through an intervening class.

10.2.2 Refined Message Information Model

The Refined Message Information Model (R-MIM) is an intermediate step towards building the Hierarchical Message Diagram. There are two distinct representations:

- graphical, a diagram in the Unified Modeling Language
- tabular, a more detailed description.

10.2.2.1 Graphical Refined Information Model

Figure 10-7 is an example of the graphical format; it corresponds to the Message Information Model in Figure 10-6. The Person class has been replaced with a pair of classes, Person_as_patient and Person_as_IHCP (individual health care provider). There has been a similar substitution for Person_name. The newly created classes are called **clone classes** and the classes that they substitute for are called **base classes**.

In this example the Stakeholder class is missing, and the attributes of interest (e.g., phon) have been included as if they were a part of the Person_as_... classes. This is consistent with the specialization relationship between Stakeholder and Person. This Refined Message Information Model has been constructed to make it clear that no message formats derived from this Refined Message Information Model will have an object that represents the Stakeholder class by itself.

Another characteristic of a Refined Message Information Model is that one or a few of the classes can be assigned specific instance cardinality. This example states that all message formats derived from the Refined Message Information Model will have a single patient. Our current software tooling does not call out class cardinalities on an information model diagram, so we use a comment to represent this fact.

Conceptually, the Refined Message Information Model has yet another difference from the Message Information Model. It uses the UML concept of navigability to show that some of the associations will only be traversed in a specific direction. For example, in Figure 10-9, the arrow head at the Individual_healthcare_practitioner end of

Patient has_a_primary_provider (0..*) Individual_healthcare_practitioner

indicates that no message will be constructed that starts with a practitioner and selects the patient for which the practitioner is the primary provider.

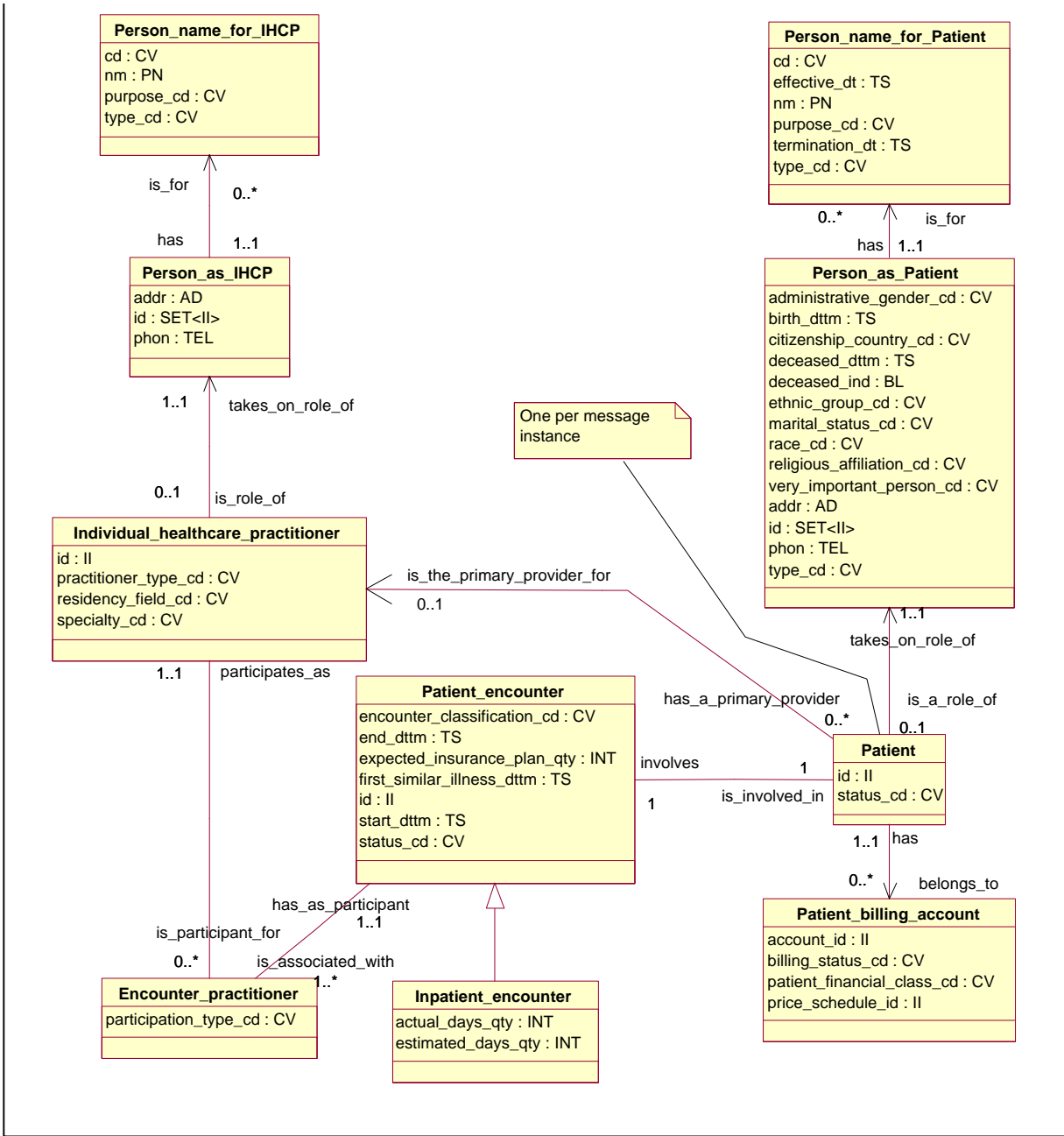


Figure 10-10. Refined Message Information Model, Graphical Format.

10.2.2.2 Tabular Refined Message Information Model

Exhibit 10-1, at the end of this chapter, is an example of the tabular version of the Refined Message Information Model shown in. It has two major sections:

- The **information model** section, on the left, lists the information model entities (classes, associations, and attributes), one per row.
- The **constraints and defaults** section states specific constraints on the information model entities that will be applied to all message formats that are in Hierarchical Message Definitions derived from the Refined Message Information Model. Some of the constraints are also a part of the Message Information Model, although they may be tightened in the Refined Message Information Model. Many of the constraints are not UML constructs; they apply specifically to the HL7 messages defined based on the Refined Message Information Model.

Some column headings in the *constraints and defaults* section contain the number sign. This indicates that the value for a given cell can be stated in a separate list of constraints, defaults and comments that accompanies the Refined Message Information Model and its associated Hierarchical Message Definitions.

The columns that compose these sections are defined below.

10.2.2.2.1 Information Model Section

Row Number. This column is the row number in the spreadsheet that is used to generate the tabular format. It is useful during discussions about the spreadsheet.

Row type. This column identifies the kind of information model entity that the row represents. The possible values are:

rmim	always the first row of the table, identifies the particular Refined Message Information Model in the nomenclature of the HL7 Repository
class	identifies a “class” in the Refined Message Information Model
attr	identifies an attribute of the “class” that is most directly above this row
assoc	Identifies an association leading from the class that is most directly above this row
stc	subtype constraint: this row corresponds to a subcomponent of the row above; it would not normally be included in an Refined Message Information Model, but it is included in order to be able to state a constraint on the subtype. This is explained in 10.2.2.4.

It is natural to believe that every row identified with *assoc* has a corresponding *assoc* row under the row that represents the distal class of the association. However, this is not always true. Where the committee has determined that the association will only be traversed in one direction the corresponding *assoc* row will not be present.

Class or Property. This column contains the information model name of the entity that is described by the row. The term *property* is used to lump together attributes and associations. This is because they will be treated in a very similar manner in the Hierarchical Message Definition. In an *rmim* row, this column contains the identifier of the parent Message Information Model.

Short name. This is an abbreviated form of the name of the information model entity that will be used to tag the corresponding message elements in ITS-specific syntaxes that use tags. In an *rmim* row, this column contains the name of the parent Message Information Model.

Inherited from. This is the class where the property (attribute or association) appears in the Message Information Model. The column is only filled in when the property appears in a different class in the Message Information Model. See 10.2.2.3. In an *rmim* row, this column contains the name of the Refined Message Information Model.

Message Element Type. For attributes, this is the data type of the attribute. For associations, this is the name of the distal class.

10.2.2.2.2 Constraints and Defaults Section

This section contains the following columns.

Cardinality. This column contains the minimum and maximum number of times that the message element that corresponds to the entity defined by the row can appear in any message instance that is based on this Refined Message Information Model. For classes, this column is frequently left blank. The cardinality of

most classes can be inferred if the cardinality of some classes are known. Most frequently this column is used to state that a central, or root class for messages derived from the RIM will be present exactly once, or at least once. In the example, the Patient and Patient_encounter classes are so designated.

Domain Specification. This column contains a specification of the domain specification for message elements that contain codes. The syntax of such a specification is defined in Chapter 7.

Coding Strength. This column contains either **CWE** (coded with exceptions) or **CNE** (coded, no exceptions). It is blank in rows that do not describe a message element that contains a code.

Mandatory. This column contains an **M** (mandatory) or is empty (not mandatory). If a row is described as mandatory, all message elements that are derived from this information model entity in any message instance must contain a non-null value, or they must have a default that is not null.

Constraint/Note. This column may contain a number that indicates an item in the associated list of constraints and defaults. That list item describes a constraint that applies to all message instances derived from this Refined Message Information Model. Example might be “at least one instance of this message element must identify the attending physician” or “this message element is only present when the *deceased* status code is present for the class.”

Default Value. This column may contain a value that the receiver should use when it receives a message instance that does not have the message element derived from this information model entity. If the column is left empty for a specific row, the “default default” is the simple form of null (NI).

Default Update Mode. This column may contain a value that defined the update mode that will be used for a message element when no update mode is explicitly sent in the message. When the column is left empty for a cell, “default default” update mode is **R** (replace).

Update mode set. This column may contain a list of values that may be sent in message instances to alter the receiver’s processing from the default update mode. If the column is left blank, the only permitted value is the default update mode.

The values that can appear are:

R	replace
D	Delete
I	Ignore
K	Key, this message element is used as a key to s collection of message elements
V	Verify, this message element must match a value already in the receiving systems database in order to process the message
ESA	Edit set add, add the message element to the collection of items on the receiving system that correspond to the message element
ESD	Edit set delete, delete the item on the receiving system that corresponds to this message element
ESC	Edit set change, change the item on the receiving system that corresponds to this message element; do not process if a matching element does not exist
ESAC	Edit set change, change the item on the receiving system that corresponds to this message element; if a matching element does not exist, add a new one created with the values in the message

Conformance Flag. A cell in this column may contain an **R** (required) or be empty. Its interpretation is discussed in Chapter 9.

10.2.2.3 Treatment of Inheritance

The Refined Message Information Model has two alternate strategies for dealing with specialization.

10.2.2.3.1 Subsumption of Inherited Properties

When none of the message instances defined on the Refined Message Information Model will include the general class without one of its specializations, the attributes and associations of the specialization may be subsumed into the specializations, and the general class removed from the R-MIM. Note that in the example message the Stakeholder class does not appear in the R-MIM. Its properties have been subsumed into the various clones of Person. This is not required.

The rows that represent the subsumed attributes and associations contain the name of the class where they originated in the **Inherited from** column of the Tabular Refined Message Information Model.

10.2.2.3.2 Maintaining a Separate Generalization.

When some message instances will contain only an object from the general class, while other instances will contain an object from one of its specializations, the generalization and the specializations must be maintained in the Refined Message Information Model. In this case each specialization must contain all the subsumed properties of the general class, with the name of the general class in the Inherited From column.

10.2.2.4 Subtype Constraints

Often, a technical committee will want to state a constraint on one of the components of the data type of an attribute row in the Refined Message Information Model or the Hierarchical Message Definition. These message elements are not usually shown in the R-MIM or the HMD, but the committee has the option of adding one or more *stc* rows to express the constraints. Each *stc* row represents a component of the message element type of whatever in the row above. In the R-MIM, the *stc* row can only appear below an *attr* row or another *stc* row. The hypothetical example below is a snippet of an R-MIM. The committee has decided that the default currency for Patient_account.total_adjustment_qty should be US Dollars. To specify this they add an *stc* row for the currency_unit component of the MO datatype, and specify its default value.

row type	msg element name	msg element type	default
class	Patient_billing_account	--	
attr	total_adjustment_qty	MO	
stc	currency_unit		USD

10.2.3 Hierarchical Message Definition

The Hierarchical Message Definition consists of four major sections:

- **The Information Model Mapping.** The columns that are in this section describe classes and attributes of the Refined Message Information Model. They are organized in a sequence that describes a “walk” from class to class of the Refined Message Information Model, where the choice of the next class is determined by the associations leaving the current class. This walk is described in 10.3.3.

- **The Message Elements.** The columns that are in this section describe the message elements. They are row-wise related to classes and attributes of the information model, as described in the Information Model mapping section. The message elements compose a hierarchy. This hierarchy is illustrated by indentation in the column *Message Element Name*.
- **General constraints and defaults.** The columns that are in this section describe specific constraints and defaults for the message element defined in the row. The columns are the same as the corresponding section of the Refined Message Information Model. The values in the columns may be the same or may represent a more restrictive constraint.
- **Message type definitions (repeating).** This section repeats, once for each message type defined in the Hierarchical Message Definition. The columns that are in this section describe one or more message types. Each message type is identified with one or more interactions in the interaction model. The column headings for each message type are the same as the column headings for the general constraints. If the cells are left empty, the values will be the same as in the general constraints section. When filled in, they, may represent more restrictive constraints, or may indicate that the specific message element is not a part of the specific message type being defined in the section.

A complete example of a Hierarchical Message Definition drawn from the example model is in Exhibit 10-2, following this chapter.

10.2.3.1 Information Model Mapping

This section describes an entity in the Refined Message Information Model that is the basis for a message element. The columns are:

Row Number. As defined in 10.2.2.2.1.

Row Type. This column is also as defined in 10.2.2.2.1. The complete set of row types for the Hierarchical Message Definition is shown below.

hmd	always the first row of the table, identifies the particular Hierarchical Message Definition in the nomenclature of the HL7 Repository.
class	identifies “class” in the Refined Message Information Model. There is only one class entry in a Hierarchical Message Definition. This is the root class for the message.
assoc	Identifies an association leading from the class that is most directly above this row
attr	identifies a message element that represents a an attribute of a “class” in the Refined Message Information Model
item	identifies a message element that represents one of whatever is repeated in a collection
stc	subtype constraint: this row corresponds to a subcomponent of the row above; it would not normally be included in an HMD, but it is included in order to be able to state a constraint on the subtype. This is explained in 10.2.2.4.

Class or Property of Class. This row is as defined in 10.2.2.2.1. A cell in this column is empty for a row of type *item*. For the *hmd* row, this cell is the designation of the Message Information Model in the nomenclature of the HL7 repository. For an *item*, the name is empty. For an *stc* row, the name will be populated if the subtype corresponds to an entity of the information model. In an *hmd* row, this column contains the identifier of the parent Message Information Model.

Rim Source Class. This column gives the class that is or contains the definition of the information model entity named in *Class or Property of Class*. This is the source as it is defined in the Message Information Model (and hence the Reference Information Model). It may be different than the class associated with the information model entity in the Refined Message Information Model for several reasons:

- The entity may be a property that has been subsumed from a generalization class that is not itself shown in the Refined Information Model.
- The entity may be in a clone class.

For the *hmd* row, this cell is the name of the Message Information Model in the nomenclature of the HL7 repository.

10.2.3.2 Message Elements Section

This section defines the elements of messages. Its columns are described below:

Message Element Name. This is the name of the message element. It is derived from the name in the information model using row-specific rules:

hmd	This is the identifier for the parent Refined Message Information Model
class	This is the same as the name of the class in the Refined Message Information Model in the <i>Class or Property of Class</i> column.
attr	This is the same as the name of the attribute in the Refined Message Information Model in the <i>Class or Property of Class</i> column. However, if the item has a maximum cardinality greater than one, the name is constructed by prepending <i>Set</i> or <i>List</i> to the name of the attribute and an item row is included immediately underneath.
assoc	This name is constructed by combining the name of the association with the name of the distal class of the association.
item	See 10.3.3.2, <i>Collections: Dealing with Cardinalities Greater than One</i>
stc	The name of the message element that is the subcomponent described by this row.

Message Element Short Name. This is an abbreviation of the name in the *Message Element Name* column. In an *hmd* row this column contains the repository identifier of the Hierarchical Message Definition.

In Message Element Type. Every message element type except the one defined in a *class* row is a subelement of a larger composite message element type. This column contains the name of the containing message element type. In an *hmd* row this column contains the name of the Hierarchical Message Definition.

Source Of Message Element Type. This column states the source of the type that is in the *Of Message Element Type* column. The possible entries are shown below.

- N** New data type. This row starts the definition of a new message element type. The subordinate rows beneath it compose the definition of the data type. Each of these subordinate rows has the name of the message element type being defined in the *In Message Element Type* column. That name will be the same one that is in the *Of Message Element Type* of this row.

- D** This message element type is an HL7 data type.
- C** This message element type is an HL7 common message element type.
- U** This message element type was previously defined in this HMD and is being reused
- R** This row represents the recursive reuse of the message element type within which is appears. See 10.3.3.3.

Of Message Element Type. This column states the type of the message element defined in a row. Short names are used.

class	The cell contains the short name of the class.
attr	The cell contains the short name of the data type of the attribute.
assoc	This name is the short name of the distal class of the association. However, if the association has a maximum cardinality greater than one, the name is preceded by <i>Set</i> < or <i>List</i> < and followed by >. Multiple class names may appear in this cell, as described in 10.2.3.5.
item	See 10.3.3.2, <i>Collections: Dealing with Cardinalities Greater than One</i>
stc	The type of the message element that is the subcomponent described by this row. See 10.2.3.6.

10.2.3.3 General Constraints

The column headings of this section are the same as those defined in the Refined Message Information Model. The values are copied from the Refined Message Information Model or replaced with values that represent tighter constraints.

10.2.3.4 Message type definitions (multiple right “sides”)

The column headings of the *General Constraints* section are repeated one or more times. Each such repetition is associated with the definition of a different message type. The values are either empty or contain a value that represents a tighter constraint.

The *Mandatory* column allows a special value that is not acceptable in the *General Constraints Section*. The entry **NP** (not permitted) means the message element defined by the row is not included in the message definition. (By inference, all components of the message element type are similarly forbidden.)

10.2.3.5 Inheritance that Leads to Instance Choices

When there is inheritance in the Message Information Model the technical committee has several choices for how to treat it.

10.2.3.5.1 No instance-specific choice.

If the committee decides that one of the specializations of a general class will appear in every message instance then it will usually eliminate the general class and subsume the properties that will be in the message to the specialization class of the Refined-MIM. The resulting message element contains the combination of the properties of the specialization and the subsumed properties of the generalization. This is exactly consistent with the object-oriented concept of specialization. In the example, this treatment was used to remove the Stakeholder and Healthcare_service_provider general classes from the Refined Message Information Model and, therefore, the Hierarchical Message Definition.

10.2.3.5.2 Instance-specific choice.

If the committee decides that different instances can have different specializations, or if some instances can have the general class without any specializations, the message type contains a choice. In the example, some instances may carry the object `Patient_encounter`, while others may carry the object `Inpatient_encounter`. This choice is represented in the Hierarchical Message Definition with a special value in the Of Message Element Type column for the association that leads to the general class. When there is no choice, this column contains the name of the distal class that will appear in the message. When there is a choice, the distal class can be different in different instances. In this case, the cell contains a list of the classes that may appear in instances, separated by the vertical bar (reminiscent of the word “or”). In the example HMD the contents of this cell for the row that corresponds to the `is_involved_in` association contains “PtEnctr | InptEnctr”. When this happens, the “in message element type” column contains an asterisk instead of the name of the message element type.

10.2.3.6 Reused Message Element Types

In Exhibit 10-2 the message element type IHCP is defined by the rows under the row that represents the association

```
Patient has_a_primary_provider Individual_healthcare_practitioner
```

There is another association that is also represented by a message element of type IHCP,

```
Encounter_practitioner is_participant_for Individual_healthcare_practitioner.
```

Rather than repeat the rows under that message element, the technical committee puts an R in the Source of Message Element column. This is illustrated in Exhibit 10-2.

10.2.3.7 Subtype Constraints

At *stc* row is used in the Hierarchical Message Definition in the same way as was described for the Refined Message Information Model in 10.2.2.4. It can be used to establish constraints for the components of data types, common message element types or types that have been defined in the same Hierarchical Message Definition and are being reused.

10.2.4 Common Message Element Type Definition

Common Message Element Types are represented in a format that looks like the Information Model Mapping and Message Element sections of an Hierarchical Message Definition. In fact, they contain exactly the rows that will appear in the Hierarchical Message Definition to represent the common message element.

10.3 Procedures

10.3.1 Create the Message Information Model

Message Information Models are defined in Chapter 6. This section provides supplementary information to support their creation. *Figure 10-6* is an example of a MIM that is used in this chapter as a basis for many other examples.

In choosing the classes for the MIM the Technical Committee will look ahead to some set of message types that it will be defining in one or several Hierarchical Message Definitions.

At its smallest, a MIM must contain all the classes that will be referred to or serve as a base class or a clone in a single Refined Message Information Mod. In turn, the R-MIM must contain all the classes necessary to build at least one Hierarchical Message Definition. However, there is benefit to constructing a MIM that covers several sets of related message types. Users of the HL7 specifications will better understand the data relationships among the messages if they are drawn from a common MIM. On the other hand, the MIM is less comprehensible if it cannot be displayed on a one or two pages of a document. A good MIM is a balance between these competing virtues.

It is difficult, however, to know exactly the classes that will be required until the analysis that is part of subsequent steps has been performed. Technical Committees are encouraged to quickly build a draft MIM without pondering its contents in great detail. After it has built a number of related Hierarchical Message Definitions, it should review the MIM and create a publication version that matches the contents of the Hierarchical Message Definitions.

Indeed, new software tools for defining HL7 messages may allow some committees to skip the step of creating a Message Information Model and build Refined Message Information Models directly from the Reference Information Model. As a final step before publication they can prepare Message Information Models that contain the minimum number of classes, attributes, and associations to document a set of Hierarchical Message Definitions.

10.3.2 Create the Refined Message Information Model

Inputs: Message Information Model or Reference Information Model, Use Case Model, Interaction Model.

Procedure.

The following discussion uses the example in Message Information Model in *Figure 10-6*.

10.3.2.1 Select a group of interactions to specify

The Hierarchical Message Definitions that the committee will ultimately create must include, at a minimum, message types for all of the interactions that are associated with a single sending Application Role and a single Trigger Event or receiver responsibility. The Refined Message Information Model is the basis for all the messages in at least one Hierarchical Message Definition, so it must include the object views necessary to cover the message types for all the interactions associated with a single trigger event. The interactions that will be used as an example are shown in *Figure 10-11*. Interactions numbered three and four will be associated with message type C00XMM011, and interaction number five will be associated with message type C00XMM013.

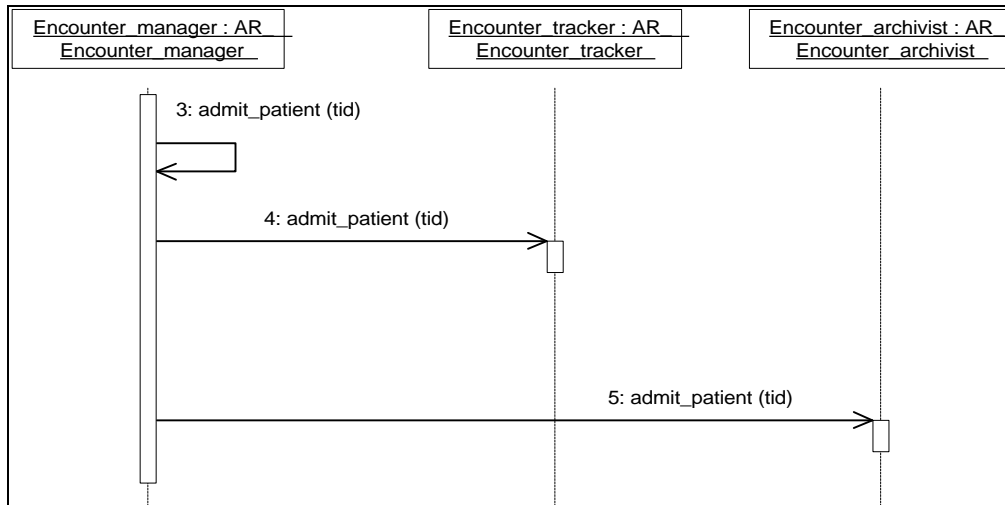


Figure 10-11. Interactions for the Example HMD

Make an “information shopping list”. The Technical Committee should review the use cases to make an informal list of the information that must be in the message. This list should be built *before* examining the Message Information Model to look for the information. If the Technical Committee constructed the Message Information Model from a close reading of the use cases, then it should review the notes from that process to build the shopping list. Figure 10-12 is an example of such a list.

The message types will need to contain
 encounter information
 patient Information
 admission information (for inpatient)
 billing account information (if available)
 information on various physicians, esp. the attending physician.

Figure 10-12. “Shopping List” of Information for a Hierarchical Message Definition

This list is very general and does not use the class names from the Message Information Model. It is built to prepare for a “walk” from class to class in the Reference Information Model, following its relationships. What is important is that it list contingencies and pay special attention to singulars and plurals. An example of a contingency is “admission information (for inpatient).” An example of plurality is “information on various physicians.”

10.3.2.2 Find Common Message Elements

It has been observed that a few months in the laboratory can save a few hours in the library. Similarly, a Technical Committee can devote whole meetings to debating the nuances of some common area in healthcare or they can adopt common message elements defined by the committees that have stewardship for the relevant classes. The Technical Committee should review its information shopping list against the list of available common message elements. Where a common message element is available it will be identified with a class in the Reference Information Model that is its root class and a message element name. For example, a common message element might be identified with the class `Individual_healthcare_practitioner` when used so that the practitioner can be identified by the sender and receiver using a common identification number.

Where a common message element will be used in creating a Refined Message Information Model, it is inserted where one could otherwise put an object view of the class that is at its root. The common message element for `Individual_healthcare_practitioner` might be inserted wherever `Individual_healthcare_practitioner` would appear. However, the common message element may not always

appear. For example, the Technical Committee could decide that its needs for referring physicians may not fit the `Individual_healthcare_practitioner` common message element, and would then use a different common message element or simply define one or more special message elements.

10.3.2.3 Gather Relevant Classes and Connecting Classes and Associations

The technical committee will identify certain classes that contain the information that is on their “shopping list”. In the final Refined Message Information Model all classes must interconnect. That is to say there cannot be two classes A and B such that it is not possible to find a path from A to B through a single association or through intervening class by associations. In order to include the necessary classes the committee will frequently have to add classes to complete the paths. Frequently there will be multiple alternate paths that may be used. The committee selects the correct path based on an understanding of the definitions of the classes and associations. For example, in the example Message Information Model it is possible to get from `Patient_encounter` to `Individual_healthcare_practitioner` by two paths. One path yields the practitioner who is the primary care practitioner of the patient, even though this practitioner may not have a direct involvement in the encounter. The other path yields a set of practitioners who do have a role in the encounter, along with an attribute that describes their role. In this example, the committee decided that both semantics were required so the intervening class, `Encounter_practitioner` was chosen, but the `has_a_primary_provider` association was also included.

10.3.2.4 Create Clone Classes

Based on the shopping list the technical committee may observe that the same class is used in two different semantic contexts. For example the `Person` class is used to describe a person who is a patient and also to describe people who are individual healthcare practitioners. Where the information that will be sent is different for the semantic contexts, the technical committee should replace the base class with two or more clone classes, named to show the semantic distinction.

10.3.2.5 Drop Unneeded General Classes

In cases where a general class will never be created in a message instance without one of its specialization, the committee should usually drop the general class subsume its attributes and associations into the one or more of the specializations. This is always permitted for abstract classes, but it can also be true for instantiable classes, if the semantics of the message instances do not support the general class standing alone. If the trigger events chosen for a set of messages only referred to inpatient encounters, the technical committee could alter our example to drop `Patient_encounter` and subsume its attributes and associations into `Inpatient_encounter`.

This subsumption is permitted, but not required. Its value is to simplify the information model.

10.3.2.6 Identify One-Way Associations

Where it is clear that an association will only be used in one direction, the technical committee may designate this. The Graphical Refined Message Information Model shows this with the arrow that on the association line that represents icons. For example, in Figure 10-9 the association

`Patient has_a_primary_provider Individual_healthcare_practitioner`

is navigable from `Patient` to `Individual_healthcare_practitioner`, but not in the reverse direction.

The Tabular Refined Message Information Model represents this by including the association under the class from which navigation is permitted, but removing it from the class from which navigation is not permitted.

Technical committees recognize one-way associations by an understanding of the use cases, interactions, and semantics. For example the navigability restriction on

Patient has_a_primary_provider Individual_healthcare_practitioner

corresponds to the notion that the message is about the characteristics of a patient, rather than about the patients for which a practitioner is the primary provider. This information is very useful when constructing the Hierarchical Message Definition.

10.3.2.7 Assign Constraints and Defaults

Based on the use cases and interactions, the committee may reduce cardinalities, establish a restrictive domain or coding strength for coded values, establish a default value or describe the update paradigm. Some examples of constraints in the sample R-MIM include tightening the Patient :: is_involved_in :: Patient_encounter association to 1..1, requiring that there be at least one instance of Encounter_practitioner with the value for an attending physician in participation_type_cd and stating that an individual healthcare practitioner will only have one name in the message.

When the committee does this in the Refined Message Information Model it is giving the users guidance that all the message derived from the R-MIM will be consistent with these constraints.

10.3.3 Build the Hierarchical Message Definition

To build the HMD, the technical committee copies classes from the Refined Message Information Model to the HMD in a certain order. The order defines a hierarchy of message elements. The first class copied appears at the top of the HMD. It defines the root message element, which contains all the subordinate message elements. The second class is chosen by traversing an association from the first, and the next by traversing an association from the second, etc.

Once the committee has defined all the message elements, it defines actual message formats by adding additional constraints, if any, to the multiple right sides of the Hierarchical Message Definition.

10.3.3.1 The Graph Walk

This section defines the process of traversing classes in the Refined Message Information Model to add rows to the Hierarchical Message Definition

10.3.3.1.1 Choose the Root Class

The root class for the message is the subject class for the relevant use cases, or another class that has a mandatory relationship to the root class in the direction from the subject class to the root class. In other words, if an instance of the subject class is in the message, the relationships indicate that there must be an instance of the root class.

Among all those classes that are related to the subject class in this manner, the root class should have the lowest cardinality. Where several classes meet the requirements, messages designed using any of them as the root will transmit the same information.

Very frequently, the root class is Patient.

10.3.3.1.2 Build the Hierarchical Message Definition

This part of building an HMD is really a pair of steps based on the “currently selected class.” Initially, the root class is selected. The steps are:

Step 1: Copy a class from the Refined Message Information Model to the Hierarchical Message Definition.

Step 2: Select another class that is related to the currently selected class through a relationship.

As the Technical Committee repeats these steps, it will “walk” from class to class of the Refined Message Information Model, making its steps along the lines that represent associations. In order to keep its bearings it will maintain a list of the classes that have been selected. As it takes a step to a new class it will add the class name to the list. At certain times, as instructed in the steps below, it will remove the most recent item on the list. This kind of list is often called a LIFO list (for an accounting method for dealing with inventory, last-in, first-out.) It is also called a pushdown stack, a metaphoric reference to the spring-loaded plate carriers used in institutional dining halls, where the new plates added to the top of the stack push down the earlier plates, so the newest plate is taken off the stack first.

Step 1: Copy a class to the HMD and the LIFO.

- A. Copy the class to the HMD. Include all attributes and associations that will or may appear in the message. Conventionally, attributes should be placed before the associations, but this is not a requirement.
- B. Place the selected class on the top of the LIFO list.

Step 2: Find the next class.

The following rules provide guidance regarding the choice of the association to use to step from the current class to the next in the “walk.” Once an association has been used to step from the current class to a new class, do not re-use it unless the current class itself has been reached again by a different association. All selections of associations must be consistent with the intended semantics of the message. This is an important judgment that is made using domain experience and common sense.

The following rules must be applied in the order they are listed. As soon one of them is reached that is applicable, select the class it specifies and return to step 1, above.

- A. **Gen-to-Spec.** If the current class is a generalization, and if it at least one of its specializations contains information that will be in the message format, or is on a path of associations toward a class that contains such information, then choose the specialized class.
- B. **Intimate Mandatory, Singular.** If a (1,1) association from the current class is navigable to a class that seems to add information about the current class, then use it next.
- C. **Other Mandatory, Singular.** Pick any (1,1) association that is navigable to needed information.
- D. **Singular.** Pick any navigable association that is (0,1) and leads to needed information.
- E. **Other.** Pick any navigable association that leads to needed information.
- F. **None.** If none of the above rules apply, cross the current class off the LIFO list. Take the prior class on the LIFO as the “current” class and immediately repeat steps A-G. If you have crossed the last class of the LIFO list, you have finished the process of selecting the classes and associations for the Hierarchical Message Definition.

The sequence of Rules (C) – (F) leads to message types that seem more coherent to people. If there is doubt or disagreement about which association to pick based on this rule, the committee may ignore rules (C) – (F) completely and pick any association. The message will have the same information content.

10.3.3.2 Collections: Dealing with Cardinalities Greater than One

When associations have cardinalities greater than one, and when the committee does not constrain the association cardinality to one in the Refined Message Information Model or the Hierarchical Message Definition, the message element that represents the association is a set or a list. In our example, the association Patient_encounter :: has as participant (1..*) Encounter_practitioner represents a set of Encounter_practitioner objects. In the HMD we create two distinct message elements. One of them is singular and represents the set as an entity. It contains a second message element that represents that which may be repeated in the set. The table below is a snippet of an HMD that shows this technique:

- the type of the set is “Set < the-type-of-the-repeated-item>”
- if the collection is a list than the type of the list is “List < the-type-of-the-repeated-item>”
- the newly created row has the row type **item**
- the message element has a name that is constructed by prepending an underling character on the name of the message element type of the row.

row type	msg element name	in msg element type	source of msg element type	of msg element type
assoc	is_involved_in_Patient_encounter	Patient	New type	Patient_encounter
attr	encounter_classification_cd	Patient_encounter	Data type	CV
	...			
assoc	has_as_participant_Encounter_practitioner	Patient_encounter	New Type	Set <Encounter_practitioner>
item	_Encounter_practitioner	Set <Encounter_practitioner>	New type	Encounter_practitioner
attr	participation_type_cd	Encounter_practitioner	Data type	CV
	...			

10.3.3.3 Recursion

In step 2, above, the committee may return to a class that has been visited before, and want the system that creates a message instance to loop back adding message elements to the message. For example, in the example in Figure 10-7, the committee may want to permit messages that repeat the organization class an arbitrary number of times in order to describe the organizational hierarchy of the healthcare document authenticator.

When this happens the committee will, at step 2, follow the is_a_subdivision_of association and would, according to the rules, place another copy of Organization on the LIFO.

The HMD so constructed might have the message elements shown below. (For clarity, the example has only a few attributes.) The special value **R** in **source of message element type** indicates that the definition is reused recursively. Rows that close the loop to create recursion should contain a constraint or comment that describes a stopping rule for building message instances. In some cases the committee may decide that this is implementation-specific. In this case the comment should identify the need to identify a stopping rule during implementation.

row type	msg element name	in msg element type	source of msg element type	of msg element type
class	Clinical_document_header	Message	New type	Clinical_document_header
attr	availability_status_cd	Clinical_document_header	Data type	CV
assoc	is_related_to_Authentication	Clinical_document_header	New type	Authentication

row type	msg element name	in msg element type	source of msg element type	of msg element type
attr	authentication_dttm	Authentication	Data type	TS
assoc	is_source_of_Healthcare_Document_authenticator	Authentication	New type	Healthcare_document_authenticator
assoc	is_a_Organization	Healthcare_Document_authenticator	New type	Organization
attr	organization_nm	Organization	Data type	ST
attr	addr	Organization	Data type	ADDR
assoc	is_a_subdivision_of_Organization	Organization	Recursion	Organization

10.3.3.4 Avoiding Name Collisions

No two components of the same message element type may have the same short name.

10.3.3.5 Finalize the Message Information Model

It is possible that the committee added classes that were not originally in the Message Information Model or found that some classes were not required. It should modify the Message Information Model so that it at least includes all the classes that were used in building the Hierarchical Message Definition. The committee may leave extra classes in if they are used in other Hierarchical Message Definitions drawn from the same Message Information Model.

10.3.4 Creating the Common Message Element Type

A technical committee creates Common Message Element Types using a process that is very similar to that for creating a Hierarchical Message Definition. This section describes the process placing emphasis on the differences.

10.3.4.1 Determining the Requirements

The analytic process for creating a Common Message Element Definition is virtually identical to that for creating a Hierarchical Message Definition. The committee should identify the following:

- a name that uniquely identifies the CMET
- a root class
- the requirements for related information and the scenarios under which the types are used
- the states of the root class that are anticipated in the definition of the types

The committee will create a Message Information Model for one CMET or for a related set of CMETs.

10.3.4.2 Building the CMET

The committee builds the message Common Message Element Type Definition following the same steps used to create a Hierarchical Message Definition. It creates a Refined Message Information Model and then performs a tree walk to create Common Message Element Type Definition.

10.4 Discussion and Helpful Hints

10.4.1 Representing Associations by Containment: Pseudo-hierarchies

Some committee members are uncomfortable with the appearance that message elements contain subordinate message elements. At first it seems a bit odd to say that the Patient contains the Patient_encounter, which contains some Encounter_practitioners, which contain some Individual_healthcare_practitioners, etc. Indeed, the metaphor leads to a seeming paradox when two different Encounter_practitioners contain the same Individual_healthcare_practitioner. The advantage of this approach is that it makes the hierarchy of the message clear.

This approach is supported by our method for naming the message elements that correspond to associations: the association name followed by the name of the distal class. This method treats the message elements that correspond to associations in a manner that is very similar to the message elements that correspond to attributes of the RIM. For example the Patient_encounter message element has two message elements as components (among others):

- **end_dttm** is a message element of type TS
- **has_as_participant_Encounter_practitioner** is a message element of type Encounter_practitioner.

One of these “variables” contains a time stamp and the other contains an encounter_practitioner.

This uniform treatment of RIM attributes and associations is the justification for referring to them together as properties of the class.

The ITS must define how to deal with the seeming paradox of an object appearing at two places in the message instance. This would occur in our example if the same physician were the attending physician and the admitting physician, or the attending physician and the primary care provider.

The ITS specifications may call for replication, so that the data that describes the object appears at both places in the message. It may also call for a way to represent the second occurrence of the same object as a “stub” that contains only enough information to point to the first occurrence, where the data actually resides. Indeed, an ITS may offer both techniques. This implementation technique is hidden from those who design HMDs. They do not specify which technique to use. From a naïve examination of the HMD it would appear that the information is always replicated.

Because the hierarchy of our message permits the same object to appear at more than one node of the hierarchy, we can think of our messages as being “pseudo-hierarchies”.

10.5 Criteria for Evaluation of Work Products

This section contains criteria for use in quality reviews of work products. No criterion is an absolute, although some are more broadly applicable than others.

10.5.1 Refined Message Information Model

Some criteria that will be used in evaluating a Refined Message Information Model are listed below.

1. All classes, attributes and associations are represented in the Reference Information Model.

2. All clone classes are identified with a name extends the name of a base class and describes the semantic distinction.
3. The cardinality of all associations is the same as or more restrictive than that in the Message Information Model.
4. At least one class has an absolute cardinality, with a minimum cardinality greater than zero.
5. Associations that are only used in one direction are labeled as such in the graphical representation. In the tabular representation, half-associations that are never used are omitted.

10.5.2 Hierarchical Message Definition

1. All message element types are based on entities of the Refined Message Information Model, Common Message Element Types or data types.
2. All associations are based on associations in the Refined Message Information Model
3. All constraints on message elements are the same as, or more restrictive than, the corresponding constraints in the source of the message element (i.e., the Refined Message Information Model, a Common Message Element Type Definition, or a data type definition.)
4. The Hierarchical Message Definition includes, at a minimum, all messages sent from a single application role to all receiving roles for a single trigger event. Where practical it includes all messages for all trigger events based on state changes in the same subject class for all receiving roles.
5. No two components of a message element may have the same short name.
6. All coded values have a stated domain.
7. Message elements that represent significant objects contains a state attribute (status_cd) and an instance identifier. (“Significant” is a subjective term; it implies that the sending and receiving systems maintain keys to identify instances of the objects.)
8. All rows that close a recursion loop are labeled as recursive in the **source of message element type** column. This includes, but is not limited to, rows that have the same value for the **in message element type** and **of message element type** columns.
9. Rows that are identifies as closing a recursion loop contain a constraint or comment (a) defines a stopping rule for recursion, or (b) indicates that the stopping rule must be determined at implementation time.
10. All references to sources of message element types are valid.

Exhibit 10-1

Row Number	Row Type	ClassorProperty (AttributeorAssociation)	ShortName	Inherited From	MessageElementType	Cardinality	Domain Specification (#)	Coding Strength	Mandatory	Constraint/ Note #	Default Value (#)	Default Update Mode	Update mode set	Conformance Flag
3	mmim	C00_RIM_0092D												
4	class	Patient_encounter	PtEncntr			0..1								
5	attr	id	id		II	1..1			M					R
6	attr	status_cd	status		CV	1..*	<@state>	CNE	M					R
7	attr	encounter_classification_cd	classfcn		CV	0..1		CNE						
8	attr	start_dttm	startDttm		TS	0..1								
9	attr	end_dttm	endDttm		TS	0..1								
10	attr	expected_insurance_plan_qty	expctdInsrncePlanQty		INT	0..1								
11	attr	first_similar_illness_dttm	firstSimlrIllnssDttm		TS	0..1								
12	assoc	has_as_participant	hasAsPartcptntEncntrPractnr		Encounter_practitioner	1..*								
13	assoc	generalizes			Inpatient_encounter	1..1								
14	assoc	involves	involvesPt		Patient	1..1								
15	class	Inpatient_encounter	InptEncntr			0..1								
16	attr	id	id	Patient_encounter	II	1..1			M					R
17	attr	status_cd	status	Patient_encounter	CV	1..*	<@state>	CNE	M					R
18	attr	encounter_classification_cd	classfcn	Patient_encounter	CV	0..1		CNE						
19	attr	end_dttm	endDttm	Patient_encounter	TS	0..1								
20	attr	expected_insurance_plan_qty	expctdInsrncePlanQty	Patient_encounter	INT	0..1								
21	attr	first_similar_illness_dttm	firstSimlrIllnssDttm	Patient_encounter	TS	0..1								
22	attr	start_dttm	startDttm	Patient_encounter	TS	0..1								
23	attr	actual_days_qty	actualDaysQty		INT	0..1								
24	attr	estimated_days_qty	estmtdDaysQty		INT	0..1								
25	assoc	specializes			Patient_encounter	1..1								
26	class	Encounter_practitioner	EncntrPractnr			0..1								
27	attr	participation_type_cd	partcptntType		CV	0..1		CNE	M	1				R
28	assoc	is_associated_with	isAsscdWithPtEncntr		Patient_encounter	1..1								
29	assoc	is_participant_for	isPartcptntForIHCP		Individual_healthcare_practitioner	1..1								
30	class	Individual_healthcare_practitioner	IHCP			0..1								
31	attr	id	id		II	1..1			M					R
32	attr	specialty_cd	speclty	Healthcare_service_provider	CV	0..1		CNE						
33	attr	practitioner_type_cd	type		CV	0..1		CNE						
34	attr	residency_field_cd	resdncyField		CV	0..1		CNE						
35	assoc	participates_as	partcpesAsEncntrPractnr		Encounter_practitioner	0..*								
36	assoc	is_role_of	isRoleOfPersnsIHCP		Person_as_IHCP	1..1								
37	class	Person_as_IHCP	IHCPpersn			0..1								
38	attr	addr	addr	Stakeholder	AD	0..1								
39	attr	id	id	Stakeholder	SET<II>	0..1								
40	attr	phon	phon	Stakeholder	TEL	0..1								
41	assoc	has	hasPrsnName		Person_name_for_IHCP	1								
42	class	Person_name_for_IHCP	PrsnNameForIHCP			1..1			M					R
43	attr	cd	cd		CV	0..1		CNE						
44	attr	nm	nm		PN	0..1								
45	attr	purpose_cd	purpse		CV	0..1		CNE						
46	attr	type_cd	type		CV	0..1		CNE						
47	class	Patient	Pt			0..1								
48	attr	id	id		Set<II>	0..*			M					R
49	attr	status_cd	status		Set<CV>	1..*			M					R
50	assoc	has_a_primary_provider	hasAprimryProvdtrIHCP		Individual_healthcare_practitioner	0..1								
51	assoc	is_a_role_of	isARoleOfPrsn		Person_as_Patient	1..1								
52	assoc	has	hasPtBillingAcctnt		SetList<Patient_billing_account>	0..*								
53	assoc	is_involved_in	isInvlvdInPtEncntr		Patient_encounter	1								
54	class	Person_as_Patient	PtPrsn			0..1								
55	attr	addr	addr	Stakeholder	AD	0..1								
56	attr	id	id	Stakeholder	SET<II>	0..1								
57	attr	phon	phon	Stakeholder	TEL	0..1								
58	attr	type_cd	type	Stakeholder	CV	0..1		CNE						
59	attr	administrative_gender_cd	adminvGendr		CV	0..1		CNE						
60	attr	birth_dttm	birthDttm		TS	0..1								
61	attr	citizenship_country_cd	citznshpCountry		CV	0..1		CNE						
62	attr	deceased_dttm	decsdDttm		TS	0..1								
63	attr	deceased_ind	decsdInd		BL	0..1								

64	attr	ethnic_group_cd	ethncGroup	CV	0..1	CNE		
65	attr	marital_status_cd	marlStatus	CV	0..1	CNE		
66	attr	race_cd	race	CV	0..1	CNE		
67	attr	religious_affiliation_cd	relgsAffltn	CV	0..1	CNE		
68	attr	very_important_person_cd	veryImprtnt	CV	0..1	CNE		
69	assoc	<i>has</i>	<i>hasPrsnName</i>	<i>Person_name_for_Patient</i>	0..*			
70	class	Person_name_for_Patient	PrsnNameForPt		0..1			
71	attr	cd	cd	CV	0..1	CNE		
72	attr	effective_dt	effctvDt	TS	0..1			
73	attr	nm	nm	PN	0..1			
74	attr	purpose_cd	purpse	CV	0..1	CNE		
75	attr	termination_dt	termtnDt	TS	0..1			
76	attr	type_cd	type	CV	0..1	CNE		
77	class	Patient_billing_account	PtBillingAcct		0..1			
78	attr	account_id	id	II	1..1		M	R
79	attr	billing_status_cd	status	CV	0..1	CNE		
80	attr	patient_financial_class_cd	finclClass	CV	0..1	CNE		
81	attr	price_schedule_id	priceSchedId	II	0..1			
82								

Exhibit 10-2

Row Number	Row Type	Class or Property of Class (Attribute or Association)	Rim Source Class	Message Element Name	Message Element Short Name	In Message Element Type	Source of Message Element Type	of Message Element Type	Cardinality	Domain Specification (#)	Coding Strength (default CWE)	Mandatory	Constraint/Note #	Default Value (#) Default Default = "N1"	Default Update Mode Default Default = R	Update mode set
<div style="border: 1px solid black; padding: 5px; width: fit-content;"> Note: this exhibit needs to be expanded to show message type definitions when R2T2 can generate them. </div>																
3	hmdl	C00_RIM_0092D	C00_MDF_0092D	C00_RRIM_0092D	C00_HMD_0092D	HMD										
4	class	Patient	Patient	Patient	Pt	MessageType	N	Pt	0..1							
5	attr	id	Patient	id	id	Pt	D	II								
6	attr	status_cd	Patient_encounter	status_cd	status	Pt	D	CV	1..1		CWE	M				
7	assoc	is_a_role_of	Patient	is_a_role_of_Person_as_Patient	isARoleOfPersnAsPt	Pt	N	PersnAsPt	1..1							
8	attr	administrative_gender_cd	Person	administrative_gender_cd	adminvGendr	PersnAsPt	D	CV	0..1		CNE					
9	attr	birth_dttm	Person	birth_dttm	birthDttm	PersnAsPt	D	TS	0..1							
10	attr	citizenship_country_cd	Person	citizenship_country_cd	citznshpCountry	PersnAsPt	D	CV	0..1		CNE					
11	attr	deceased_dttm	Person	deceased_dttm	decsdDttm	PersnAsPt	D	TS	0..1							
12	attr	deceased_ind	Person	deceased_ind	decsdInd	PersnAsPt	D	BL	0..1							
13	attr	ethnic_group_cd	Person	ethnic_group_cd	ethncGroup	PersnAsPt	D	CV	0..1		CNE					
14	attr	marital_status_cd	Person	marital_status_cd	marlStatus	PersnAsPt	D	CV	0..1		CNE					
15	attr	race_cd	Person	race_cd	race	PersnAsPt	D	CV	0..1		CNE					
16	attr	religious_affiliation_cd	Person	religious_affiliation_cd	relgsAffltn	PersnAsPt	D	CV	0..1		CNE					
17	attr	very_important_person_cd	Person	very_important_person_cd	veryImptrntPrsn	PersnAsPt	D	CV	0..1		CNE					
18	attr	addr	Stakeholder	addr	addr	PersnAsPt	D	AD								
19	attr	id	Stakeholder	id	id	PersnAsPt	D	SET -II-								
20	attr	phon	Stakeholder	phon	phon	PersnAsPt	D	TEL								
21	attr	type_cd	Stakeholder	type_cd	type	PersnAsPt	D	CV	0..1		CNE					
22	assoc	has	Person	has_Set_Person_name_for_Patient	hasSetPrsnNameForPt	PersnAsPt	N	Set <PrsnNameForPt>	0..*							
23	item		Person	_PrsnNameForPt	_PrsnNameForPt	Set <PrsnNameForPt>	D	PrsnNameForPt	1							
24	attr	cd	Person_name	cd	cd	PrsnNameForPt	D	CV			CWE					
25	attr	effective_dt	Person_name	effective_dt	effctvDt	PrsnNameForPt	D	TS	0..1							
26	attr	nm	Person_name	nm	nm	PrsnNameForPt	D	PN								
27	attr	purpose_cd	Person_name	purpose_cd	purpse	PrsnNameForPt	D	CV	0..1		CNE					
28	attr	termination_dt	Person_name	termination_dt	termntDt	PrsnNameForPt	D	TS	0..1							
29	attr	type_cd	Person_name	type_cd	type	PrsnNameForPt	D	CV	0..1		CNE					
30	assoc	has	Patient	has_Set_Patient_billing_account	hasPtBillingAcct	Pt	N	Set <PtBillingAcct>	0..*							
31	item		Patient	_PtBillingAcct	_PtBillingAcct	Set <PtBillingAcct>	D	PtBillingAcct	1							
32	attr	account_id	Patient_billing_account	account_id	id	PtBillingAcct	D	II	0..1							
33	attr	billing_status_cd	Patient_billing_account	billing_status_cd	status	PtBillingAcct	D	CV	0..1		CNE					
34	attr	patient_financial_class_cd	Patient_billing_account	patient_financial_class_cd	finclClass	PtBillingAcct	D	CV	0..1		CNE					
35	attr	price_schedule_id	Patient_billing_account	price_schedule_id	priceSchedId	PtBillingAcct	D	II	0..1							
36	assoc	has_a_primary_provider	Patient	has_a_primary_provider_Individual_healthcare_practitioner	hasAprimryProvdrlHCP	Pt	N	IHCP	0..1							
37	attr	specialty_cd	Healthcare_service_provider	specialty_cd	spectly	IHCP	D	CV	0..1		CNE					
38	attr	id	Healthcare_service_provider	id	id	IHCP	D	II								
39	attr	practitioner_type_cd	Healthcare_service_provider	practitioner_type_cd	type	IHCP	D	CV	0..1		CNE					
40	attr	residency_field_cd	Healthcare_service_provider	residency_field_cd	resdncyField	IHCP	D	CV	0..1		CNE					
41	assoc	is_role_of	Healthcare_service_provider	is_role_of_Person as_IHCP	isRoleOfPersnAsIHCP	IHCP	N	PersnAsIHCP	1..1							
42	attr	addr	Stakeholder	addr	addr	PersnAsIHCP	D	AD								
43	attr	id	Stakeholder	id	id	PersnAsIHCP	D	SET -II-								
44	attr	phon	Stakeholder	phon	phon	PersnAsIHCP	D	TEL								
45	attr	type_cd	Stakeholder	type_cd	type	PersnAsIHCP	D	CV	0..1		CNE					
46	assoc	has	Person	has_Person_name_for_IHCP	hasPrsnNameForIHCP	PersnAsIHCP	N	PrsnNameForIHCP	1							
47	attr	cd	Person_name	cd	cd	PrsnNameForIHCP	D	CV			CWE					
48	attr	effective_dt	Person_name	effective_dt	effctvDt	PrsnNameForIHCP	D	TS	0..1							
49	attr	nm	Person_name	nm	nm	PrsnNameForIHCP	D	PN								

49	atr	nm	Person_name	nm	nm	PsnNameForIHCP	D	PN										
50	atr	purpose_cd	Person_name	purpose_cd	purpse	PsnNameForIHCP	D	CV	0.1		CNE							
51	atr	termination_dt	Person_name	termination_dt	termtnDt	PsnNameForIHCP	D	TS	0.1									
52	atr	type_cd	Person_name	type_cd	type	PsnNameForIHCP	D	CV	0.1		CNE							
53	assoc	is_involved_in	Patient	is_involved_in_Patient_encounter	isInvlvdInPtEncntr	Pt	N	PtEncntr InptEncntr	1									
54	atr	encounter_classification_cd	Patient_encounter	encounter_classification_cd	classfcn	*	D	CV	0.1		CNE							
55	atr	end_dtm	Patient_encounter	end_dtm	endDtm	*	D	TS	0.1									
56	atr	expected_insurance_plan_qty	Patient_encounter	expected_insurance_plan_qty	expcdInsmcePlanQty	*	D	INT	0.1									
57	atr	first_similar_illness_dtm	Patient_encounter	first_similar_illness_dtm	firstSimlrIllnssDtm	*	D	TS	0.1									
58	atr	id	Patient_encounter	id	id	*	D	II	1..1			M						
59	atr	start_dtm	Patient_encounter	start_dtm	startDtm	*	D	TS	0.1									
60	atr	status_cd	Patient_encounter	status_cd	status	*	D	CV	1..1		CWE	M						
61	assoc	has_as_participant	Patient_encounter	has_as_participant_Set_Encounter_practitioner	hasAsPartcpntSetEncntrPractnr	*	N	Set <EncntrPractnr>	1..*									
62	item		Patient_encounter	_EncntrPractnr	_EncntrPractnr	Set <EncntrPractnr>	D	EncntrPractnr	1									
63	atr	participation_type_cd	Encounter_practitioner	participation_type_cd	partcptnType	EncntrPractnr	D	CV	0.1		CNE	1						
64	assoc	is_participant_for	Encounter_practitioner	is_participant_for_Individual_healthcare_practitioner	isPartcpntForIHCP	EncntrPractnr	U	IHCP	1..1									
65	atr	actual_days_qty	Inpatient_encounter	actual_days_qty	actualDaysQty	InptEncntr	D	INT	0.1									
66	atr	estimated_days_qty	Inpatient_encounter	estimated_days_qty	estmdDaysQty	InptEncntr	D	INT	0.1									

Exhibit 10-3

1 at least one of the instances must identify the attending physician

11. Developing HL7 Models using UML and Rational Rose

11.1 Introduction

HL7 has provided a variety of tools for modelers to use in capturing and recording the models required by the HL7 Version 3 Message Development Framework (MDF). This chapter guides the HL7 modeler in the use of Rose-98 from Rational Software to record and document several of the models and it introduces the use of RoseTree II for developing the Hierarchical Message Description.

The HL7 Version 3 methodology follows the Unified Modeling Language (UML) specification very closely. Although Rose is intended to be fully UML compliant, not all of the model elements specified in the MDF can be directly captured in Rose. Similarly, not all of the modeling capabilities of Rose are used in developing HL7 models. This chapter addresses these differences.

Rose provides source information for the Use Case, Information and Interaction Models. The documentation and maintenance of distinct versions of the HL7 Reference Information Model, and the development of message design specifications are both integrally dependent upon a model repository, which is an SQL database maintained in Access. A family of tools, including RoseTree, allows for capturing models in the repository, provides access to the repository and assists the user in defining message design specifications.

In order to provide for version control and the management of all of the MDF model information, the contents of the Rose models have been extended using data fields captured as "properties" added to the Rose elements. On an element-by-element basis, this chapter defines each such property, its name and intended use.

11.2 Model terminology, properties, descriptions and stereotypes

11.2.1 Preparing Rose for properties and stereotypes

As discussed in the following paragraphs, HL7 has defined a number of additional properties and stereotypes for elements of Rose models. In order to use these properties and stereotypes, files that define these must be placed in a place where Rose can use them. The files are distributed in a zipped archive named HL7_Rose_additions.ZIP. This package contains three files. The installation and use of these files is as follows:

11.2.1.1 hl7RIM.pty (Property file)

This file contains the definition of properties that HL7 has added to Rose. This file is only needed when the modeler wishes to create a new model that uses the HL7 properties. In that case, first place the file hl7RIM.pty in the same directory that holds "Rose.exe." This is usually found in your "Program Files" directory under sub-directories "\Rational\Rational Rose 98". With the file in place, create a new model (Menu - File:New). Then select (Menu - Tools:Model Properties:Update...) and select the hl7RIM.pty file from the dialog box that appears. This will add the HL7 properties tabs. The addition can be verified by reviewing the options (Menu - Tools:Options...), in which case there should be an HL7 "tab" on the dialog.¹

11.2.1.2 HL7_stereo_icons.bmp (Stereotype icons)

This file contains the icons for displaying HL7 stereotypes in the Rose browser. Place this file in the same directory as Rose.exe

¹ The HL7 toolsmith plans a somewhat simpler approach to accomplishing this task and the task of adding stereotypes (below). Until that happens, this method must suffice.

11.2.1.3 HL7_DefaultStereotypes.ini (Stereotype definitions)

This file is a replacement for the file "DefaultStereotypes.ini" that is part of the Rose98 installation package. It contains definitions for the HL7 stereotypes. To install, this file, first find the file "DefaultStereotypes.ini" in the directory that contains "Rose.exe." Rename this file "DefaultStereotypes.bak" to be certain that you can restore it, should that ever be desired. rename the file "HL7_DefaultStereotypes.ini" to "DefaultStereotypes.ini," and place it in the same directory that holds "Rose.exe." Close Rose, if it is an open application. The new stereotypes will be available the next time you start Rose.²

11.2.2 Package vs. Category

Rose uses the terms "Package" and "Category" somewhat interchangeably. According to the Rose help file, a package "consists of a specification module and an implementation module" and appears on a component diagram. Rose also says that a category "allows you to define and manipulate logical collections of classes." Thus, category is the term that HL7 wishes to use. Note, however, that in order to create a new category in Rose, one must click on the "package" symbol on the tool-bar, or select "New:Package: from the right-click menu in the browser.

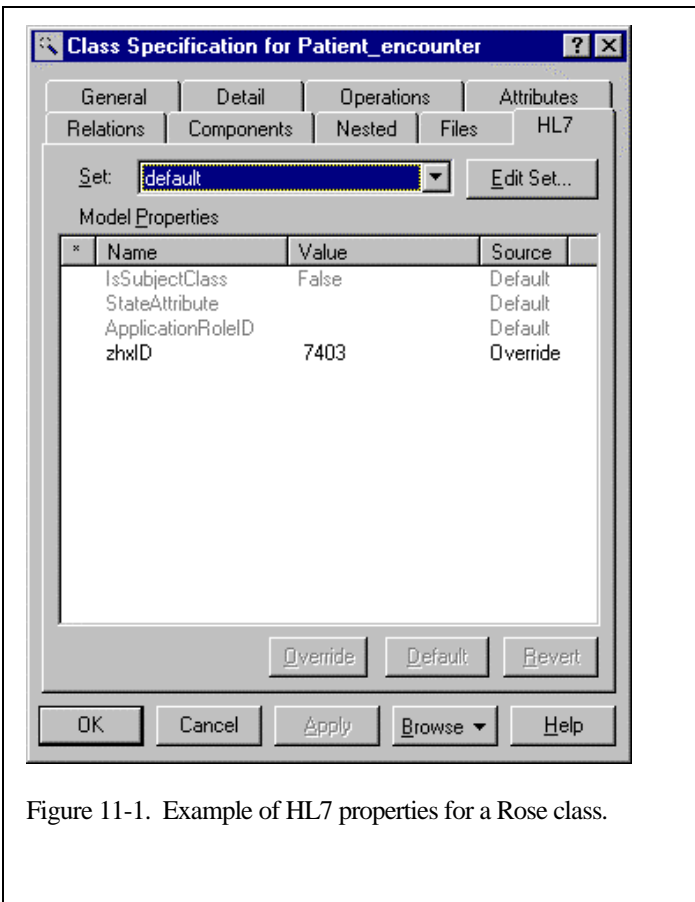


Figure 11-1. Example of HL7 properties for a Rose class.

11.2.3 The use of added HL7 properties to capture meta-data

In order to meet its modeling needs, HL7 requires additional information for many of the model elements. HL7 provides a property set for the model that serves to capture this additional information as HL7 properties defined in the specification of each model element.

Figure 1 is an example of a specification dialog from Rose that shows a "tab" titled "HL7," and an example of a set of properties for a particular class recorded therein. Not all of the properties for this class have been set. Those properties that have been set are labeled as "Override," and those that have not are shown in gray and labeled "Default."

A complete listing of the properties defined for use by HL7 modelers is included in Table 11-1. This chapter discusses each property as part of the discussion of the element to which the property applies later in this documents.

The full set of properties may also be

viewed in Rose by selecting (Menu - Tools:Options...) and then selecting the HL7 tab. The Type field (see Figure 11-3) selects the model element, and the box below that field will list each of the applicable properties.

² If your copy of Rose already contains additional stereotypes that you wish to retain, contact the HL7 toolsmith for an alternate method of accomplishing this installation.

11.2.3.1 Loading properties in Rose

HL7 generates and distributes most of the models an HL7 modeler will require. These models have the properties included. To add HL7 properties to a "new" model, you must have a copy of the file hl7RIM.pty on your system, as described above. Create a new model (Menu - File:New). Then select (Menu - Tools:Model Properties:Update...) and select the hl7RIM.pty file from the dialog box that appears. This will add the HL7 property tabs to specification dialogs. The addition can be verified by reviewing the options (Menu - Tools:Options...).

11.2.3.2 Generic **zhxID** property for capturing model history.

Every element of an HL7 model includes a property **zhxID**. This property is a unique identifier (GUID) used to track the version history of each element of the model. The repository tools assign the GUID values. Modelers should never change these values or assign new ones, but they may copy them to a new model element to indicate that the new element's derives from the element whose GUID is copied.

11.2.4 Capturing rationale. Issues and references in descriptions

HL7 uses special paragraphs in the description of each element to capture comments about the rationale for modeling, any open issues, and references to external source documents. These paragraphs appear as the closing paragraph(s) of the element description or documentation. The special paragraphs begin with a reserved phrase that serves to identify them. The reserved phrases - "**OpenIssue:**" "**Rationale:**" and "**ExtRef:**" - must not be used to start paragraphs in the body of the description. The use of these special paragraphs is:

Rationale: This allows the modeler to document the rationale or justification for the specification of a particular element. It may occupy one or more paragraphs, but only one modeling rationale component should appear for any given model element. The first paragraph of the rationale should begin "Rationale:" The rationale ends when another reserved-phrase paragraph occurs, or when the documentation field ends, whichever comes first.

OpenIssue: This allows the modeler to identify and discuss any open issues that remain to be resolved with respect to the model element. It may occupy one or more paragraphs, and there may be multiple open issues for a model element. The first paragraph of each open issue should begin with "OpenIssue:" The open issue ends when another reserved-phrase paragraph occurs, or when the documentation field ends, whichever comes first.

ExtRef: This paragraph allows the modeler to identify an external document that contains additional information about the element. This reference may be a URL or the name and identifier of the external document. Each external reference occupies a single paragraph, and there may be multiple external references for a model element. If the external reference is not the final paragraph of the documentation, it must be followed by another reserved-phrase paragraph.

11.2.5 Use of Rose Stereotypes by HL7

Rose provides the ability to define stereotypes for model elements. The stereotype provides two useful functions. First, it can display a unique icon in the Rose browser which permits the ready identification of the different stereotypes used in building a model. Secondly, the model export programs can detect the stereotypes and use them to control model processing.

11.2.5.1 Stereotypes defined by HL7

To take advantage of this facility, HL7 has defined three special stereotypes for Rose categories, two stereotypes for Rose classes, two stereotypes for Rose use cases and two stereotypes for generalization of use cases. An overview of these stereotypes follows, and examples can be seen in Figure 11-2.

Application_role is a stereotype for a class. It distinguishes classes used to define HL7 application roles in the interaction model. Its browser icon is a class symbol with a violet letter "A" on it.

Data_type is a stereotype for a class. It distinguishes class symbols used to define HL7 data types. Its browser icon is a class symbol with a blue letter "D" on it.

Data_type_category is a stereotype for a category (package). It classifies those categories in the Information Model that contain data type definitions. Its browser icon is a directory icon with a blue letter "D" on it.

extend is a stereotype of the dependent relationship used in use case models, in which one use case adds additional behavior to another.

include is a stereotype of the dependent relationship used in use case models in which one use case uses another as part of its execution

Interaction_model_category is a stereotype for a category (package). It classifies those categories that contain the interaction model definitions. Its browser icon is a directory icon with the letters "In" in violet on it.

specialize is a stereotype of the dependent relationship used in use case models in which one use case specializes another.

Storyboard is a stereotype for a use case. It identifies those use cases in the interaction model that define storyboards. Its browser icon is a use case icon with the letter "S" in violet on it.

Storyboard_example is a stereotype for a use case. It identifies those use cases in the interaction model that contain example text for storyboards. Its browser icon is a use case icon with the letter "x" in violet on it.

Use_case_model_category is a stereotype for a category (package). It classifies those categories that contain the use case model definitions. Its browser icon is a directory icon with a red letters "U" on it.

11.2.5.2 Invoking HL7 stereotypes

Rose provides a stereotype drop-down list on the "General" tab of the specification dialog for each model element. To invoke one of the HL7 stereotypes, simply click on this drop-down and select the desired stereotype.

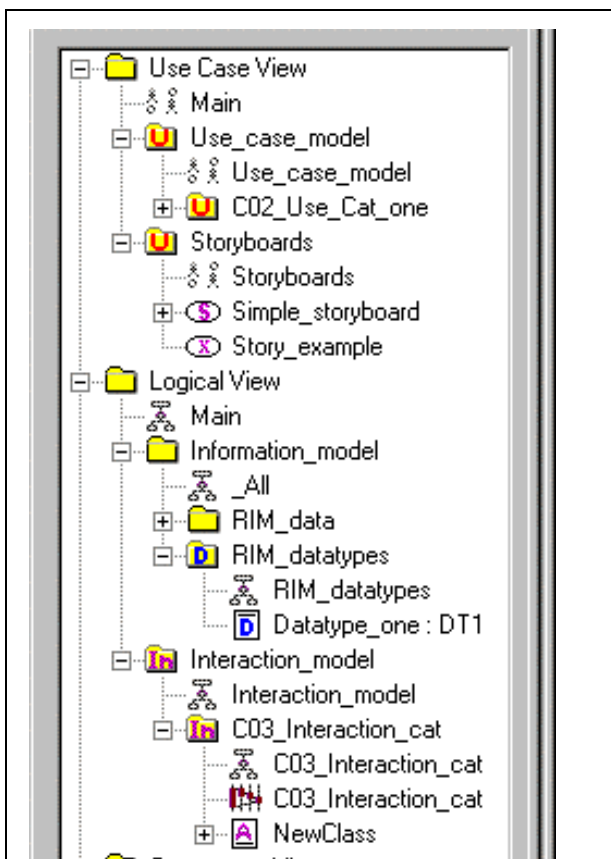


Figure 11-2. Structure of HL7 Models in Rose Browser

11.3 Structure of the HL7 Models as represented in Rose

11.3.1 Overall structure of the Rose model

HL7 imposes a specific overall structure on the models developed in Rose. This facilitates the automated processes used to export models from Rose to the repository and vice versa. The structure also serves to promote uniformity in the HL7 models. The structure of a Rose model can be viewed as a directory tree structure with the Rose browser (Menu- View:Browser).

At the root level, Rose defines four default elements. These are three packages labeled "Use Case View," "Logical View" and "Component View", and a processor labeled "Deployment View." HL7 models use only the first two of these. HL7 defines four second-level categories to hold its models. These are the "Use_case_model" defined under the default Use Case View, the "Information_model" and the

"Interaction_model" defined under the default Logical View, and the "Storyboards" defined under the default Use Case View. The HL7 categories correspond to the models of the same name as defined in the MDF. These categories use the category defined above, and their browser symbols are shown in Figure 11-2.

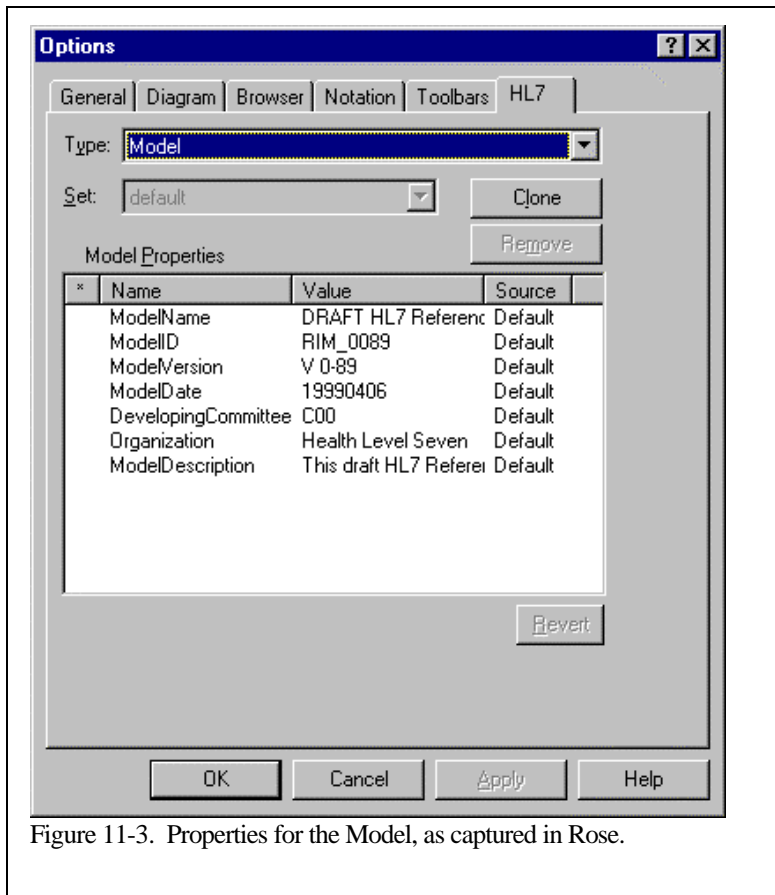


Figure 11-3. Properties for the Model, as captured in Rose.

Figure 11-2 shows the Rose browser with the four default packages, and the HL7-defined categories underneath them. In order to simplify navigation through the model, the "Main" diagram under both the Use Case View and the Logical View will hold category symbols for the use case diagrams and information model subject areas respectively.

11.3.2 Model definition

Each model contains a special set of properties to hold information about the model itself. The following attributes apply to an

HL7 model: model identifier, name, version number, last modified date, developing organization, committee identifier, and description. Record these attributes as properties in the model property set

11.3.2.1 Creation with Rose

In Rose, select (Menu - Tools:Options...). Then select the HL7 tab, and select Model in the Type pull-down. The result will be a dialog box similar to the one in Figure 11-3. You can change the values by selecting them and entering a new value. The Apply button captures all of the new values.

11.3.2.2 Qualifying properties

The **ModelName** property holds the formal name for the model. The unique model identifier (such as RIM_0089) is recorded in **ModelID**. **ModelVersion** holds the version number in a stylized string such as "V 0-89." **ModelDate** is the last-modified date for the model expressed in a YYYYMMDD format.

DevelopingCommittee holds the committee identifier (such as C00 for the Methodology and Modeling Committee). **Organization** is always "Health Level Seven." Finally, **ModelDescription** holds one or more paragraphs describing the model. This cannot be readily edited in the properties dialog box. It is easier to prepare this in a separate text editor (such as NotePad), copy it, and paste it into the dialog by selecting the whole of the current entry and pressing Ctrl-V.

11.4 Use case model

11.4.1 Model Structure

The use case model is captured under two sub-categories in Rose. Record the use cases under the Use_case_model category. Capture storyboards under a single category names "Storyboards," where the parent category is Rose's "Use Case View. " Committees may create use case categories under these two master categories, and may nest the categories they define.

11.4.2 Use case model categories

Committees may add committee-defined categories to group use case and actor definitions in whatever categories the committee finds convenient. For example, they might group use cases by project under a category defined by the committee. Use case categories carry the Use_case_model_category stereotype,. The attributes of a use case category are its name and description.

11.4.2.1 Creation with Rose

Create a new category by right-clicking on the Use_case_model category in the Rose Browser, and selecting (New:Package). To create a nested category, right-click on the parent category in the browser. Open the specification box for the new category and select the Use_case_model_category stereotype. You must provide a name for the new category that starts with "Cnn_" where Cnn is the identifier for the committee. There are no constraints on the category name except those imposed by the requirements for decency and uniqueness.

For each created category, add a use case diagram with the same name as the category. Do this by right-clicking on the category icon in the browser and selecting (New:Use case diagram).

11.4.2.2 Qualifying properties

All use case categories must have the **RespComm_id** property set to the identifier of the responsible committee.

11.4.2.3 Putting actors and use cases in defined categories

The presence of an actor or use case on the use case diagram for a given category documents that the actor or use case is part of that category. An actor or use case should be defined one of the category diagrams at the lowest level of nesting. The modeler should also drag the actors and use cases to the higher-level diagrams to record their appearance at that level, too. The modeler can make an actor or use case appear in as many categories as desired, simply by dragging them to the appropriate diagram.

11.4.3 Use cases

The following attributes define a use case: identifier, title (or name), description, and for leaf-level use cases the subject class, start state, end state and transition label for the associated state transition.

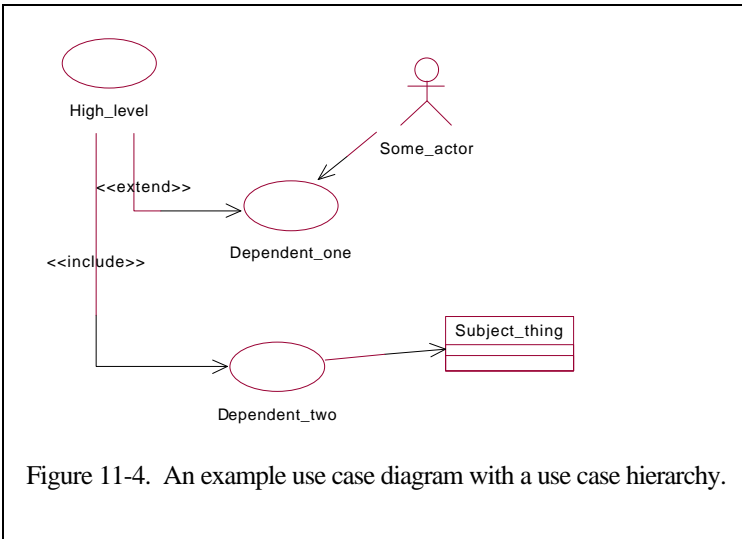


Figure 11-4. An example use case diagram with a use case hierarchy.

11.4.3.1 Creation with Rose

The use case itself is defined simply from the use case diagram tool bar. For Use Cases, the HL7 MDF meta-model specifies an identifier that is distinct from the name (title). Capture this identifier in Rose using the **ID** property. Enter the description in the normal Rose fashion. Creating relationships between use cases, between actors and use cases and between leaf-level use cases is

discussed below.

11.4.3.2 Qualifying properties

All use cases require the **ID** property to record their unique identifier. In addition, each leaf-level use case is linked to a state transition in one of the subject classes. Although the relevant subject class is identified by a relationship, the remaining elements that define the transition are entered as three properties. These are the name of the starting state as **StartState**; the name of the ending state as **EndState**; and the transition name as **StateTransition**.

11.4.4 Actors

The following attributes define an actor: name and description.

11.4.4.1 Creation with Rose

Create an actor by selecting the actor symbol from the use case diagram toolbar. Record the actor name and description in the specification dialog. There are no properties associated with actors, except for the history and version attribute. (Because an Actor is a stereotype of a class, there are other properties listed on the HL7 tab for actors, but these are used to capture information for classes, application roles and data types, which are also stereotypes of class.)

11.4.5 Use case dependency relationships

There no attributes for the use case dependency relationship.

11.4.5.1 Creation with Rose

Construct use case dependency relationships in Rose using the "Uni-directional association" tool. Select the Uni-directional association tool and drag from the higher-level use case to the dependent use case.

HL7 does not call for a description for a use case dependency relationship. The only property will be the history attribute. Note that the use case model chapter distinguishes two types of generalize relationship

between use cases. These are termed the "includes" and "extends" relationships. Select the appropriate stereotype from the drop-down list on the generalization specification dialog.

11.4.6 Linking actors to use cases

There are no attributes for the actor to use case relationship.

11.4.6.1 Creation with Rose

Draw the relationship that indicates that an actor participates in a use case with the "Unidirectional Association" tool. To establish this relationship, select the tool, and drag from the actor to the use case. Neither a description nor properties (except for the history identifier) are needed for the participation link.

11.4.7 Linkage to the subject class for a state transition

11.4.7.1 Creation with Rose

You must also capture the link between each leaf-level use case and its associated subject class in the use case diagrams. Follow a three-step process.

- 1) Open the Rose browser, and drag the needed subject class from the information model area of the browser onto the use case diagram. To minimize the visual impact on the diagram, right-click on the class and deselect to Show all attributes and Show visibility.
- 2) Select the Unidirectional Association tool from the toolbar. Drag from the use case to its associated subject class. This step is sufficient to document the subject class relationship.
- 3) If you do not wish to have the subject classes shown on the use case diagram, select the subject class and press the delete key. This will delete the subject class and the association from this diagram, but Rose will retain the association.

There is no description for this association, but there is a history property.

11.4.7.2 Qualifying properties

Capture the remainder of the state transition for the leaf-level use case in properties that are part of the use case as discussed above.

11.4.8 Storyboards

Storyboards consist of named, documented sequences of interactions and/or use cases and one or more descriptive examples of the storyboard.

11.4.8.1 Model structure

You will capture storyboards as stereotyped use cases under the Storyboards category, which is in the Use Case View. The "Main" diagram in the Storyboards category shows one use case for each storyboard. Each storyboard contains a use case diagram to hold the storyboard examples, and a sequence diagram to express the sequence of interactions. As of April 1999, the toolsmith has not defined a way to capture use case sequences in Rose.

11.4.8.2 Storyboards

The attributes that apply to a storyboard are: an identifier, a name, and a description of the purpose for the storyboard.

11.4.8.2.1 Creation with Rose

Start on the "Main" diagram for Interaction_model_storyboards and use the Use Case tool to add a use case to the diagram. Set the stereotype to "Storyboard." The name of the use case should be the storyboard name. Enter the purpose for the storyboard as the description. Note that the purpose is different from an example. Entry of the example(s) is discussed below.

11.4.8.2.2 Qualifying properties

Capture the identifier for the storyboard in the use case specification with an **ID** property.

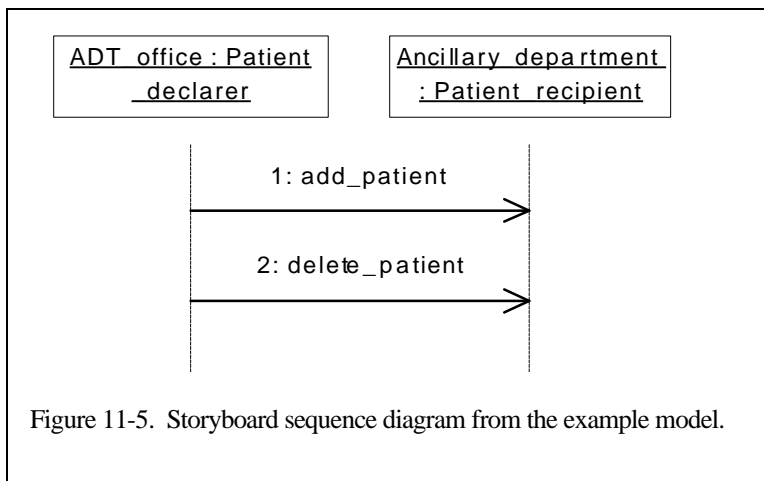
11.4.8.3 Storyboard examples

Each storyboard may have one or more examples. The storyboard examples are documented as stereotyped use cases that reside on a diagram that is contained within the use case that defines storyboard itself. The attributes that apply to a storyboard example are: its identifier and the description.

11.4.8.3.1 Creation with Rose

If this is the first example created for a storyboard, you must create a use case diagram that is contained within the storyboard. In the browser, right-click on the storyboard (use case), and select (New:Use Case Diagram). Rose will create a NewDiagram. Rename this diagram with the same name as the storyboard (use case) that contains it.

Open the use case diagram for the storyboard. Use the Use Case tool to create a new use case. Give this use case the Storyboard_example stereotype. Its description will be the storyboard example. Open the specification for the new use case. The storyboard example may be given any name, but it is recommended that "Ex_" be added to the front of the name to avoid confusion with other names. Use the documentation



of the example to hold the narrative description of the example.

11.4.8.3.1.1 Qualifying property

Place the storyboard example identifier in the **ID** property.

11.4.8.4 Storyboard interaction sequence diagrams

The interaction sequence that constitutes a storyboard is modeled as a Sequence Diagram contained

within the storyboard. Create the sequence diagram by right-clicking on the storyboard use case, and selecting (New:Sequence Diagram). Figure 11-7 is an example from the example model.

11.4.8.4.1 Establishing communication objects

Add the communication objects (columns) for storyboard sequence diagrams in the same way they were added when the interactions were defined, with one important exception. Storyboards document communication between different instances of application roles. In the example contained in Figure 11-7, communication is between a Patient_declarer role in the ADT office and a Patient_recipient in an ancillary department. In this case, the instances are named ADT_office and Ancillary_department. If the storyboard involved a second instance of Patient_declarer (perhaps in the MPI administration office) a column for this instance would be created with an instance name of MPI_administrator. It would also refer to the Patient_declarer role.

Working with the sequence diagram, use the Object tool to create an object for each required application role instance. Name these as appropriate, and use the class box on the general tab of the object specification to link the objects to the application roles that they instantiate.

11.4.8.4.2 Creation with Rose

Use the "Object Message" tool to create each of the interactions in the sequence that make up the storyboard. Create the interaction by dragging from the sending object to the receiving object. Double-click on the new message, and use the drop-down list in the name box of the general tab to select the trigger event for the interaction.

Note that the sequence of the interactions is very important in a storyboard. You can adjust the sequence by sliding the interactions up and down in the diagram once they have been created.

11.4.8.4.3 Qualifying properties

Complete the specification by entering the id of the interaction in the specification of the message using the **ID** property. The only other necessary property is the history property.

11.5 Information model

11.5.1 Model structure

The structure of the information model follows the pattern of the subject areas defined for the RIM and for stewardship. The following sections consider the detailed relationship between classes and their respective subject areas.

Only one item appears at the "root" level of the information model. This is the "_All" diagram, which shows all of the classes of the RIM in a single very large diagram.

11.5.2 Subject areas and data type categories

The following attributes apply to a subject area or data type category: name, description and responsible committee.

11.5.2.1 Creation with Rose

There are a number of RIM subject areas defined by the Methodology and Modeling Committee. These subject areas are hierarchically structured under four to six top-level subject areas. Additional subject areas are defined to indicate the stewardship for classes in the RIM, and to indicate selected classes-of-interest for particular technical committees.

The root level of the Information_model category contains one category for each of the top-level RIM subject areas, one for the data type categories and one for each of the stewardship and classes-of-interest subject areas.

These are created using the Rose browser. Create a new category for each subject area in the information model by right-clicking on an existing category, and selecting New:Package. Use the name of the subject area or of the data type category for the name of the newly created category. Use the category description to capture the description of the subject area. There are no properties, except the history property, associated with a subject area description. Even so, the HL7 tab of the category specification provides for a property named **MIM_id**. This is reserved for categories holding MIM specifications and is discussed at the end of this document. If the new category is a data type category, select the Data_type_category stereotype in the specification dialog.

In addition, create a new class diagram for each subject area or data type category by right-clicking on the subject area category and selecting New:Class diagram. Give the diagram the same name as the parent category.

After all of the categories have been created in the browser, you can drag the lower-level categories onto their parent categories, to create or modify the hierarchy. This is done in the browser.

When classes or data types are created in the information model (see below), they will be created on the class diagram of the lowest level category in which they appear. This will cause the class definition to reside under that category in the browser.

11.5.2.2 Populating higher-level and stewardship subject areas

In order for a class or data type to also appear in the higher-level subject areas or categories, it must be dragged from the browser to the class diagram for the higher-level category. In a similar fashion, classes should be dragged from the browser onto the class diagram for the relevant stewardship or classes-of-interest subject areas.

11.5.2.3 Creating committee DIMs

Committees can use these same methods to create subject areas that represent all or portions of their domain information models (DIM). These subject areas should include a category and a class diagram with the name of the subject area. Classes for the DIM can be dragged from the browser to the requisite diagrams. Note that subject areas for a DIM should be named with a prefix that is the committee identifier of the technical committee in question. (For example, a DIM for the orders and results technical committee might be named "C04_DIM.")

11.5.2.4 Qualifying properties

Record the identifier of the responsible committee in **RespComm_id**. The history identifier will be added during repository processing. The remaining properties are not used for classes.

11.5.3 Classes

The following attributes apply to a class: name, description, abstract indicator, and, for subject classes, the name of the state attribute.

11.5.3.1 Creation with Rose

As noted, create classes for the information model on the class diagram for the lowest level RIM subject area in which the class resides. Create the class with the class tool, and open its specification dialog to name it and to enter its description on the "General" tab. If the class is abstract, check the "Abstract" box on the "Detail" tab of the class specification.

11.5.3.2 Qualifying properties

If the class is a subject class, the **IsSubjectClass** property should be true, and the **StateAttribute** property should be used to hold the name of the attribute within the class that will record the state of the class. A history property will be added during repository processing. The remaining properties are not used for classes.

11.5.4 Attributes

The following attributes apply to attributes: name, datatype, repeating indicator, mandatory indicator, V2.3 datatype, V2.3 field references, and description.

11.5.4.1 Creation with Rose

Create attributes in the specification dialogue for the class of which they are a part. Record the attribute name and description in the usual fashion. The specification of a data type for a RIM attribute occurs when the attribute is used in an HL7 HMD for the first time. Once the data type has been determined for the attribute, record the data type in the "Type" box on the "General" tab of the attribute specification.

11.5.4.2 Qualifying properties

Properties on the HL7 tab of the attribute specification dialog capture the repeatability and inclusion of the attribute, and link the attribute to data types and data elements from HL7 Version 2.3. The **MayRepeat** property should be set true for any RIM attribute that may repeat. The **MandatoryInclusion** property should be set true for any RIM attribute that must be present in all HL7 models and all HL7 messages that derive from this model. The default is false, and setting the indicator true in the RIM is deprecated.

If the attribute is known to derive from one or more data elements in HL7 Version 2.3, complete the **V23_Datatype** property and/or the **V23_Fields** property. The entry for the latter is the data element number (identifier) for the Version 2.3 data element. Record multiple references are recorded creating a comma-delimited list of entries.

11.5.5 Generalizations

The only attribute for a generalization relationship is its description.

11.5.5.1 Creation with Rose

Create the generalization relationship using the Generalization tool. Select the tool, and drag from the subtype to the supertype. Enter a description of the relationship, if desired. Other than history tracking, no other property or attribute is needed for this relationship.

11.5.6 Associations

The following attributes apply to an association between two classes: name, the inverse name, the multiplicity for each of the classes, and description.

11.5.6.1 Creation with Rose

Use the association tool from the toolbar to create associations between two classes in the model. Drag from one class to the other. Open the association specification by double-clicking on it.

On the "General" tab, fill in the role name for each of the two roles (classes) in the association. The name of the particular role is the verb-phrase that should follow the class name when the association is spoken. Thus, if "Patient is a role of Person," then, the name "is_role_of" is placed in the role box for the role that the "Patient" class fulfills.

Enter the description of the Association on the "General" tab. There are no properties, except history, needed for an association.

Enter the multiplicity of each end of the association on the "Role detail" tab for each role. Select the multiplicity for the class that fulfills the role in the "Cardinality" box. (Regrettably, the Rose tool has not yet switched to the term "multiplicity," and continues to use "cardinality.")

11.5.7 Composite aggregations

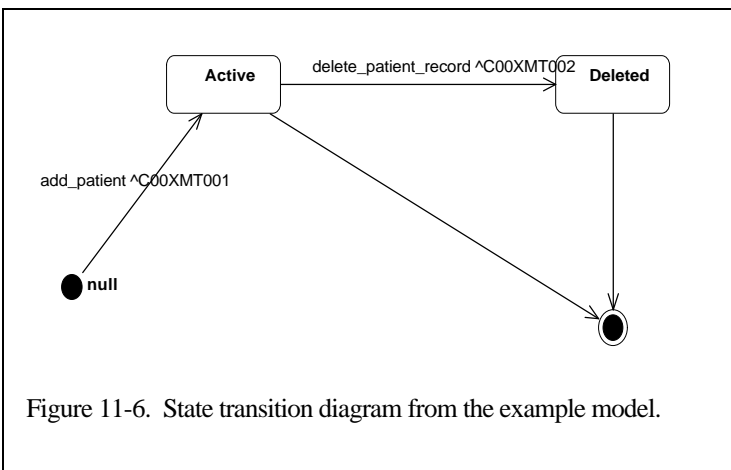
The following attributes apply to a composition relationship between two classes: the composite-to-part name, the part-to-composite name, the multiplicity for the part class, and the description.

11.5.7.1 Creation with Rose

Use the Aggregation tool from the toolbar to create the relationship. Drag from the part class to the related composite class. You need not name the relationship, but may do so if desired. The repository will assign default names of "has_parts" and "is_part_of" if you do not enter a name. Use the "Role A detail" tab to specify the multiplicity of the part class. The multiplicity of the composite class is always 1..1.

The relationship does not require any properties, except for the history data.

11.5.8 States



Create a state transition diagram for each subject class. To do this, right-click on the class in the Rose browser, and select (Open State diagram). This will create and open a state diagram for the class. Note that you should complete the **StateAttribute** and **IsSubjectClass** properties in the class specification for every subject class. Figure 5 provides an example of a state diagram.

The following attributes apply to a state: name and description.

11.5.8.1 Creation with Rose

As a first step create the start state and end state on the diagram using the appropriate tools. You need not name the end state, but you should name the start state "null." Add and name each state for the diagram. The state description should be completed, but it states carry no properties, except the history property.

11.5.8.2 Creating concurrent and nested states

A subject class may be in multiple states concurrently. UML allows only a single state machine (state diagram) for any given class. UML does allow the modeler to nest a state within another state. This affords the ability to create concurrent, independent, state-transition paths within a single state a machine.

There are two ways to create a nested state. One way is to place the cursor on top of the super-state when the sub-state is first defined. Alternatively, you may drag a state on the diagram and drop on its super-state. In either case, the sub-state appears within the boundaries of the super-state. All states in a single state machine must have unique names.

There is a variety of ways to create and manipulate concurrent states in a nested state hierarchy. The simplest is to define a single super-state that encompasses all of the other states. If a class is in one of the sub-states, it is also in the super-state. Thus, independent state transitions from the super-state to different sub-states create concurrent pathways. Discussion of other alternative strategies for concurrent states is beyond the scope of this chapter.

11.5.9 State transitions

The following attributes apply to state transitions: label (or name), description, and the identifier of the trigger event that causes the transition.

11.5.9.1 Creation with Rose

Use the state transition tool to create each state transition. Drag from the previous state to the subsequent state for each transition. There is no difference between nested and non-nested states as they relate to creating transitions. The transition has a description and a history property. In addition, you may link each state transition to one trigger event. Capture this linkage by entering the trigger event ID in the "Send event" box on the "Detail" tab of the state transition specification. This causes a display of the trigger event ID on the state transition diagram following the name of the transition. The name of the transition may be, but need not be, the name of the causing trigger event.

Use caution if the state transition diagram calls for multiple self-transitions. A self-transition is a state transition that begins and ends on the same state. Although Rose will allow the creation of multiple self-transitions, it will overlay all of these transitions on top of each other in the diagram. This makes it very difficult to select and label the individual self-transitions.

11.5.10 Data types

Classes with the Data_type stereotype model data types in Rose. Attributes that must be included for a data type include: name, description, data type symbol, its parameters (if it is generic) and indicators as to whether the data type is internal, primitive, generic, and/or an alias for another data type.

11.5.10.1 Data type symbols and names in Rose

By convention, when you construct data types from other data types, create the symbol using the "<" and ">" symbols. For example, a set of integer is SET<INT>. Also, modelers should include the data type symbol in the name entered for the data type. Do this by concatenating the symbol to the end of the data type using spaces and a colon (":") as the separator. For example, the name in rose for the integer data type is "Integer : INT"

11.5.10.2 Creating basic data type with Rose

Create data types on the diagram of one of the data type categories (stereotype of category). Create the data type with the class tool, and open its specification dialog to name it and to enter its description on the "General" tab. Select the Data_type stereotype in the specification. If the data type is internal, check the "Abstract" box on the "Detail" tab of the class specification. Instructions for capturing the rest of the attributes for a data type follow.

11.5.10.3 Creating generic data types with Rose

If the data type being defined is generic, first follow the steps in the preceding paragraph. Next, on the "General" tab of the specification, select "ParameterizedClass" in the "Type" drop down list. This selection causes a dotted parameter box to be added at the top right of the diagram symbol. Switch to the "Detail" tab.

For each parameter of the generic data type, right-click on the "Formal Arguments" box and select (Insert). For the argument name, provide the symbol for the generic parameter, usually one of the letters "T," "S" or "R." For the argument type, enter the code for the data type generalization that constrains the possible data types for the parameters. Examples are "DSCR" for discrete data types, and "ORD" for ordered data types.

If additional description is needed for the generic type parameter, add this as part of the description for the data type as a whole.

11.5.10.4 Qualifying properties

Regardless of whether the data type is generic or not, fill out the two data type properties - **isPrimitiveDT** and **DTsymbol**. If the data type is primitive, set **isPrimitiveDT** true. Enter the symbol for the new data type in **DTsymbol**.

11.5.10.5 Creating alias data types with Rose

If the data type you are creating is an alias, follow the steps above for creating a basic data type, and then add a single component to the data type. Make the component name "alias," and assign the data type for the alias component as the data type being aliased.

11.5.11 Data type components

The modeler creates components for composite and alias data types as attributes of the parent data type. Attributes of these components include their name, description, the data type for the component, whether it is mandatory or optional and whether it is by reference rather than direct.

11.5.11.1 Creation with Rose

Create data type components as attributes in the specification dialogue for the data type (class) of which they are a part. Record the component name and description in the usual fashion. You must provide a data type for each component. Record the data type in the "Type" box on the "General" tab of the component specification.

11.5.11.2 Qualifying properties

Properties on the HL7 tab of the component (attribute) specification dialog capture whether the component is mandatory or optional and whether it is by reference. If the component must appear in all instances of the data type, set **MandatoryInclusion** true. If the component is by reference, set **isReferenceDT** true.

11.5.12 Data type generalizations

At times, modelers will need to create data type generalizations to represent data types that may take on several, more-elemental forms.

11.5.12.1 Creation with Rose

Create generalizations in the "RIM_Datatype_Generalizations" category. Drag the classes to be generalized onto the "RIM_Datatype_Generalizations" diagram, if they are not already there. Create the new generalization data type as described above. For each of the subtypes, use the Generalization tool and drag from the subtype to the generalization. The MDF allows a data type to be a subtype of more than one generalization, unlike the generalization relationship for classes.

11.5.13 Generic data type instances

One goal of the methodology is to assure that each data type assigned to any of the RIM attributes is also defined in the RIM. In order to assure this completeness, the model must declare each of the instantiations of the generic data type.

11.5.13.1 Creation with Rose

By convention, instantiations of generic data type will be recorded in the category "RIM_Datatypes_Instances" which is a sub-category of "RIM_Datatypes_Generic." On the class diagram, create a new class, name it, and assign the Data_type stereotype. These data types will have names like "Interval_of_time : IVL<TS>". In addition to the name, two of the properties of the data type must also be

set. **DTsymbol** should include the full symbol (IVL<TS> in the previous example). **InstancedDTsymbol** should be completed with the data type assigned to the type parameter of the generic data type (TS in the previous example).

11.6 Interaction model

11.6.1 Model structure

A separate sub-category in Rose holds the interaction model -- the "Interaction_model" sub-category under the LogicalView. Committees may define their own categories under the Interaction_model. Application roles can be defined in any of these new categories, and interaction diagrams placed under the categories cause the interactions defined on those diagrams to "reside" in the category.

11.6.2 interaction model categories

Committee may define additional categories to application roles and interactions in whatever categories the committee finds convenient. For example, a committee may choose to group these elements according to their subject class. Interaction model categories carry the Interaction_model_category stereotype,. The attributes of an interaction model category are its name and description.

11.6.2.1 Creation with Rose

Create a new category by right-clicking on the Interaction_model category in the Rose Browser, and selecting (New:Package). Create nested categories by right-clicking on the parent category in the browser. Open the specification box for the new category and select the Interaction_model_category stereotype. You must provide a name for the new category that starts with "Cnn_" where Cnn is the identifier for the committee. There are no other constraints on the category name except those imposed by the requirement for uniqueness. Interaction model categories have no properties other than the history identifier.

For each created category, add a class diagram and/or a sequence diagram with the same name as the category. Do this by right-clicking on the category icon in the browser and selecting (New:Class diagram) or (New:Sequence diagram).

11.6.2.2 Qualifying properties

All interaction model categories must have the **RespComm_id** property set to the identifier of the responsible committee.

11.6.2.3 Putting application roles and interactions in defined categories

The presence of an application role on the class diagram for a given category documents that the application role is part of that category. Define each application role on one of the category diagrams at the lowest level of nesting. The modeler should drag the application roles to the higher-level diagrams to record their appearance at that level, too. The modeler can make an application role appear in as many categories as desired, simply by dragging them to the appropriate diagram.

You may place interactions in the categories of only a single category hierarchy. Unlike other elements, interactions can only appear on a single, defining diagram. Thus, they will be part of the category in whose sequence diagram they are defined, and will part of all parents (ancestors) of that category.

11.6.3 Application roles & Trigger Events

11.6.3.1 Model structure - application role hierarchy

The application roles are modeled as stereotyped classes in a class diagram. Under any interaction model category, create a class diagram of the same name. Figure 11-6 is an example, application role diagram. Place application roles in a generalization hierarchy. The top level of this hierarchy is a class named

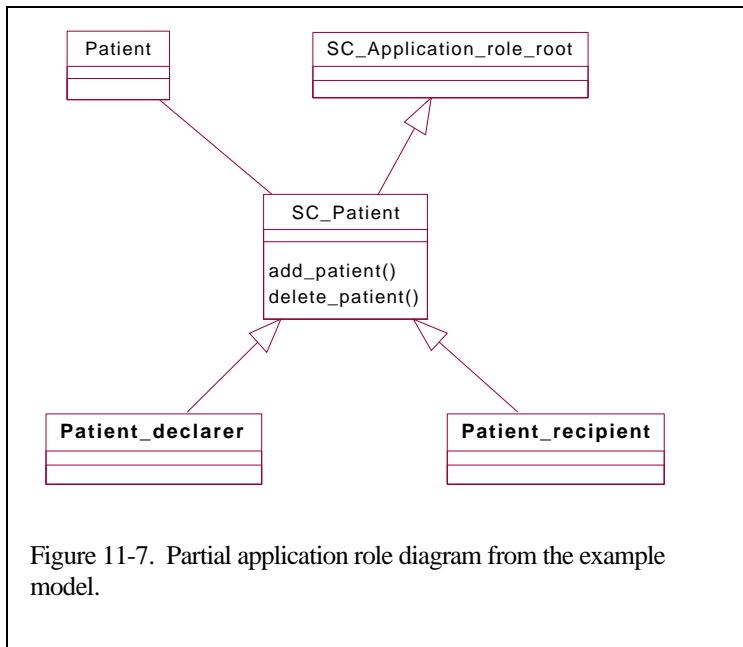


Figure 11-7. Partial application role diagram from the example model.

SC_Application_role_root. It does not use the Application_role stereotype. Use the "SC_" prefix for any class in the interaction model that is not an application role, and do not apply a stereotype to these classes.

The first level down in the hierarchy is a set of classes that represent linkages to each of the subject classes for which application roles are being defined. These should be named SC_<subject_class_name>. Use the generalization tool to link each of these to the parent SC_Application_role class.

11.6.3.2 Linkage to subject classes

The classes in the second level of the

hierarchy are linked to their subject classes using a three-step process:

- 1) Open the Rose browser, and drag the needed subject class from the information model area of the browser onto the application role class diagram. To minimize the visual impact on the diagram, right-click on the class and deselect "Show all attributes" and "Show visibility."
- 2) Select the Association tool from the toolbar. Drag from the second-level application role to its associated subject class. This step is sufficient to document the subject class relationship.
- 3) If you do not wish to have the subject classes shown on the diagram, select the subject class and press the delete key. This will delete the subject class and the association from this diagram, but Rose will retain the association.

By establishing the link at this level of the hierarchy, each of the application roles defined lower in the hierarchy will inherit this subject class linkage.

11.6.3.3 Application roles

Attributes that apply to an application role include: name, identifier and description.

11.6.3.3.1 Creation with Rose

Define the actual application roles at the third and lower levels of the hierarchy. Give them the Application_role stereotype. The stereotype distinguishes application roles from information model classes. Application roles also have stereotyped names drawn from their subject class like "<Subject_class_name>_recipient." The HL7 MDF meta-model requires an identifier for each application role that is distinct from its name. Record the identifier using the **ApplicationRoleID** property.

Link each application role to its parent in the hierarchy using the Generalization tool and enter the appropriate description for it.

Be sure to indicate that all of the upper-level, application role classes are abstract classes. Only the bottom of the hierarchy should be instantiable.

11.6.3.3.2 Qualifying properties

Each application role must include the **ApplicationRoleID** property to hold its unique identifier. Note that the HL7 property tab will also show properties for classes and data types. You can ignore these properties when defining an application role.

11.6.3.4 Trigger events (operations)

Attributes that apply to a trigger event are: name, identifier, description and dependency.

11.6.3.4.1 Relationship to application roles

Record trigger events in Rose as operations on the application roles. An interaction in Version 3 involves a trigger event, a sending application role, and a receiving application role. Capture trigger events as operations on the receiving application roles. This style is a convenience for recording HL7 interactions in Rose. It does not represent a functional relationship between trigger events and receiving role operations.

Because a trigger event may initiate multiple interactions, and because these interactions may be received by multiple application roles, a given trigger event may need to appear as an operation of several application roles. However, each trigger event only causes transitions in one subject class. Therefore, most of the receiving roles for a given trigger event will trace to a single subject class. Since the application roles are a generalization hierarchy, the trigger event can be defined at the second level (or some other intermediate level) of the hierarchy, from where it will be inherited by several application roles.

Thus the first step in defining trigger events in the model is to consider whether an intermediate level application role class should be used (or created) to hold the event. The upshot of all this is that the appropriate class to hold the trigger event is either:

- a) one of the application roles that receive an interaction that is triggered by this event, or
- b) a parent (generalization) of a role that receives an interaction triggered by the event.

Where there is a choice of roles to use, select the one that is highest in the hierarchy, or in the case of a tie, the one that feels right! In either case, choose a role that derives from the same subject class that the trigger event affects. The inheritance hierarchy assures that the trigger event signature appears as an operation of the lower-level, application roles. The hierarchy also provides a link from the event to its affected subject class.

11.6.3.4.2 Creation with Rose

Open the specification for the appropriate application role, and move to the "Operations" tab. Insert an operation whose name is the name of the trigger event being defined. Double-click on this entry to open its specification. Enter the event name and description on the "General" tab. The history identifier is the only property needed.

Switch to the "Detail" tab. Record the trigger event ID as an argument on the "Detail" tab. Give the argument the same name as the trigger event ID, assign the type "tid" to the argument.

If the trigger event has a defined "dependency," switch to the "Pre Conditions" tab of the specification. Document the dependency in the Pre Conditions text box.

11.6.3.4.3 Secondary copies of trigger event definitions

Because a trigger event may trigger multiple interactions, it may not be possible to record it as an operation in only a single place in the application role hierarchy. If it is necessary to document an already defined event as an operation of an application role in a different branch of the hierarchy, do the following:

- 1) Create a new operation and record the trigger event name as the operation name (as above)
- 2) Record the trigger event identifier as one of the arguments of the operation, but use the type "tidx."

You need not repeat the event description and dependency for this copy of the trigger event.

11.6.4 Interactions

Use one or more Rose Sequence Diagrams to capture the interactions. Right-click on the Interactions category in the browser, and select (New:Sequence diagram). Figure 11-7 shows an example of such a diagram.

The following attributes apply to an interaction: an identifier, the identifier of the message structure (within an HMD) that it sends, a description, and the responsibility of the receiver to initiate a follow-on interaction. In addition, you will link the interaction to one trigger event, one sending application role, and one receiving application role.

11.6.4.1 Diagram structure

The sequence diagram shows a set of application roles as columns headed by an object that is an instance of an application role. You add the interactions as arrows from one column to another.

11.6.4.2 Application role objects (columns)

Build the sequence diagram by laying out a set of columns. Use the Object tool to add a column. Select one of the defined, application-role classes in the "Class" box on the general tab of the object specification. The name of the object should be the same as the name of the application role. Any interaction that you specify on the diagram will require an application role column for both its sending and receiving roles. Create one column for each role needed. Note that the same application role may appear on multiple sequence diagrams, if desired.

11.6.4.3 Interactions

11.6.4.3.1 Creation with Rose

To create an interaction between different roles, use the "Object Message" tool and drag from the sending application role column to the receiving role column. . To create an interaction where the sending and receiving roles are of the same type, use the "Message to Self" tool and click on the appropriate application role column. Although each interaction is numbered, the sequence does not affect the definition of the interactions.

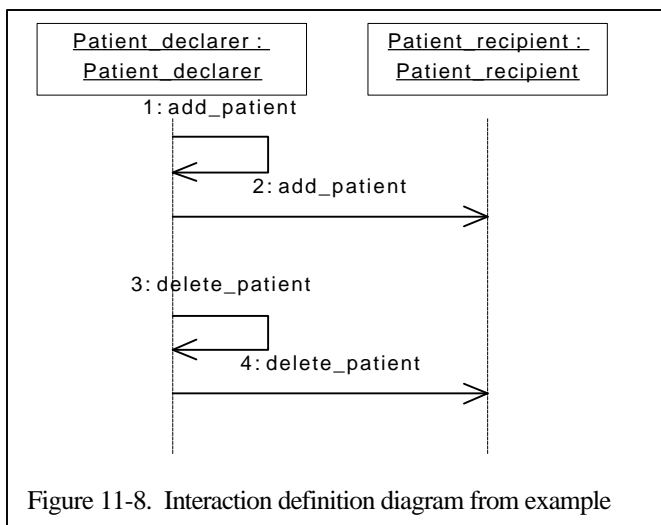


Figure 11-8. Interaction definition diagram from example

Once the interaction is created, double-click on it to open the specification. The "Name" box on the general tab provides a drop-down list that provides all of the trigger events defined to send messages to the receiving role. If you built the application roles and trigger events correctly, the trigger event that causes the interaction being defined should be on that list. Select it.

Use the documentation box on the general tab to capture the description for this interaction.

11.6.4.3.2 Defining interaction, HMD and message structure identifiers

Completion of the definition of an interaction will require that the modeler have unique identifiers for three elements - the interaction, the HMD in which the message structure is defined, and the message structure itself. Ultimately, HL7 will manage the definition of these identifiers to assure that they have a coherent structure and are unique across all committees.

When a committee first defines an interaction, however, it will not have the identifiers available. Thus, the committee may assign any identifier that begins with "Cnn_" where "Cnn" is the committee identifier. The committee is responsible for assuring the uniqueness of these temporary identifiers.

11.6.4.3.3 Qualifying properties

The properties box on the HL7 tab for an interaction contains five properties. Fill the first, **ID**, with the unique identifier of the interaction. In the next two, **HMD** and **MsgID**, insert the HMD identifier and the message structure identifier for the message that is sent by this interaction. The first three properties are mandatory. The fourth property is **RcvResp**. It captures the responsibility of the receiver to initiate a follow-on interaction. If this responsibility is to be defined, use the interaction identifier of the follow-on interaction as the value of the **RcvResp** property. The fifth property is the history property for the interaction.

11.7 Creating a Message Information Model (MIM)

Notice: This section of the chapter correctly describes how use Rose to create a MIM from the RIM. However, be advised that creating such a MIM has little value, other than to provide a MIM graphic for use by the Technical Committee. That is, as of October 15, 1999, there is no way to import a MIM into the repository and no functionality in RoseTree II to use such a MIM.

Moreover, RoseTree II will create a proper MIM as a by-product of creating an R-MIM. In the future, RoseTree II will take advantage of previously defined MIMs, but does not do so today.

11.7.1 Model Structure

An HL7 Message Information Model (MIM) is a proper sub-set of one version of the RIM. Modelers use a MIM to support the construction of one or more HMDs during message design. In order to assure the integrity of the MIM, the modeler should start with a complete, unaltered copy of the RIM in Rose. The modeler then creates a new category and diagram to hold the MIM, and establishes the MIM, with its constraints in that category.

Creation of a MIM is a destructive process in that the integrity of the RIM source file is usually damaged in the process. This is because the creation of the MIM may change the definition of classes and associations, and attributes may be deleted. Thus the Rose file that results from a MIM should be saved with a different name and should not be used for further modeling.

11.7.1.1 MIM (as a Subject Area)

The MIM is recorded as a category or subject area in Rose. Attributes that apply to a MIM are: an identifier, a name and the identifier of the RIM from which it is drawn.

11.7.1.1.1 Creation with Rose

Start with a Rose file that holds a clean, complete copy of the RIM. Create a new category for the MIM by right-clicking on the Information_model category in the Rose Browser, and selecting (New:Package).

Build the name of the category by appending the name of the MIM to a string formed by concatenating the committee's identifier with "MIM" using the underscore as a component separator. The MIM category for the example model is named "C00_MIM_Message_development_framework" where "Message_development_framework" is the name of the MIM. Use the documentation section of the category specification to capture the MIM description.

Also create a new class diagram for this category by right-clicking on the category in the browser and selecting (New:Class diagram). Give the diagram the same name as the category.

11.7.1.1.2 Alternate method of creation

The process of populating the MIM involves dragging classes from the Rose Browser onto the class diagram for the MIM. This may be tedious, particularly if the committee already has a DIM with the same or similar contents. If there is an existing category that is part of a clean, complete copy of the appropriate RIM, the modeler can build the MIM diagram starting from that base.

In the Rose Browser, "drag" the category to be used so that it is a sub-category of "Information_model." (Most DIMS are already at this level.) Rename both the category and its class diagram using the name format discussed above, and change the description of the category to be the description of the MIM.

11.7.1.1.3 Qualifying property

The **MIM_id** property is used to record the unique identifier of this MIM. This property is in the category specification. Although the identifier will be modified by HL7 to be unique across all HL7 MIMs, a preliminary version should be created by appending the committee identifier to the model ID of the RIM from which it is drawn. The RIM model ID can be found in the Model properties section from (Menu - Tools:Options...). The preliminary identifier for the MIM in the example model is "RIM0083C00." If the committee is creating several MIMs against a single version of the RIM, these MIMs can be further distinguished by adding a numeric or alphabetic suffix to the identifier described above.

11.7.2 Assembling the MIM

11.7.2.1 Adding classes and relationships

The MIM is represented by a class diagram that shows the classes, associations and relationships that comprise the MIM. If the needed classes and relationships are not already on the diagram, it is necessary to add them.

11.7.2.1.1 Adding classes to the MIM with Rose

Open the class diagram that will hold the MIM, and display the Rose Browser, as well. Drag each class that should be part of the MIM from the browser to the diagram. As each class is added, any relationships between the newly added class and classes that are already part of the MIM will be displayed. (These classes could also be added using (Menu - Query:Add classes...)).

11.7.2.1.2 Adding relationships to the MIM with Rose

At times, one or more of the MIM relationships may not be displayed, either because they were inadvertently dropped, or because they were not on a DIM diagram that was used as the basis for the MIM. All of the relationships between all pairs of classes on the MIM class diagram can be shown by selecting (Menu - Query:Filter relationships). In the dialog box that appears, select each of the "All" buttons, and then select OK. This operation will add any missing relationships and all of the self-relationships for any of the classes. It will also resurrect any relationships that were intentionally deleted, as outlined in the following step.

11.7.2.2 Removing extra classes, attributes and relationships from the MIM

Since a MIM is a proper sub-set of the RIM, it may be desirable to eliminate: classes that are on the diagram, but are not part of the MIM; selected attributes from classes that are part of the MIM; or relationships between classes that are part of the MIM.

11.7.2.2.1 Removing classes and relationships with Rose

Unwanted classes and relationships need only be removed from the diagram, not from the model. To remove one of these elements, select it on the diagram, and press the "DEL" (delete) key.

11.7.2.2.2 Removing attributes with Rose

You may remove an attribute in one of two ways. First, you can right-click on it in the browser and select Delete. This is a destructive delete that cannot be undone. Alternatively, you can open the specification dialog for the attribute. On the HL7 tab of the attribute specification, set the **DeleteFromMIM** property to true. This will drop the attribute from the MIM when it is processed, but it leaves the attribute on the MIM diagram and thus allows you to “undo” the deletion.

11.7.2.3 Changing constraints for the MIM

Once you have included the proper subset of elements on the MIM diagram, you should change any constraints that should be tightened in the MIM. Constraints that you may may be tighten include: the multiplicity of associations and composite aggregations, the repeatability of attributes, and the inclusion of attributes.

11.7.2.3.1 Changing constraints with Rose

Change association and aggregation multiplicities with the Role detail tabs of the relevant specification. Change the repeatability and inclusion parameters for attributes by altering the properties on the HL7 tab of the specification for the affected attribute.

11.8 Properties defined by HL7 for use in Rose

Table 1 provides a complete list of all of the properties that have been defined for use in HL7 modeling.

Property name	Purpose	Used for
zhxID	This component tracks the version history of each element of the model. It contains the unique element identifier assigned to each model element. The repository assigns values for this element. Modelers should not change these values or assign new ones, but they may copy them to indicate an element's historic predecessor.	All model elements
ApplicationRoleID	Holds the unique identifier of an Application Role (stereotype of class).	App Roles
Cardinality	Records the constraints on the cardinality of attributes as documented in the RIM and in MIMs.	Attributes
DeleteFromMIM	Used in constructing a Message Information Model to indicate which attributes to omit from the MIM.	Attributes
MandatoryInclusion	Indicates with a value of "True" whether the inclusion of an attribute in an HMD and in the messages derived from that HMD is mandatory. The default is not mandatory, and use of mandatory inclusion in the RIM is deprecated.	Attributes
MayRepeat	Indicates with values of "True" or "False" whether an	Attributes

Property name	Purpose	Used for
	attribute may repeat in an HMD. The default is non-repeating.	
Vocab_domain	Captures the identifier (name) of the vocabulary domain that constrains the values of coded attributes. This property is captured both for RIM attributes and in MIMs.	Attributes
Vocab_strength	Captures the strength of encoding for the elements of a coded attribute. This property is captured both for RIM attributes and in MIMs.	Attributes
V23_Datatype	This component can document the Version 2.3 datatype for an attribute that is related to or derived from data fields in HL7 Version 2.3.	Attributes
V23_Fields	This component provides a reference to the source Version 2.x field for an attribute that is related to or derived from data fields in HL7 Version 2.3 standard. Concatenate multiple values with commas, if multiple references to Version 2.x exist for an attribute.	Attributes
IsSubjectClass	Set true for classes that are subject classes.	Classes
StateAttribute	For classes that are subject classes, this component provides the name of the state attribute for the class. Only one state attribute component may appear for a given class.	Classes
DTsymbol	Holds the symbol for a defined data type (stereotype of class).	Data type
InstancedDTsymbol	Holds the data type assigned to the generic type parameter when an instantiation of generic type is recorded.	Data type
IsPrimitiveDT	Indicates that a data type definition is for a primitive data type (stereotype of class).	Data type
IsReferenceDT	Indicates that the type for a data type component (attribute of a data type stereotype of class) is found by reference.	Data type component
HMD	The identifier of the HMD from which the message structure for the message transferred by an interaction is drawn.	Interactions
MsgID	The identifier of the message structure within the HMD (above) that defines the message transferred by an interaction.	Interactions
RcvResp	This property holds the identifier of the follow-on interaction, when the receiving application role for an interaction has the responsibility to initiate a follow-on interaction.	Interactions
DevelopingCommittee	Holds the id of the HL7 committee developing the model like "C00."	Model
ModelDate	A text version of the last modified date formatted like "19970606"	Model
ModelDescription	Contains the textual description of the model.	Model
ModelID	Holds the unique identifier assigned to this model.	Model

Property name	Purpose	Used for
ModelName	Holds the formal name for the model	Model
ModelVersion	A text version of the version number like "V 30-08"	Model
Organization	This is the organization defining the model, "Health_Level_Seven"	Model
MIM_id	Used in a subject area category that holds a MIM. It provides the unique identifier for the MIM. The first portion of this identifier should be the ModelID of the RIM from which the MIM is derived.	Subject Area
RespComm_id	Captures the identifier of the responsible committee for all subject areas and categories.	Subject Area
EndState	The HL7 MDF says that leaf level use cases should be connected to a state transition of the subject class. The use case diagram provides the link to the subject class. The modeler must use three properties to identify the state transition -- this property and the two following ones named StartState and StateTransition . The first two are, as their names imply, the names of the starting state and ending state of the transition, and the third is the name of the transition.	Use cases
StartState	(See description of EndState above.)	Use cases
StateTransition	(See description of EndState above.)	Use cases
ID	Holds the unique identifier of a use cases, interactions and storyboards.	Various

Table 11-1. HL7-defined properties in Rose.

11.9 Overview of using RoseTree II and reviewing models

11.9.1 Functional Objectives

The capabilities within the RoseTree family of tools were developed to satisfy the following functional objectives:

- Provide support for the definition of Refined Message Information Models (R-MIM) and Hierarchical Message Descriptions (HMD).
- Support model migration and validation facilities.

The application supporting these functions builds a complete in-memory copy an HL7 model. This in-memory model uses a VB class instance for each element of the model, thus simplifying the programming tasks.

11.9.2 User Interface

The user interface establishes a consistent correspondence between work products and Menus in the application. Throughout the RoseTree family of tools, the "File" and "Edit" menus always have the following meanings:

- **"File"** refers to a complete HL7 product - a Model, a MIM, and R-MIM an HMD or a CMET. Thus, the user always loads or saves these structures through the functions of the "File" menu.
- **"Edit"** refers to actions performed on the elements of these structures - classes or attributes in a model; nodes or rows in a tabular R-MIM or an HMD.

The standard HL7 repository can hold one or many models. These may be versions of the RIM, or MIMs,

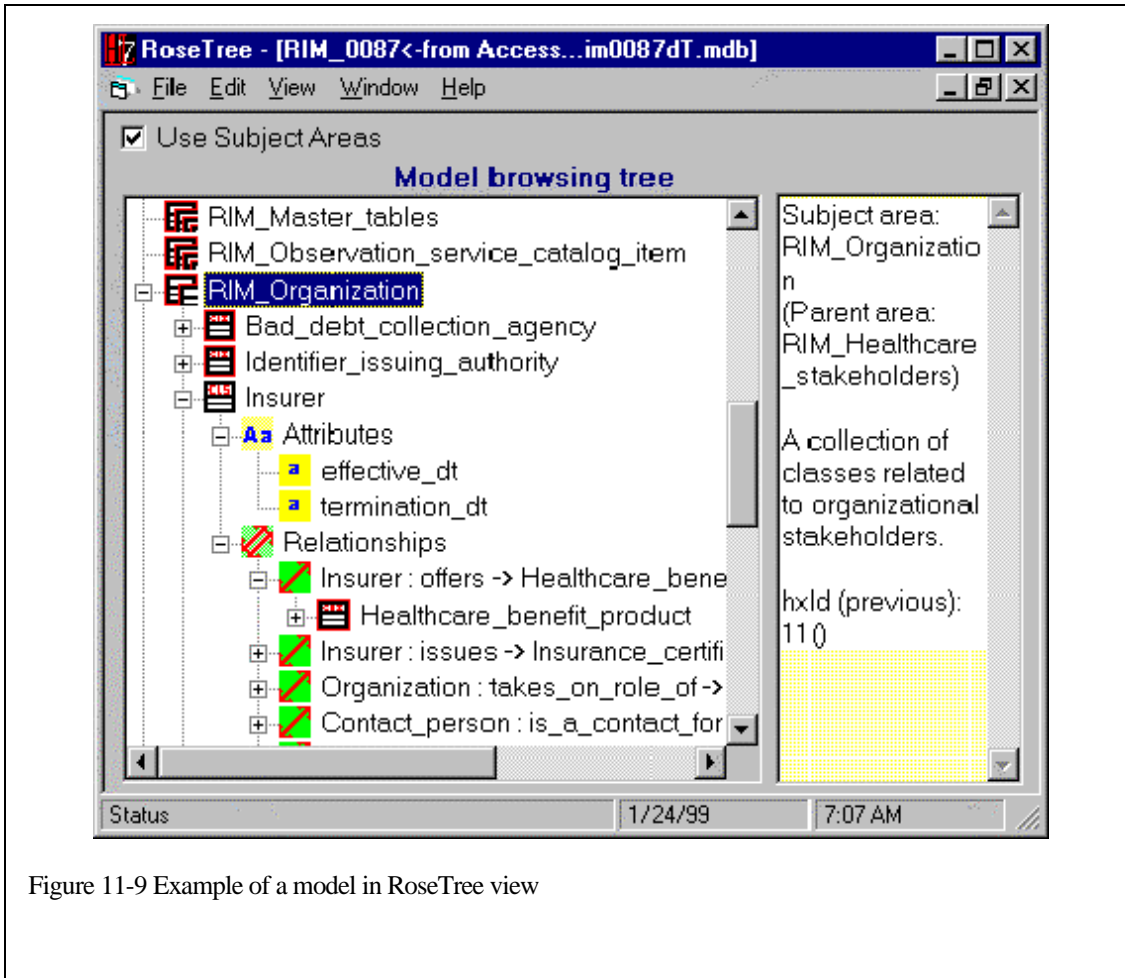


Figure 11-9 Example of a model in RoseTree view

R-MIMs, CMETs and HMDs defined against a given model. Thus when one "opens" a repository file (an Access mdb file) one will always be required to make a selection, even when there is only one model or structure in the repository. Rose files, on the other hand hold only a single model

11.9.2.1 Navigation

The RoseTree II tool uses multiple linked windows known as a "Multiple Document Interface" structure. As the user opens or creates each new structure, the program generates one or more new windows to hold the structure. If these windows have a logical relationship, as between a Model and a MIM, or a MOD and an HMD, the program logic will keep track of these relationships to assure consistent representation of information among them.

The user interface supports both key board and mouse-interactions. To the greatest degree possible, a user can initiate any action in the program either by a mouse-click or by a "hot-key" selection. This choice allows both browsing and "heads-down" editing.

The keyboard offers very rapid navigation of the tree view of models R-MIMs and HMDs. The core operations for this navigation are:

- **UP ARROW** and **DOWN ARROW**. These keys cycle downward through all expanded Node objects. Node objects are selected from left to right, and top to bottom. At the bottom of a tree, the selection jumps back to the top of the tree, scrolling the window if necessary.
- **RIGHT ARROW** and **LEFT ARROW**. These keys also step through expanded Node objects, but if the **RIGHT ARROW** key is pressed while an unexpanded Node is selected, the Node expands; a second press will move the selection to the next Node. Conversely, pressing the **LEFT ARROW** key while an expanded Node has the focus collapses the Node.
- **ENTER**. Pressing this key while a Node is selected, alternately expands or collapses a Node.
- **ANSI** key. If the user presses one of these keys, the focus will jump to the nearest Node that begins with that letter. Subsequent pressings of the key will cause the selection to cycle downward through all expanded nodes that begin with that letter.

11.9.2.2 Viewing Trees and Tables

11.9.2.2.1 Tree View of Models

The tree view of a model is a hierarchy that starts with one node for each of one or more classes. "Inside" each class are two selections - attributes and associations. The former expands to reveal each of the attributes of the class. The associations group expands to reveal each of the associations for a class, starting with the Gen-Spec relations and followed by the associations. Expanding each individual associations reveals the class at the other end of the association.

This class can, in turn, be expanded as above. The process continues *ad infinitum* (or until you run out of memory). Note that this tree is built as you select options. There is no "single" path to be followed. Thus, a given class may appear in many different places in the same tree, depending only upon the path taken to reach it. As each node of the tree is selected, its textual definition appears in a frame adjacent to the tree.

11.9.2.2.2 Tree View of R-MIMs

The tabular R-MIM (see HMD chapter) is essentially a two-level tree structure. The higher level of the R-MIM contains each of the classes and "clones" that are part of the R-MIM. The second level is made up of the attributes for the classes and the "half associations" that link a class to other classes in the R-MIM.

Remember that the R-MIM is an object diagram, rather than a class diagram. Thus, the R-MIM may contain more than one object drawn from a particular class in the model. Each of these appearances represents a semantically different object.

11.9.2.2.3 Table View of R-MIMs and HMDs

The mid-October 1999 version of RoseTree II provides the ability to save R-MIMs and HMDs in CSV (comma-separated variable) file format. HL7 has also provided an Excel spreadsheet whose macros allow the CSV files to be imported and formatted in the preferred HL7 style.

11.9.3 Viewing a Model

This section covers the steps and actions needed to open and navigate a model.

11.9.3.1 Opening a Model

Users can use RoseTree II to open models for viewing and navigation, regardless of whether the models are in an HL7 repository (Access database) or in a Rose file. The following describes the method for opening models from each source:

11.9.3.1.1 Load from the Repository

Use "File..Open" to select an Access database (mdb) file that holds an HL7 repository. You will be presented with a window that lists each of the models, and R-MIMs that the repository contains with check boxes next to each. Check only one of these. Click the "OK" button and the program will assemble the appropriate model in memory. The base level of the tree view will be presented when assembly is

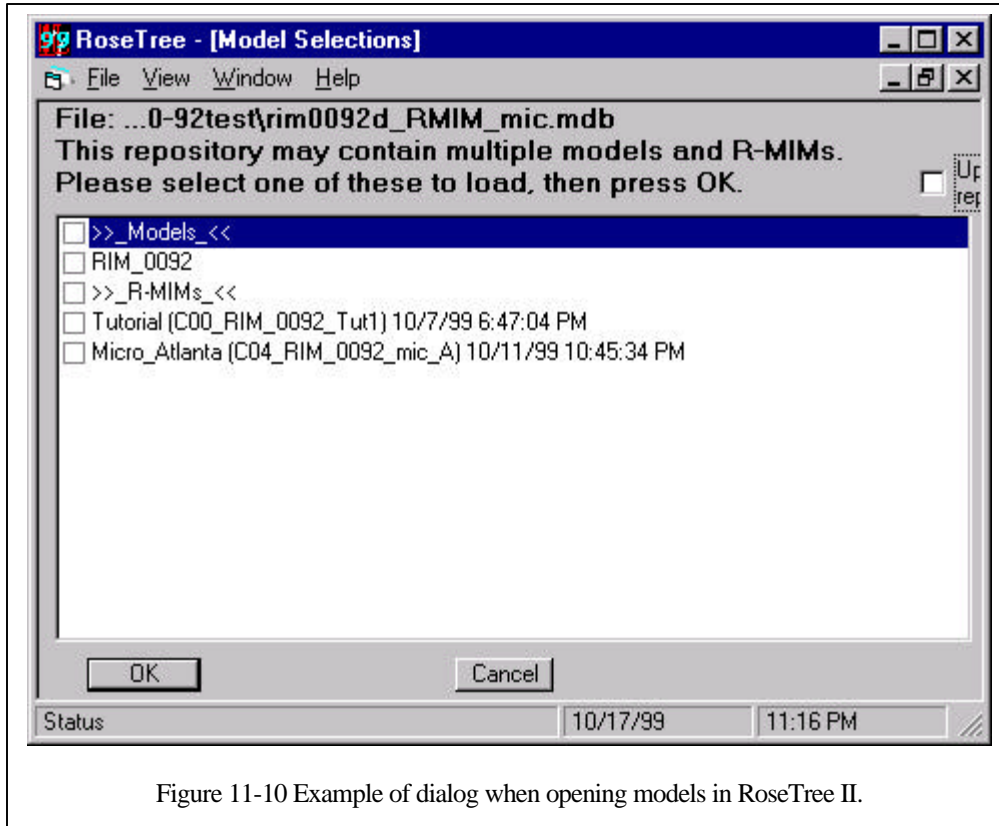


Figure 11-10 Example of dialog when opening models in RoseTree II.

complete. (See below for loading R-MIMs.)

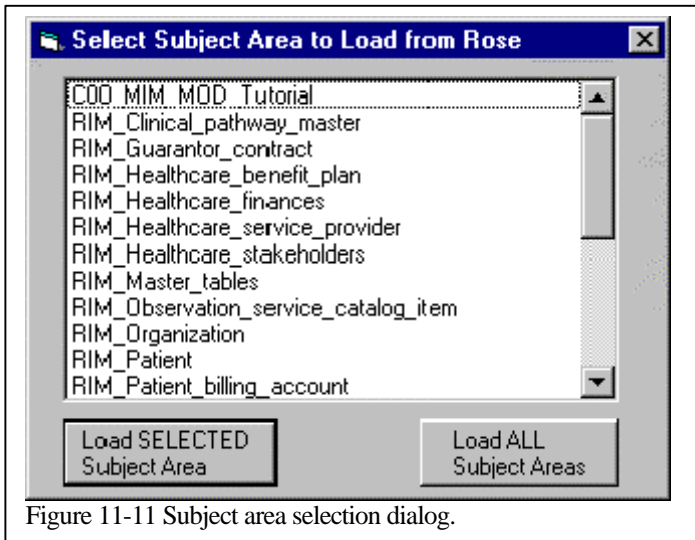


Figure 11-11 Subject area selection dialog.

11.9.3.1.2 Load from a Rose Model

This approach requires that the users have Rational Rose98 installed on their machine. Use "File...Open" to select a Rational Rose (mdl) file that contains the model you want. Do this selection even if Rose is already open with the model file you wish. The first thing the program does is to look for an open (running) version of Rose. If it finds one, it will check to see if the named file is the active file for Rose.

If the program does not find Rose running, or if Rose contains a different file, the program will cause Rose to load the appropriate file and proceed from there. If Rose is running with the

appropriate file, the user will be asked whether to use the loaded model (saves file load time) or whether the program should reload the file into Rose.

Once a model file has been selected and opened in Rose, RoseTree II will present a list of the Subject Areas in that model, along with the choice of loading a single subject area, or the whole model. Selecting a single subject area will load faster, and will limit the contents to the MIM in that subject area.

Extraction of a model from Rose takes five to ten times longer than loading the equivalent model from an Access repository. The program tracks its progress in loading the model on the screen. Once the extraction is complete, the program presents the tree view of the model.

11.9.3.2 Navigating a Model

By default, the initial representation of a model in Rose will list each of the subject areas of the model, in alphabetic sequence. (Note the subject area hierarchical relations are not displayed.) Selecting one of these subject areas will expand it to show all of the classes in that subject area.

If you prefer not to have the subject areas displayed, "uncheck" the "Use Subject Areas" check box and choose "File...Reload tree." This will display a list of all of the classes in the model beneath the root node for the model.

As noted above, the program builds the tree dynamically as you select and expand nodes of the tree. This enhances performance and avoids constraining your path through the model. If you have expanded the tree in numerous areas and to a great depth, performance may begin to suffer, and/or the display may become confusing. In this case, you can use the "File...Reload tree" option to reset the tree. If a class is selected when you choose this option, the program will deselect the "Use Subject Areas" option, it will reload the tree with all classes, and it will scroll the tree to assure that the selected class is visible in the frame.

11.10 Building an R-MIM in RoseTree

The first step in creating an HMD is to construct the R-MIM that defines the class and type structure to be used in the HMD and that lists the associations that may be walked in creating the HMD. You may either build an R-MIM from "scratch," starting with the information model, or you can load an existing R-MIM from a repository and extend or modify it, as needed.

11.10.1 Building an R-MIM for the First Time

The following steps apply if you are building an R-MIM from scratch.

11.10.1.1 Select a MIM

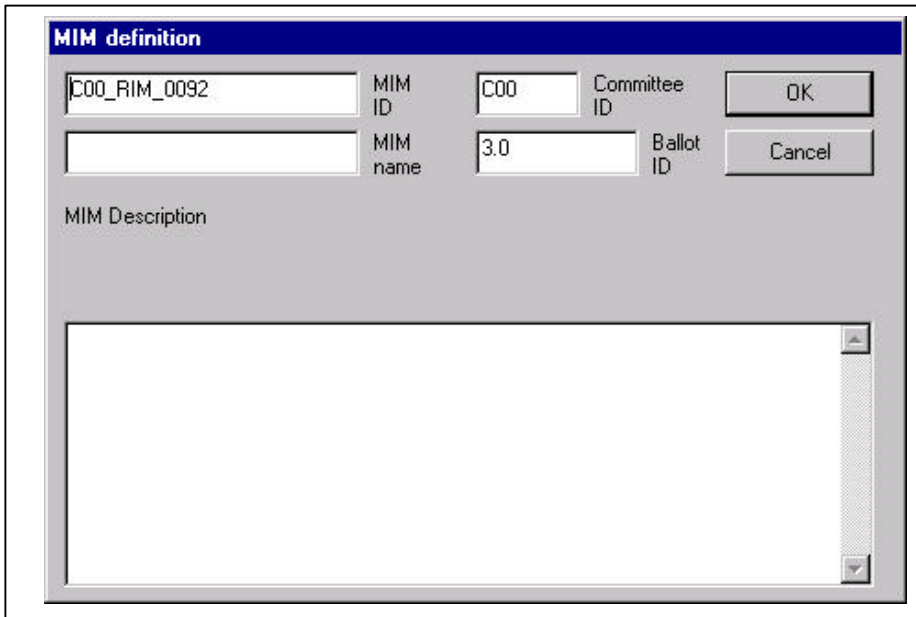
In the future (post 10/99) RoseTree II will allow you to select a MIM from the repository or a subject area to be used as a MIM. For now, however, users must build their R-MIMs by growing the diagram from some initial class.

11.10.1.2 Select the Initial Class

Select one of the classes of the MIM or RIM and choose "Edit...Start new RMIM."

In truth, at this point you are also starting a new MIM. Therefore, the program will present two dialog boxes asking for the defining parameters for the MIM and R-MIM.

11.10.1.3 MIM parameters



The parameters that define the MIM are entered in a dialog box like that in . These parameters are the **MIM_ID**, a unique identifier (maximum 16 characters) assigned by the committee; the assigned **CommitteeID**; a **MIM_name**; the HL7 ballot version (as **Ballot_ID**) that the MIM will be used to support; and a free-text **description** of the MIM.

Figure 11-12 MIM definition dialog.

11.10.1.4 R-MIM parameter

A second dialog box, similar to will be presented with the MIM data entered, and an open text box in which the user can enter the **RMIM_ID**. This is an arbitrary, committee-assigned identifier of 16 or fewer characters.

11.10.2 R-MIM definition window

Notice: As will rapidly become obvious, the rest of this chapter has not been written. As the HMD and R-MIM concepts evolved through the summer and fall of 1999, HL7 was barely able to provide the RoseTree II tools, much less document the use of these tools.

The section headings that follow indicate the capabilities of RoseTree II. As documentation is completed, it will be published separately from the MDF in order that users of RoseTree II can take advantage of it.

Prior to the publication of the next edition of the MDF, readers should expect full documentation for RoseTree II as well as significant growth in the capabilities of that tool.

- 11.10.2.1 Graphic representation of R-MIM nodes
- 11.10.2.2 Setting node parameters
- 11.10.2.3 State of R-MIM nodes – active, inactive, hidden
- 11.10.2.4 Changing R-MIM display options
- 11.10.2.5 Re-ordering nodes

11.10.3 R-MIM node types

- 11.10.3.1 Class nodes
- 11.10.3.2 Attribute nodes
- 11.10.3.3 Association nodes
- 11.10.3.4 Sub-component and item nodes

11.10.4 Adding classes to the R-MIM

11.10.5 Cloning classes in an R-MIM

11.10.6 Controlling association linkages

11.10.6.1 Caveats to the Process

11.10.6.1.1 Mouse Only

In the current version of the program, the program manages the option list using a list view control. This requires you to make all selections with the mouse. If this proves to be limiting, alternatives that allow key-board selection will be considered for (distant) future releases.

11.10.6.1.2 Pop-up Cardinality Menu

Whenever you select an association or an attribute as an option, the program immediately presents a pop-up menu from which you should select the multiplicity of the item just selected. This "non-standard" interface feature speeds selections. You may opt to "Cancel" at this point. If you click anywhere but on the pop-up menu, this action is also treated as a "Cancel."

11.10.6.1.3 Canceling a Multiplicity Selection

If you select "Cancel" from the pop-up menu for an attribute, the attribute will not be added and the option for that attribute will remain available.

- 11.10.7 Saving an R-MIM**
 - 11.10.8 Opening a Saved R-MIM from the Repository**
 - 11.10.9 Editing a MOD**
-

11.11 Constructing an HMD from an R-MIM

- 11.11.1 Select root class**
 - 11.11.2 HMD parameters**
 - 11.11.3 Walk the walk**
-

12. Example_model_for_MDF

This model is Copyright by Health Level Seven

Identifications:

Organization: Health Level Seven

Version: V 0-84x5 19990422

ModelID: RIM_0084X5

Developed by: Modeling and Methodology

Description of: **Example model for MDF**

Updated for inclusion in MDF-99.

Changes by Beeler & Rishel.

Subject Areas for: Example_model_for_MDF

Subject Area: C00 MIM Message development framework

This subject area holds the Message Information Model for the Example model of the MDF,

Contains classes: **Encounter_practitioner**
 Individual_healthcare_practitioner
 Inpatient_encounter
 Patient
 Patient_admission
 Patient_billing_account
 Patient_encounter
 Person
 Stakeholder
 Stakeholder_identifier

Subject Area: RIM Healthcare finances

A collection of subject areas related to the financial aspects of Healthcare.

Contains classes: **Patient_billing_account**

Subject Area: RIM Healthcare service provider

A collection of classes related to Healthcare service providers.

Contains classes: **Encounter_practitioner**
 Individual_healthcare_practitioner

Subject Area: **RIM Healthcare stakeholders**

A collection of subject areas related to healthcare stakeholders.

Contains classes: **Encounter_practitioner**
 Individual_healthcare_practitioner
 Patient
 Person
 Stakeholder
 Stakeholder_identifier

Subject Area: **RIM Patient**

A collection of subject areas related to patients.

Contains classes: **Patient**

Subject Area: **RIM Patient billing account**

A collection of classes related to the patient billing account.

Contains classes: **Patient_billing_account**

Subject Area: **RIM Patient encounter**

A collection of classes related to patient encounters.

Contains classes: **Inpatient_encounter**
 Patient_admission
 Patient_encounter

Subject Area: **RIM Patient encounters**

A collection of subject areas related to patient encounters and patient services

Contains classes: **Inpatient_encounter**
 Patient_admission
 Patient_encounter

Subject Area: **RIM Person**

A collection of classes related to person stakeholder.

Contains classes: **Person**

Subject Area: **RIM Stakeholder**

A collection of classes in related to stakeholders.

Contains classes: **Stakeholder**
 Stakeholder_identifier

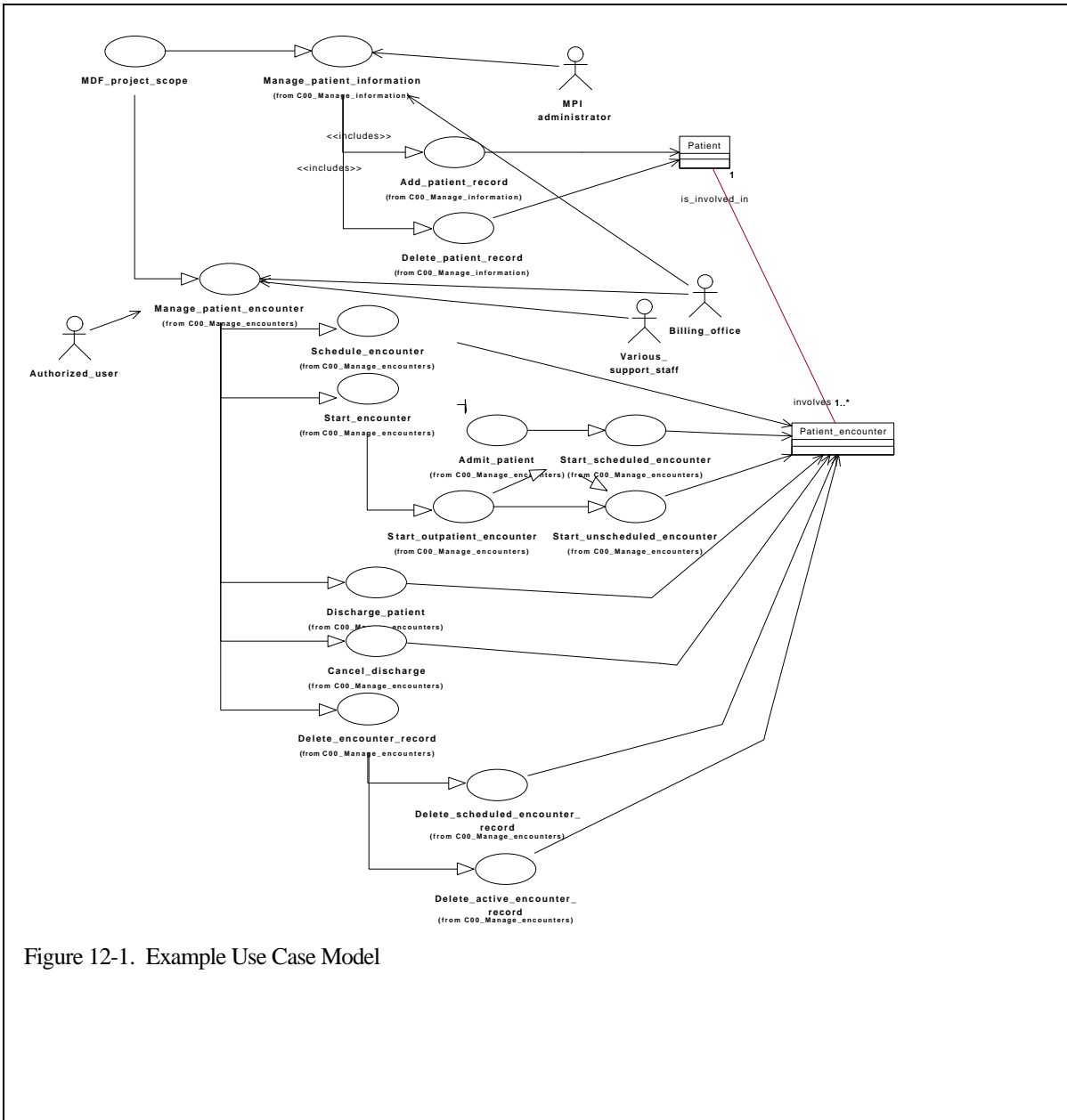


Figure 12-1. Example Use Case Model

Use case model in: **Example_model_for_MDF**

Actors in: **Example_model_for_MDF**

Actor: Authorized user

The individual responsible for managing encounter information.

Actions taken by: **Authorized user**

Manage_patient_encounter (C00XM005)

Actor: Billing office

The staff of the organization's billing office.

Actions taken by: **Billing office**

Manage_patient_encounter (C00XM005)

Manage_patient_information (C00XM001)

Actor: MPI administrator

The person responsible for the Master Patient Index.

Actions taken by: **MPI administrator**

Manage_patient_information (C00XM001)

Actor: Various support staff

Individuals in various ancillary departments of the organization.

Actions taken by: **Various support staff**

Manage_patient_encounter (C00XM005)

Use cases in: **Example_model_for_MDF**

Use case: **Add_patient_record (C00XM002)**

When a person becomes interesting to the healthcare provider, they are recorded as a patient. This happens if the provider needs to record information about the patient, becomes responsible for the patient's care, or either provides or plans to provide services for a patient.

Is child of: **Manage_patient_information (C00XM001)**

Changes **Patient** :fm: **Null** :to: **Active (Add_patient)**

Use case: **Admit_patient (C00XM015)**

The inpatient encounter becomes active when a patient is admitted. If the encounter has not been previously scheduled, it can be created at the time of admission.

Is child of: **Start_encounter (C00XM007)**

Is parent of: **Start_scheduled_encounter (C00XM020)**
Start_unscheduled_encounter (C00XM021)

Use case: **Cancel_discharge (C00XM009)**

In some cases, discharge processing is carried out for a patient by mistake. When that happens, it is necessary to cancel the discharge in order to return the encounter to an active status.

Is child of: **Manage_patient_encounter (C00XM005)**

Changes **Patient_encounter** :fm: **Discharged** :to: **Active (Cancel_discharge)**

Use case: **Delete_active_encounter_record (C00XM011)**

If an encounter is activated by mistake, it must be deleted.

Is child of: **Delete_encounter_record (C00XM012)**

Changes **Patient_encounter** :fm: **Active** :to: **Deleted (Delete_active_encounter)**

Use case: **Delete_encounter_record (C00XM012)**

Delete erroneous encounter data, owing to an error in the original records.

Is child of: **Manage_patient_encounter (C00XM005)**

Is parent of: **Delete_scheduled_encounter_record (C00XM010)**
Delete_active_encounter_record (C00XM011)

Use case: **Delete patient record (C00XM003)**

In some cases, a person is erroneously established as a patient. When this happens it is necessary to delete the patient information.

Is child of: **Manage_patient_information (C00XM001)**

Changes **Patient** :fm: **Active** :to: **Deleted (delete_patient_record)**

Use case: **Delete scheduled encounter record (C00XM010)**

If an encounter is scheduled by mistake, it may need to be deleted.

Is child of: **Delete_encounter_record (C00XM012)**

Changes **Patient_encounter** :fm: **Scheduled** :to: **Deleted (Delete_scheduled_encounter)**

Use case: **Discharge patient (C00XM008)**

When the patient's treatment is ended, or the patient's condition no longer requires care, he or she is discharged, Discharging the patient ends the stay, and removes the patient from direct care.

Is child of: **Manage_patient_encounter (C00XM005)**

Changes **Patient_encounter** :fm: **Active** :to: **Discharged (Discharge_patient)**

Use case: **Manage patient encounter (C00XM005)**

Healthcare providers create inpatient and non-inpatient encounters. Providers offer non-inpatient care when the patient's visit to the provider is brief enough so that the patient does not stay overnight at the facility. Healthcare providers offer inpatient care when the patient needs to stay overnight or for a more extended period at the provider's premises. This is done for surgical procedures or in cases of severe illness. The inpatient encounter is created and manipulated to facilitate scheduling, delivery, and financially accounting for the care that is provided to an inpatient.

Is child of: **MDF_project_scope (C00XM000)**

Is parent of: **Schedule_encounter (C00XM006)**
Start_encounter (C00XM007)
Discharge_patient (C00XM008)
Cancel_discharge (C00XM009)
Delete_encounter_record (C00XM012)

Involves actions by: **Authorized_user**
Billing_office
Various_support_staff

Use case: **Manage patient information (C00XM001)**

The healthcare provider keeps track of patient information in order to consistently link all health and healthcare service related information for a person.

Is child of: **MDF_project_scope (C00XM000)**

Is parent of: **Add_patient_record (C00XM002)**
Delete_patient_record (C00XM003)

Involves actions by: **Billing_office**
MPI administrator

Use case: **MDF_project_scope** (C00XM000)

The Example Model project will create the messages needed to support the administrative management of encounters.

The following areas are excluded from consideration: · Patient accounting, billing and insurance processing · Order processing, results reporting, patient care activities · Management of the patient's movement from place to place and/or organization to organization within the hospital · non inpatient, non acute care settings

Is parent of: **Manage_patient_information** (C00XM001)
 Manage_patient_encounter (C00XM005)

Use case: **Schedule_encounter** (C00XM006)

An encounter is scheduled when the provider determines that a patient needs care, and is able to set a definite date for the patient to arrive to receive that care. The encounter must be related to a patient and to a provider. If the patient is unknown to the institution, the patient's record can be created at this point.

Is child of: **Manage_patient_encounter** (C00XM005)

Changes **Patient_encounter** :fm: **Null** :to: **Scheduled** (**Schedule_encounter**)

Use case: **Start_encounter** (C00XM007)

Actions that occur when the patient presents for care.

Is child of: **Manage_patient_encounter** (C00XM005)

Is parent of: **Admit_patient** (C00XM015)
 Start_outpatient_encounter (C00XM016)

Use case: **Start_outpatient_encounter** (C00XM016)

The outpatient encounter becomes active when a patient is admitted. If the encounter has not been previously scheduled, it can be created at the time of admission.

Is child of: **Start_encounter** (C00XM007)

Is parent of: **Start_scheduled_encounter** (C00XM020)
 Start_unscheduled_encounter (C00XM021)

Use case: **Start_scheduled_encounter** (C00XM020)

Start an encounter that has been scheduled.

Is child of: **Admit_patient** (C00XM015)
 Start_outpatient_encounter (C00XM016)

Changes **Patient_encounter** :fm: **Scheduled** :to: **Active** (**Start_encounter**)

Use case: **Start_unscheduled_encounter** (C00XM021)

Start an encounter that has not been scheduled.

Is child of: **Admit_patient** (C00XM015)
 Start_outpatient_encounter (C00XM016)

Changes **Patient_encounter** :fm: **Null** :to: **Active** (**Start_encounter**)

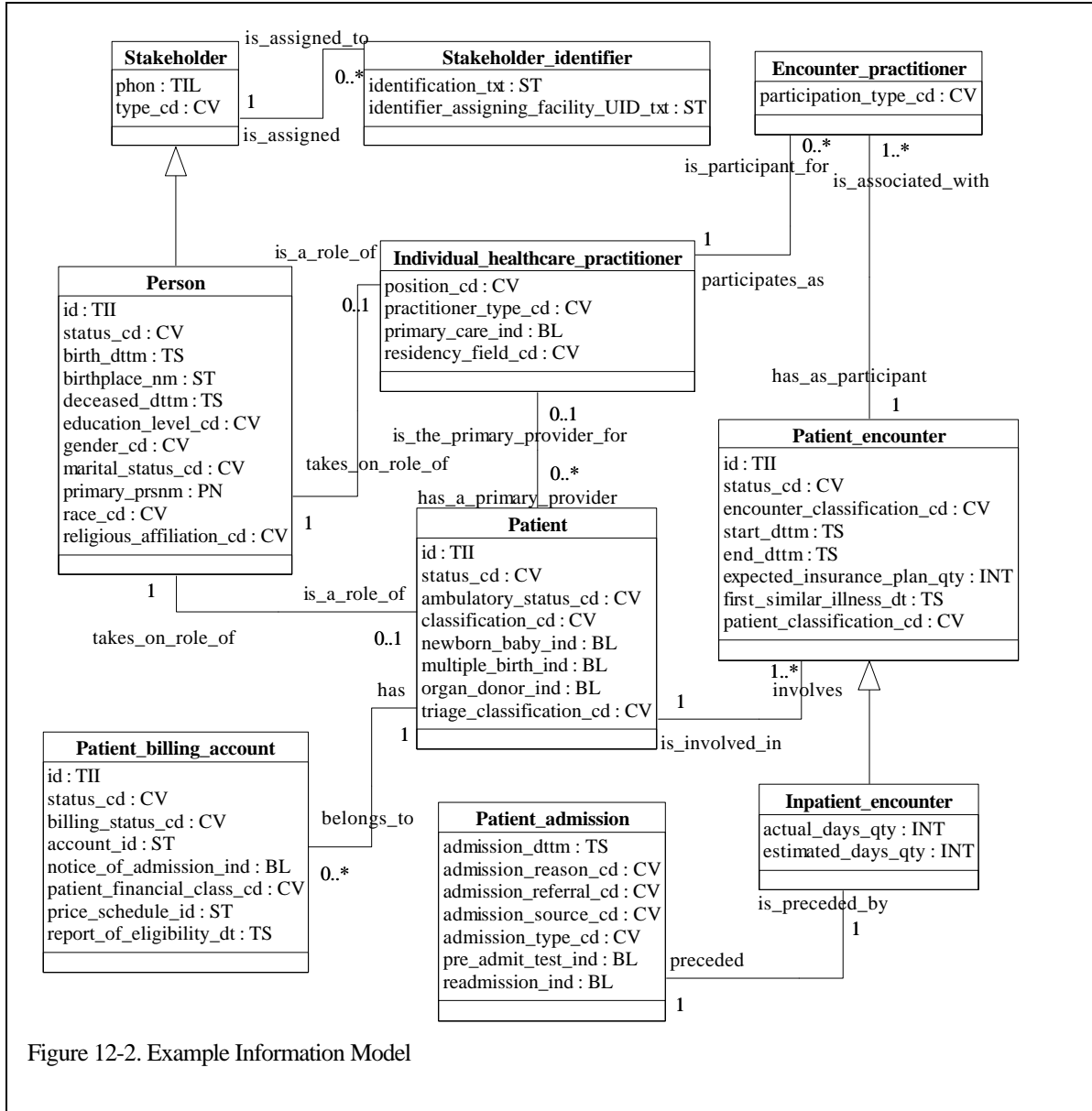
Information model in: Example_model_for_MDF
--

Classes in: **Example_model_for_MDF**

Class: Encounter_practitioner

Associated with: **Individual_healthcare_practitioner**
Patient_encounter

Description of: Encounter_practitioner



An association between a Healthcare practitioner and a patient encounter.

Associations for: **Encounter_practitioner**

is_participant_for (1,1) :: Individual_healthcare_practitioner :: participates_as (0,n)

is_associated_with (1,1) :: Patient_encounter :: has_as_participant (1,n)

Attributes of: **Encounter_practitioner**

participation_type_cd : CV

A code depicting the role of the type of participation the healthcare practitioner assumes in the encounter (e.g. attending physician, admitting physician, consulting physician, referring physician).

Class: **Individual_healthcare_practitioner**

Associated with: **Encounter_practitioner**
 Patient
 Person

Description of: **Individual_healthcare_practitioner**

A person in the role of a healthcare provider.

Associations for: **Individual_healthcare_practitioner**

participates_as (0,n) :: Encounter_practitioner :: is_participant_for (1,1)

is_the_primary_provider_for (0,n) :: Patient :: has_a_primary_provider (0,1)

is_a_role_of (1,1) :: Person :: takes_on_role_of (0,1)

Attributes of: **Individual_healthcare_practitioner**

position_cd : CV

A code indicating the position of a healthcare practitioner in an healthcare organization (e.g., head of department, trainee, hospital consultant, . . .).

practitioner_type_cd : CV

A code indicating the type of healthcare professional (e.g., medical doctor, nurse, pharmacist, laboratory worker, . . .).

primary_care_ind : BL

An indication that the healthcare practitioner is a primary care provider.

residency_field_cd : CV

The physician residency code.

Class: **Inpatient_encounter**

Subtype of: **Patient_encounter**

Associated with: **Patient_admission**

Description of: **Inpatient_encounter**

A patient encounter involving an admission to an inpatient facility.

Associations for: **Inpatient_encounter**

is_preceded_by (1,1) :: Patient_admission :: preceded (1,1)

Attributes of: **Inpatient_encounter**

actual_days_qty : INT

The number of actual days of an inpatient stay. The actual days quantity can not be calculated from the admission and discharge dates because of possible leaves of absence.

estimated_days_qty : INT

The estimated number of days in an inpatient encounter.

Class: **Patient**

Associated with: **Individual_healthcare_practitioner**
Patient_billing_account
Patient_encounter
Person

Description of: **Patient**

A person who may receive, is receiving, or has received Healthcare services.

Associations for: **Patient**

has_a_primary_provider (0,1) :: Individual_healthcare_practitioner :: is_the_primary_provider_for (0,n)

has (0,n) :: Patient_billing_account :: belongs_to (1,1)

is_involved_in (1,n) :: Patient_encounter :: involves (1,1)

is_a_role_of (1,1) :: Person :: takes_on_role_of (0,1)

Attributes of: **Patient**

ambulatory_status_cd : CV

Condition of a patient. (e.g., pregnant, hearing impaired, . . .).

classification_cd : CV

A classifying code for patients.

id : TII

multiple_birth_ind : BL

A indication as to whether the patient is part of a multiple birth.

newborn_baby_ind : BL

A indication that the patient is a newborn baby.

organ_donor_ind : BL

An indication that the patient is an organ donor.

status_cd : CV

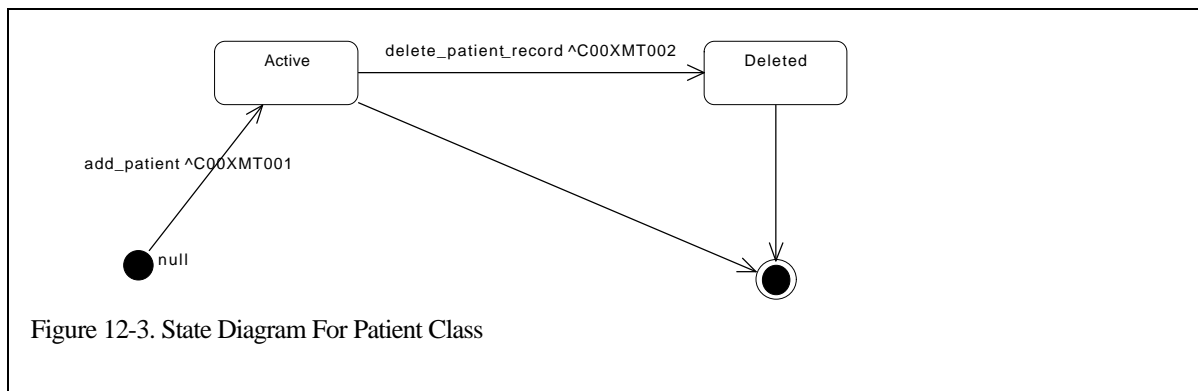
triage_classification_cd : CV

The triage classification of the patient.

States listed in: **classification_cd**

Active

A person for whom information is retained to support consideration of past, present, or future care.



The pre-condition for the state is valuation of patient identification information including at least one patient identifier, and a name for the patient. Normally other demographic information such as sex, date of birth, address will be valued as well.

Transitions from: **Active**

:to: **Deleted (delete_patient_record)**

Determination that a person has been erroneously added as a patient.

Triggered by: **delete_patient (C00XMT002)**

Deleted

A person who was erroneously identified as a patient.

The pre-condition for the state is determination that the patient record does not relate to a person who should be recorded as a patient. The termination date for the Medical Record Number - stakeholder ID where the identification text = "MR" will be set equal to or earlier than the current date.

null

Transitions from: **null**

:to: **Active (add_patient)**
Add a patient record.

Involves identification of a person and specification of that person as a patient.
Triggered by: **add_patient (C00XMT001)**

Class: Patient_admission

Associated with: **Inpatient_encounter**

Description of: Patient_admission
The beginning of an inpatient encounter.

Associations for: Patient_admission

preceded (1,1) :: Inpatient_encounter :: is_preceded_by (1,1)

Attributes of: Patient_admission

admission_dttm : TS
The date and time of the patient was admitted into an inpatient facility.

admission_reason_cd : CV
A code depicting the reason for the inpatient admission.

admission_referral_cd : CV
A code depicting the type of referral associated with this inpatient admission.

admission_source_cd : CV
A code indicating the source category associated with this inpatient encounter.

admission_type_cd : CV
A code indicating the circumstance under which the patient was or will be admitted.

pre_admit_test_ind : BL
An indication that pre-admission tests are required for this inpatient encounter.

readmission_ind : BL

An indication that the inpatient encounter is a readmission.

Class: **Patient billing account**

Associated with: **Patient**

Description of: Patient billing account

A financial account established for a patient to track the billable amount for services received by the patient and payment made for those services.

Associations for: Patient billing account

belongs_to (1,1) :: Patient :: has (0,n)

Attributes of: Patient billing account

account_id : ST

The unique identifier of a patient account.

billing_status_cd : CV

A code indicating the status of billing.

id : TII

notice_of_admission_ind : BL

A indicator documenting whether the insurance company requires a written notice of admission.

patient_financial_class_cd : CV

A code depicting a category for the source of payment.

price_schedule_id : ST

A reference to the unique identifier of the price schedule to be used for charges in the patient billing account.

report_of_eligibility_dt : TS

The date a report of eligibility was received.

status_cd : CV

Class: **Patient encounter**

Supertype of: **Inpatient_encounter**

Associated with: **Encounter_practitioner**
Patient

Description of: **Patient encounter**

An interaction between a patient and a Healthcare participant for the purpose of providing patient services or assessing the health status of a patient.

Associations for: **Patient encounter**

has_as_participant (1,n) :: Encounter_practitioner :: is_associated_with (1,1)

involves (1,1) :: Patient :: is_involved_in (1,n)

Attributes of: **Patient encounter**

encounter_classification_cd : CV

A classification of a patient encounter.

end_dttm : TS

The date and time that the patient encounter ends.

expected_insurance_plan_qty : INT

A count of the number of insurance plans that may provide Healthcare benefit coverage for this patient encounter.

first_similar_illness_dt : TS

The date the patient first experienced a similar illness. Used to determine pre-existing conditions

id : TII

A unique identifier assigned to the patient encounter.

patient_classification_cd : CV

A classification of the patient.

start_dttm : TS

The date and time that the patient encounter begins.

status_cd : CV

A code depicting the status of the patient encounter.

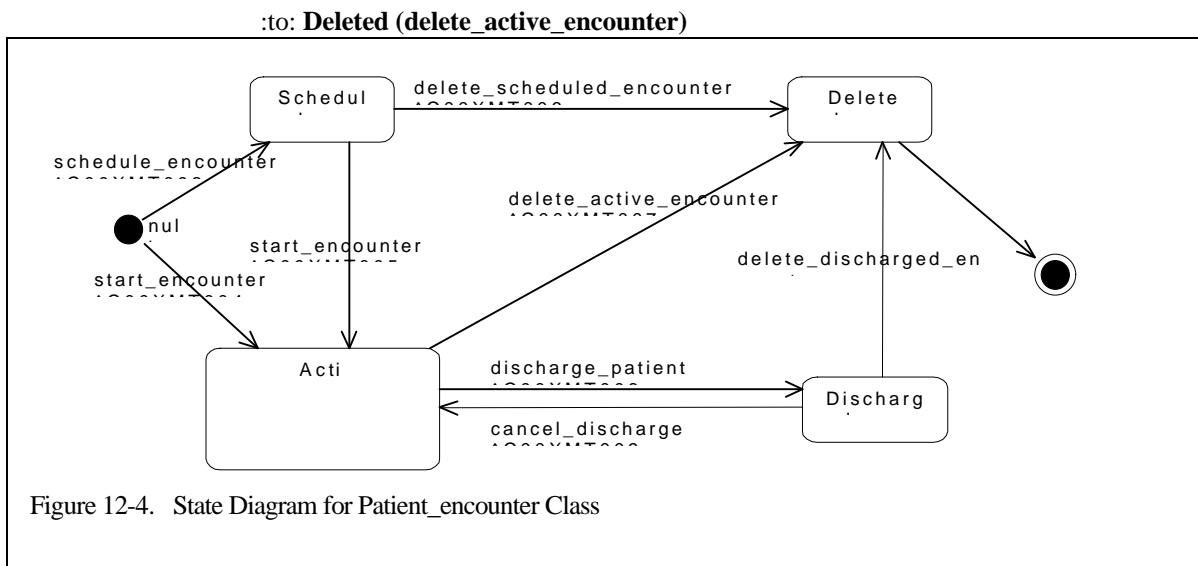
States listed in: **status_cd**

Active

The encounter has become active, whether or not it was scheduled, and the patient has started to receive care.

The precondition for the state is that a) the start_dttm for the patient encounter is earlier than, or equal to the current date time, and b) an encounter practitioner, with a participation_type_cd of "ADM" (admitting practitioner) has been assigned

Transitions from: **Active**



An active encounter is found to have been wrongly activated, and is deleted.

Triggered by: **delete_active_encounter (C00XMT007)**

:to: Discharged (discharge_patient)

A patient is discharged when their period of care by the institution is ended.

Triggered by: **discharge_patient (C00XMT008)**

Deleted

An encounter that is active or scheduled has been deleted because it has been determined that either the encounter was wrongly scheduled or activated.

Alternatively, a discharged encounter has been deleted.

The precondition for the state is that the status_cd for the patient encounter is set to "D" (deleted)

Discharged

A patient in an active encounter has been discharged because the patient has received the needed care, and either sent home or transferred to another facility.

The precondition for the state is that the end_dttm has for the patient encounter has been valued, and that it is earlier than or equal to the current date time.

Transitions from: **Discharged**

:to: **Active (cancel_discharge)**

A cancel discharge comes about when a previous discharge is determined to be erroneous.

Triggered by: **cancel_discharge** (C00XMT009)

null

Transitions from: **null**

:to: **Active (start_encounter)**

The encounter is created as an active encounter without having been previously scheduled.

Triggered by: **admit_patient** (C00XMT004)

:to: **Scheduled (schedule_encounter)**

The encounter is created as a scheduled encounter.

Triggered by: **schedule_encounter** (C00XMT003)

Scheduled

A new encounter is scheduled to become active at a specific date.

The precondition for the state is valuation of the start_dttm for the patient encounter. Other data for the encounter may be valued as well

Transitions from: **Scheduled**

:to: **Active (start_encounter)**

A scheduled encounter is activated.

Triggered by: **activate_scheduled_encounter** (C00XMT005)

:to: **Deleted (delete_scheduled_encounter)**

A scheduled encounter is found to have been wrongly scheduled, and is deleted.

Triggered by: **delete_scheduled_encounter** (C00XMT006)

Class: Person

Subtype of: **Stakeholder**

Associated with: **Individual_healthcare_practitioner**
Patient

Description of: Person

A type of stakeholder. An individual human being.

Associations for: Person

takes_on_role_of (0,1) :: Individual_healthcare_practitioner :: is_a_role_of (1,1)

takes_on_role_of (0,1) :: Patient :: is_a_role_of (1,1)

Attributes of: Person

birth_dttm : TS

The date and time of a person's birth.

birthplace_nm : ST

The place the person was born.

deceased_dttm : TS

The date and time that a person's death occurred.

education_level_cd : CV

The amount of education a person achieved.

gender_cd : CV

A code depicting the gender (sex) of a person.

id : TII

marital_status_cd : CV

A code depicting the marital status of a person.

primary_prsnm : PN

The primary name of a person.

race_cd : CV

A code depicting the race of a person.

religious_affiliation_cd : CV

A person's religious preference.

status_cd : CV

Class: Stakeholder

Supertype of: **Person**

Associated with: **Stakeholder_identifier**

Description of: Stakeholder

A person or organization that has an investment, share, or interest in healthcare.

Associations for: Stakeholder

is_assigned (0,n) :: Stakeholder_identifier :: is_assigned_to (1,1)

Attributes of: Stakeholder

phon : TIL

The phone number of a stakeholder.

type_cd : CV

A code depicting the type of stakeholder (e.g., person, organization, . . .).

Class: Stakeholder_identifier

Associated with: **Stakeholder**

Description of: Stakeholder_identifier

A unique identifier assigned to a person or organization.

Associations for: Stakeholder_identifier

is_assigned_to (1,1) :: Stakeholder :: is_assigned (0,n)

Attributes of: Stakeholder_identifier

identification_txt : ST

A unique identifier assigned to a stakeholder.

identifier_assigning_facility_UID_txt : ST

Universal Identifier (UID) for the stakeholder identifier assigning facility.

Rationale: required by components of V2.3 field

Interaction model in: Example_model_for_MDF
--

Trigger events in: **Example_model_for_MDF**

Trigger event: **activate_scheduled_encounter (C00XMT005)**

Dependency of: **activate_scheduled_encounter** (C00XMT005)
An encounter must have been scheduled, and there may be no current encounter for this patient.

Subject class: **Patient_encounter**

Causes transitions:
Patient_encounter :fm: **Scheduled** :to: **Active (start_encounter)**

Trigger event: **add_patient** (C00XMT001)

Dependency of: **add_patient** (C00XMT001)
Patient must not be recorded in the system.

Subject class: **Patient**

Causes transitions:
Patient :fm: **null** :to: **Active (add_patient)**

Trigger event: **admit_patient** (C00XMT004)

Dependency of: **admit_patient** (C00XMT004)
There may be no current encounter for this patient.

Subject class: **Patient_encounter**

Causes transitions:
Patient_encounter :fm: **null** :to: **Active (start_encounter)**

Trigger event: **cancel_discharge** (C00XMT009)

Dependency of: **cancel_discharge** (C00XMT009)
A patient must have been discharged from an active encounter and the encounter cannot have been deleted.

Subject class: **Patient_encounter**

Causes transitions:
Patient_encounter :fm: **Discharged** :to: **Active (cancel_discharge)**

Trigger event: **delete_active_encounter** (C00XMT007)

Dependency of: **delete_active_encounter** (C00XMT007)
The encounter must be active.

Subject class: **Patient_encounter**

Causes transitions:
Patient_encounter :fm: **Active** :to: **Deleted (delete_active_encounter)**

Trigger event: **delete_patient** (C00XMT002)

Dependency of: **delete_patient** (C00XMT002)
Patient must have been previously added.

Subject class: **Patient**

Causes transitions:
Patient :fm: **Active** :to: **Deleted** (**delete_patient_record**)

Trigger event: **delete_scheduled_encounter** (C00XMT006)

Dependency of: **delete_scheduled_encounter** (C00XMT006)
An encounter must have been scheduled and not been previously deleted or activated.

Subject class: **Patient_encounter**

Causes transitions:
Patient_encounter :fm: **Scheduled** :to: **Deleted** (**delete_scheduled_encounter**)

Trigger event: **discharge_patient** (C00XMT008)

Dependency of: **discharge_patient** (C00XMT008)
There must be an active encounter for this patient.

Subject class: **Patient_encounter**

Causes transitions:
Patient_encounter :fm: **Active** :to: **Discharged** (**discharge_patient**)

Trigger event: **schedule_encounter** (C00XMT003)

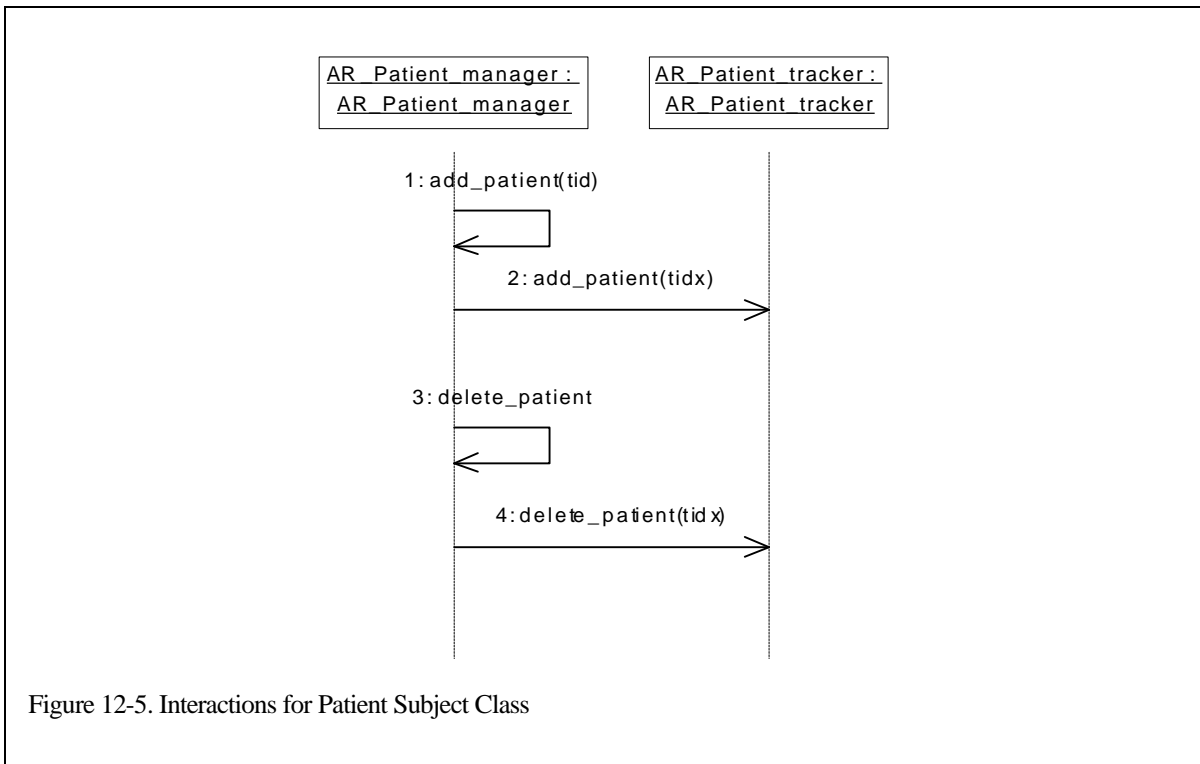


Figure 12-5. Interactions for Patient Subject Class

Subject class: **Patient_encounter**

Causes transitions:

Patient_encounter :fm: null :to: **Scheduled** (schedule_encounter)

Interactions in: **Example_model_for_MDF**

Interaction: C00XM101

Message structure: **C00XMH01 - C00XMM001**

Initiated by: **add_patient** (C00XMT001)

Sender: **Patient_manager** (C00XAR01)

Receiver: **Patient_manager** (C00XAR01)

Description:
to Patient_manager

Interaction: C00XM101a

Message structure: **C00XMH01 - C00XMM001**

Initiated by: **add_patient** (C00XMT001)

Sender: **Patient_manager** (C00XAR01)

Receiver: **Patient_tracker** (C00XAR02)

Description:
to Patient_tracker

Interaction: C00XM102

Message structure: **C00XMH01 - C00XMM002**

Initiated by: **delete_patient** (C00XMT002)

Sender: **Patient_manager** (C00XAR01)

Receiver: **Patient_tracker** (C00XAR02)

Description:
to Patient_tracker

Interaction: C00XM102a

Message structure: **C00XMH01 - C00XMM002**

Initiated by: **delete_patient** (C00XMT002)

Sender: **Patient_manager** (C00XAR01)

Receiver: **Patient_manager** (C00XAR01)

Description:
to Patient_manager

Interaction: **C00XM103**

Message structure: **C00XMH02 - C00XMM010**

Initiated by: **schedule_encounter** (C00XMT003)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_tracker** (C00XAR04)

Interaction: **C00XM104**

Message structure: **C00XMH02 - C00XMM011**

Initiated by: **admit_patient** (C00XMT004)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_manager** (C00XAR03)

Description:
to Encounter_manager

Interaction: **C00XM104a**

Message structure: **C00XMH02 - C00XMM011**

Initiated by: **admit_patient** (C00XMT004)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_tracker** (C00XAR04)

Description:
to Encounter_tracker

Interaction: **C00XM104b**

Message structure: **C00XMH02 - C00XMM013**

Initiated by: **admit_patient** (C00XMT004)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_archivist** (C00XAR05)

Receiver responsibility: **C00XM104a**

Description:
to Encounter_archivist

Interaction: C00XM105

Message structure: **C00XMH02 - C00XMM011**

Initiated by: **activate_scheduled_encounter** (C00XMT005)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_manager** (C00XAR03)

Description:
to Encounter_manager

Interaction: C00XM105a

Message structure: **C00XMH02 - C00XMM011**

Initiated by: **activate_scheduled_encounter** (C00XMT005)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_tracker** (C00XAR04)

Description:
to Encounter_tracker

Interaction: C00XM105b

Message structure: **C00XMH02 - C00XMM015**

Initiated by: **activate_scheduled_encounter** (C00XMT005)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_archivist** (C00XAR05)

Description:
to Encounter_archivist

Interaction: C00XM106

Message structure: **C00XMH02 - C00XMM012**

Initiated by: **delete_scheduled_encounter** (C00XMT006)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_tracker** (C00XAR04)

Interaction: C00XM108

Message structure: **C00XMH02 - C00XMM014**

Initiated by: **discharge_patient** (C00XMT008)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_manager** (C00XAR03)

Description:
to Encounter_manager

Interaction: C00XM108a

Message structure: **C00XMH02 - C00XMM014**

Initiated by: **discharge_patient** (C00XMT008)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_tracker** (C00XAR04)

Description:
to Encounter_tracker

Interaction: C00XM108b

Message structure: **C00XMH02 - C00XMM014**

Initiated by: **discharge_patient** (C00XMT008)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_archivist** (C00XAR05)

Description:
to Encounter_archivist

Interaction: C00XM109

Message structure: **C00XMH02 - C00XMM017**

Initiated by: **cancel_discharge** (C00XMT009)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_manager** (C00XAR03)

Description:
to Encounter_manager

Interaction: C00XM109a

Message structure: **C00XMH02 - C00XMM017**

Initiated by: **cancel_discharge** (C00XMT009)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_tracker** (C00XAR04)

Description:
to Encounter_tracker

Interaction: C00XM109b

Message structure: **C00XMH02 - C00XMM017**

Initiated by: **cancel_discharge** (C00XMT009)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_archivist** (C00XAR05)

Description:
to Encounter_archivist

Interaction: C00XM113

Message structure: **C00XMH02 - C00XMM011**

Initiated by: **delete_active_encounter** (C00XMT007)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_manager** (C00XAR03)

Description:
to Encounter_manager

Interaction: C00XM113a

Message structure: **C00XMH02 - C00XMM016**

Initiated by: **delete_active_encounter** (C00XMT007)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_tracker** (C00XAR04)

Description:
to Encounter_tracker

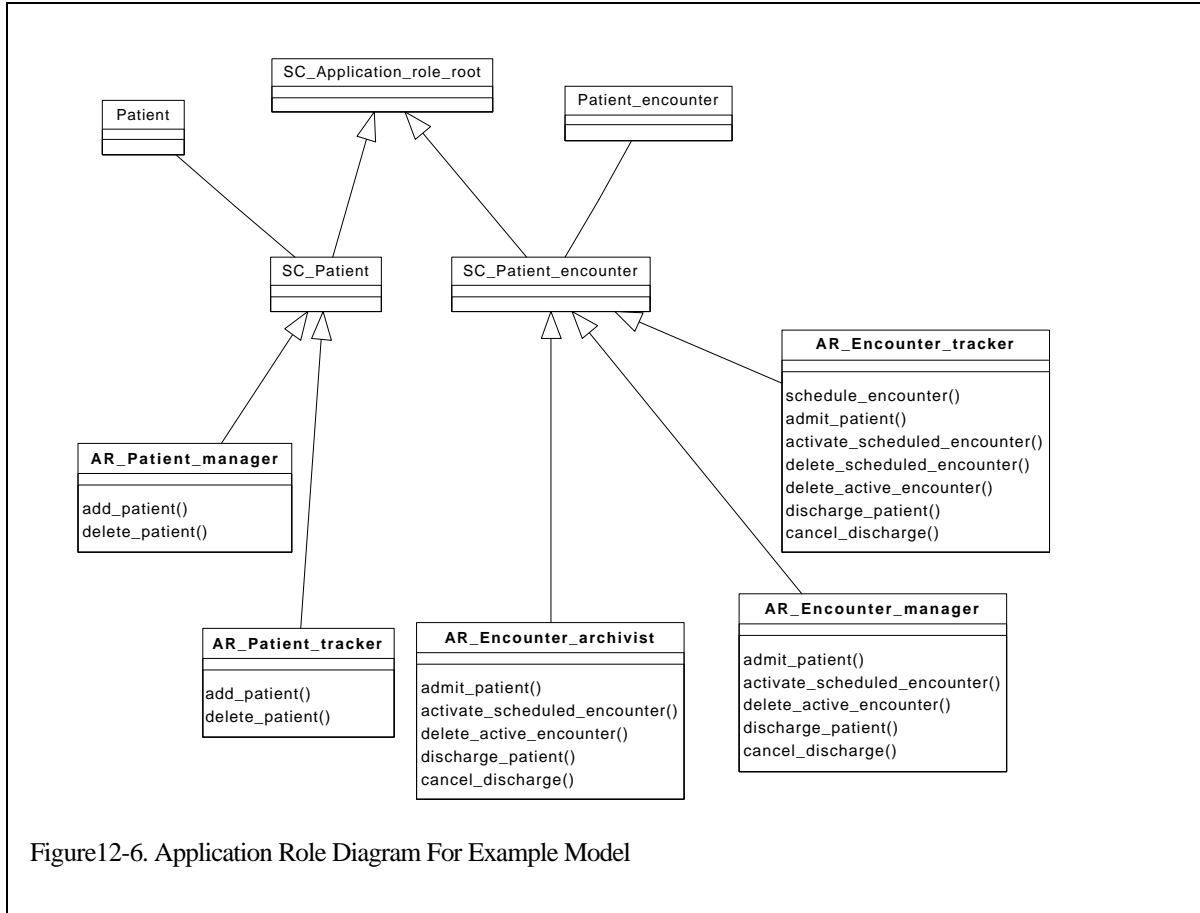


Figure12-6. Application Role Diagram For Example Model

Interaction: C00XM113b

Message structure: **C00XMH02 - C00XMM016**

Initiated by: **delete_active_encounter** (C00XMT007)

Sender: **Encounter_manager** (C00XAR03)

Receiver: **Encounter_archivist** (C00XAR05)

Description:
to Encounter_manager

Application roles in: **Example_model_for_MDF**

Application role: **Encounter archivist (C00XAR05)**

The encounter archivist application role supports all the state transitions for the subject class. It reflects the needs of applications that capture key information for a patient encounter in order to manage that data over a long period of time. It manages information related to a patient encounter that appears in the following classes: Patient, Person, Stakeholder, Stakeholder Identifier, patient encounter, and encounter practitioner classes.

Receives: **C00XM104b**
 C00XM105b
 C00XM108b
 C00XM109b
 C00XM113b

Application role: **Encounter manager (C00XAR03)**

The encounter manager application role supports all the state transitions for the subject class. It manages information related to a patient encounter that appears in the following classes: Patient, Person, Stakeholder, Stakeholder Identifier, patient encounter, encounter practitioner, inpatient encounter, and inpatient admission classes.

Sends: **C00XM103**
 C00XM104
 C00XM104a
 C00XM104b
 C00XM105
 C00XM105a
 C00XM105b
 C00XM106
 C00XM108
 C00XM108a
 C00XM108b
 C00XM109
 C00XM109a
 C00XM109b
 C00XM113
 C00XM113a
 C00XM113b

Receives: **C00XM104**
 C00XM105
 C00XM108
 C00XM109
 C00XM113

Application role: **Encounter tracker (C00XAR04)**

The encounter tracker application role supports all the state transitions for the subject class. It reflects the needs of applications that support the ordering, scheduling and performing of services for a patient during an encounter. It manages information related to a patient encounter that appears in the following classes: Person, Stakeholder Identifier, patient encounter, and encounter practitioner classes.

Receives: **C00XM103**
 C00XM104a
 C00XM105a
 C00XM106
 C00XM108a

C00XM109a
C00XM113a

Application role: **Patient manager** (C00XAR01)

The patient manager application role supports all the state transitions for the subject class, patient. It manages the information related to a patient that appears in the following classes: Patient, Person, Stakeholder, Stakeholder Identifier, Individual healthcare provider.

Sends: **C00XM101**
 C00XM101a
 C00XM102
 C00XM102a

Receives: **C00XM101**
 C00XM102a

Application role: **Patient tracker** (C00XAR02)

The patient tracker application role supports all the state transitions for the subject class. It reflects the needs of applications that capture person related information for a patient. It manages information for a patient that appears in the Person, Stakeholder, and Stakeholder Identifier classes.

Receives: **C00XM101a**
 C00XM102

Scenarios in: **Example_model_for_MDF**

Scenario: **Discharge screw up** (C00S3)

This scenario illustrates a reversed or canceled discharge.

Examples for: **Discharge screw up** (C00S3)

Example - a

The patient, John Q Smith was admitted to the hospital for an emergency appendectomy. The treatment was successful, and the patient was discharged 2 days after surgery. However, three hours after the discharge, nursing staff noted that, even though the patient had been held in the hospital due to their concern for his condition, administrative staff had processed a discharge. Therefore the discharge was canceled. Two days later, he was able to go home.

Interaction sequence: **C00XM104a**
 C00XM108a
 C00XM109a
 C00XM108a

Scenario: **Registering Santa Claus** (C00S2)

This demonstrates an erroneous patient registration.

Examples for: **Registering Santa Claus** (C00S2)

Example - b

A person identified as Mr. Santa Claus is registered as a patient on December 23rd. However, subsequent audit reveals that this is not an actual person, and that the identification information was supplied as a hoax.

Interaction sequence: **C00XM101a**
C00XM102

Scenario: Simple scenario (C00S1)

This is a basic patient registration, admit and discharge scenario involving a hospital ADT office and an ancillary department.

Examples for: **Simple scenario (C00S1)**

Example - a

Patient Tom Jones first registers with the ADT office which sends data about him to the ancillary department.

Subsequently, Jones is admitted and later discharged. Information about the encounter creation and the discharge are sent from the ADT office to the ancillary department.

Interaction sequence: **C00XM101a**
C00XM104
C00XM108

13. Specification of the HL7 MDF Components

This chapter contains the formal specification of the models that must be developed in carrying out the design processes of the MDF. This specification defines each element of each model, the attributes of those elements, and the associations between them. For example, in support of chapter 10, Creating Message Specifications, this chapter will define a Message Information Model, define the data to be recorded about a MIM (its attributes) and define the association between each element of the MIM and other elements in other models, such as the RIM.

The purpose of this specification is to collect in one place a complete and unambiguous definition of each of the work-products or deliverables that are the subjects of the MDF

13.1 Meta-model

One way to express such a specification is to define create a model of the information structures that the MDF requires. This is a model that describes models - a meta-model. The formalism used here is the same formalism that the MDF requires for documenting the health care domain in the information model. While the RIM has classes like "Patient" and "Person," the meta-model contains classes like "Model," "Hierarchical Message Description" and even "Class."

The expression of the meta-model follows the same graphical and literary expressions used for the RIM, formalisms that should be familiar to HL7 participants. As does the RIM, the meta-model contains subject areas to organize the material and data types. The latter serve to express the HL7 constraints on creating names for model elements like classes, descriptive text that includes open issues, and the like.

The following pages open with a table of contents for the meta-model followed by five pages of diagrams for the various models, and then the detailed literary expression, which is the formal specification.

13.2 Limitations of this meta-model

Because the meta-model is being developed simultaneously with major revisions to the MDF, there are places where the meta-model is not fully concordant with the MDF. In these cases, the chapters are correct, and the meta-model will change to match them. There are three places where required changes are known:

13.1.1 Vocabulary domain specifications

The meta-model for vocabulary domain specifications is correct, but incomplete. Chapter 7 includes the specification of tables that cannot be fully derived from the meta-model. These include notions of versioning, version dating, attribution of changes and relations to LOINC.

13.1.2 Use case generalization

Chapter 5 introduces three types of use case associations - generalization, "extends" and "includes." The current meta-model includes only the first of these.

13.1.3 Message design model

This chapter is still being tested. Thus changes to the equivalent meta-model sections – MIM, R-MIM, HMD and MET – should be expected.

Table of contents for: HL7_V3_Meta-Model

Diagrams for: HL7_V3_Meta-Model	
Identifications:	13-9
Subject Areas for: HL7_V3_Meta-Model	13-9
Classes in: HL7_V3_Meta-Model	13-16
Class: Actor	13-16
Class: Alias_name	13-17
Class: Application_role	13-18
Class: Association	13-20
Class: Attribute	13-21
Class: Attribute_domain_constraint	13-23
Class: Attribute_type	13-23
Class: Choice_branch	13-24
Class: Choice_MET	13-25
Class: Class	13-25
Class: Code_subsystem	13-27
Class: Code_system	13-28
Class: Coded_term	13-28
Class: Collection_MET	13-30
Class: Common_message_element_type	13-30
Class: Composite_aggregation	13-31
Class: Composite_data_type	13-32
Class: Composite_MET	13-32
Class: Data_type	13-33
Class: Data_type_alias	13-35
Class: Data_type_category	13-35
Class: Data_type_component	13-36
Class: Data_type_generalization	13-37
Class: Derivative_domain	13-38
Class: Domain_version	13-39
Class: Enumerated_domain	13-39
Class: Generalization_relationship	13-40
Class: Generic_type_parameter	13-40
Class: Hierarchical_message_description	13-41
Class: HL7_committee	13-42
Class: HMD_attribute_row	13-43

Class: HMD_class_row	13-44
Class: HMD_domain_constraint	13-44
Class: HMD_notation	13-44
Class: HMD_other_row	13-45
Class: HMD_relationship_row	13-45
Class: HMD_row	13-45
Class: Interaction	13-46
Class: Interaction_model_category	13-48
Class: Interaction_sequence	13-49
Class: LOINC_link	13-50
Class: Message_element_type	13-50
Class: Message_information_model	13-52
Class: Message_MET	13-54
Class: Message_row_control	13-54
Class: Message_type	13-55
Class: MET_component	13-56
Class: MIM_aggregation	13-58
Class: MIM_association	13-58
Class: MIM_attribute	13-59
Class: MIM_attribute_domain_constraint	13-60
Class: MIM_class	13-60
Class: MIM_generalization	13-60
Class: MIM_relationship	13-61
Class: MIM_state	13-61
Class: Model	13-62
Class: Primitive_data_type	13-64
Class: Primitive_MET	13-65
Class: Project	13-65
Class: Refined_message_information_model	13-66
Class: RMIM_attribute_domain_constraint	13-67
Class: RMIM_attribute_row	13-67
Class: RMIM_class_row	13-68
Class: RMIM_notation	13-69
Class: RMIM_note	13-69
Class: RMIM_other_row	13-70
Class: RMIM_relationship_row	13-70
Class: RMIM_row	13-71

Class: RMIM_state_row	13-74
Class: State	13-74
Class: State_transition	13-75
Class: Storyboard	13-77
Class: Storyboard_example	13-78
Class: Subject_area	13-78
Class: Subject_class	13-79
Class: Trigger_event	13-80
Class: Union_message_type	13-82
Class: Use_case	13-82
Class: Use_case_category	13-84
Class: Use_case_relationship	13-85
Class: Use_case_sequence	13-85
Class: V23_data_type	13-86
Class: V23_field_segment	13-86
Class: V23_fields	13-87
Class: V23_segments	13-88

Class: Version_MET	13-88
Class: Vocabulary_domain	13-89
Data type definitions in: HL7_V3_Meta-Model	13-92
Data type: Boolean : Boolean	13-92
Data type: CodedElement : CodedElement	13-92
Data type: CompoundHx : CompoundHx	13-92
Data type: Date : Date	13-93
Data type: DateTime : DateTime	13-93
Data type: DescriptiveText : DescriptiveText	13-93
Data type: Enumerated : Enumerated	13-93
Data type: IdentifierString : IdentifierString	13-94
Data type: Integer : Integer	13-94
Data type: MultiplicityString : MultiplicityString	13-94
Data type: NameString : NameString	13-95
Data type: String : String	13-95
Data type: VersionNumber : VersionNumber	13-95
Data type categories for: HL7_V3_Meta-Model	13-96

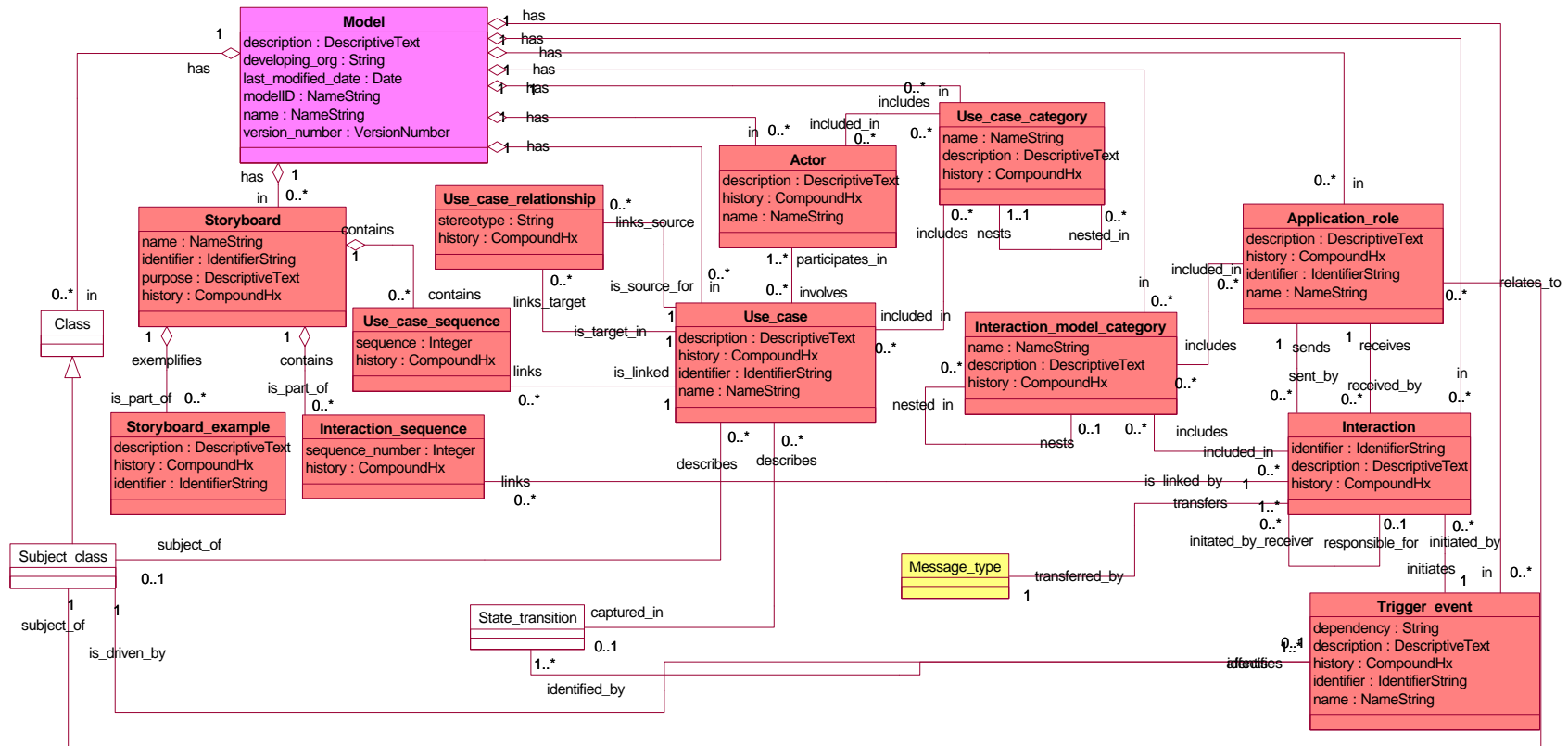


Figure 13-1. Use Case and Interaction Models

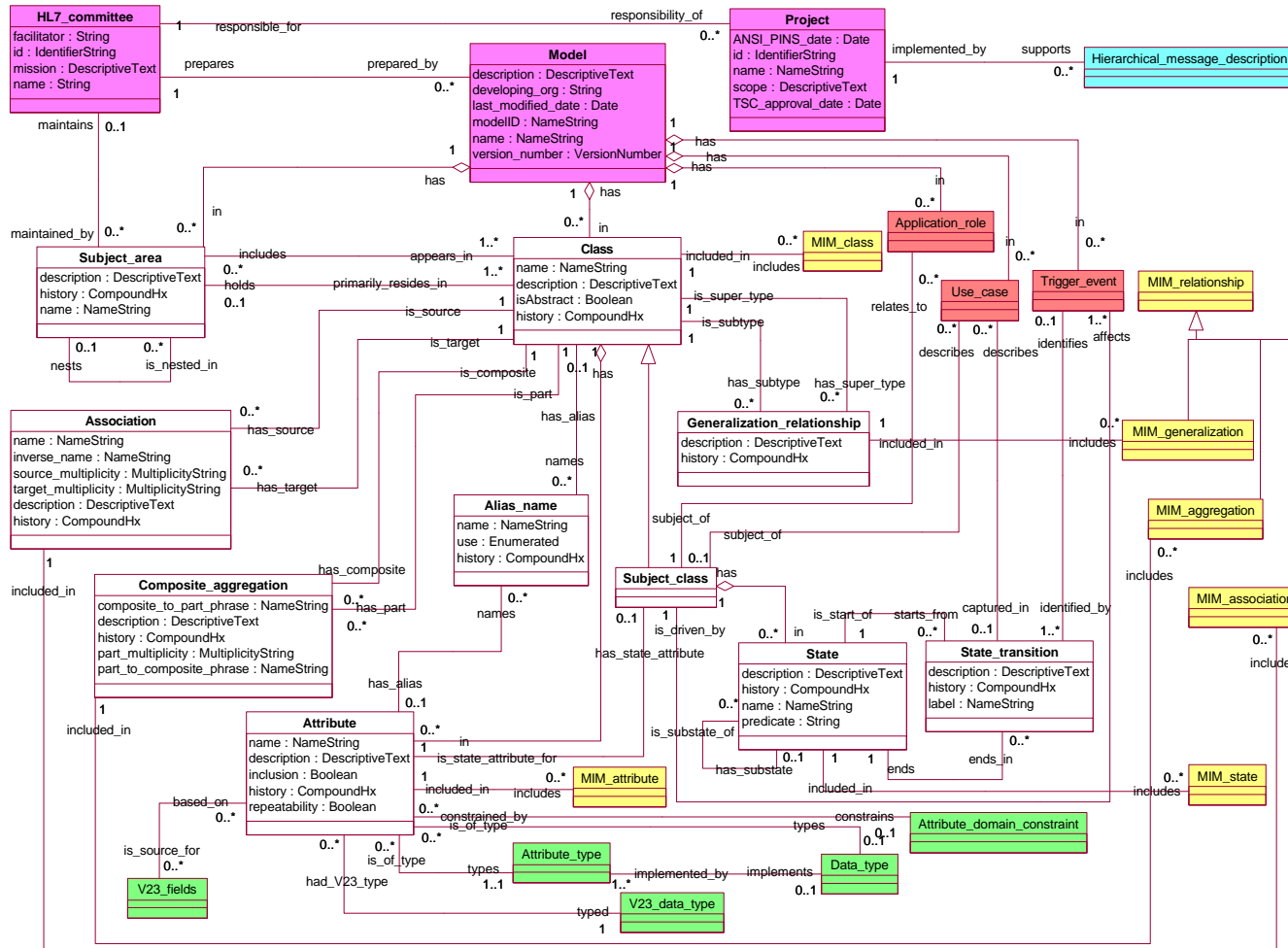


Figure 13-2. Information Model

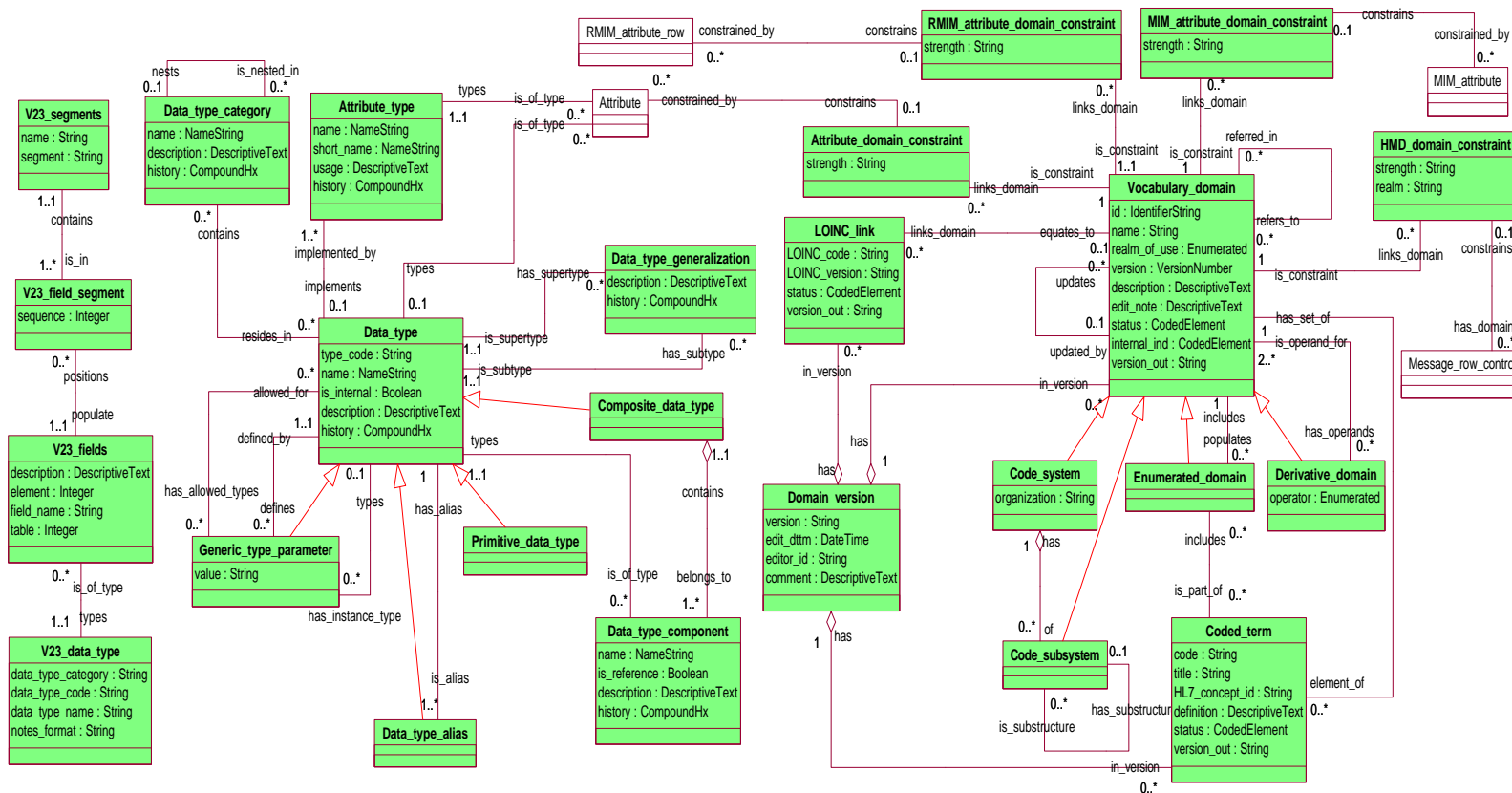


Figure 13-3. Data type and Vocabulary Domain Models

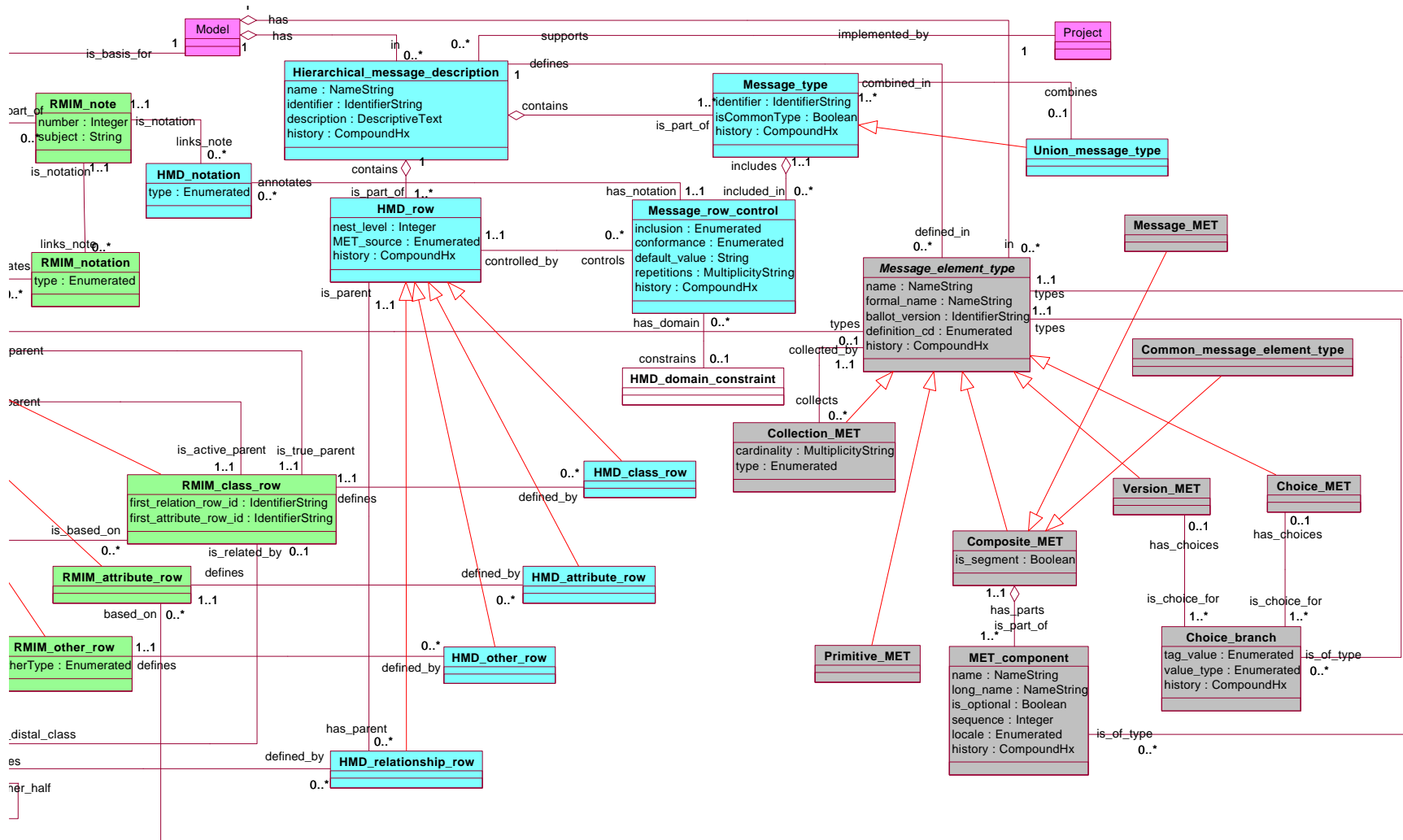


Figure 13-5. Message Design Model (right side)

Model: **HL7_V3_Meta-Model**

This model is Copyright by HL7

Identifications:

Organization: HL7

Version: V 1-13 19991020

ModelID: MET_0113

Developed by: Modeling and Methodology

Description of: **HL7_V3_Meta-Model**

This model represents the meta-model of the third release of the HL7 Version 3 MDF, as updated in October 1999.

It includes the datatype and domain model, extensions to the information, interaction and use case models, and the results of two complete revisions of the Message Design Model. The latter includes the message element type model, a model of the message information model, a model of the refined message information model (R-MIM) and the relationships between these artifacts and the elements of the HMD.

This version is felt to be complete, it matches the current HL7 tooling, and is mostly correct, but probably needs some revision after the message design concepts have been tested.

Subject Areas for: **HL7_V3_Meta-Model**

Subject Area: **MET Data type model**

The data type model defines the structure of the data types that may be assigned to information model attributes when these attributes are included in messages. It expresses the hierarchical relationship between data types and their components. It defines the role for attribute types in the information model. It also includes the structure of HL7 Version 2.x fields and data types.

Contains classes: **Attribute**
Attribute_type
Composite_data_type
Data_type
Data_type_alias
Data_type_category
Data_type_component
Data_type_generalization
Generic_type_parameter
HL7_committee
Model
Primitive_data_type
V23_data_type
V23_field_segment
V23_fields
V23_segments

Subject Area: MET Hierarchical message description

The Hierarchical message description model specifies the semantic links between elements of a MIM, the message object diagram (MOD), the abstract message definition for a set of message structures, and the message structures themselves.

Contains classes: **Hierarchical_message_description**
HMD_attribute_row
HMD_class_row
HMD_domain_constraint
HMD_notation
HMD_other_row
HMD_relationship_row
HMD_row
Message_element_type
Message_row_control
Message_type
Model
RMIM_attribute_row
RMIM_class_row
RMIM_note
RMIM_other_row
RMIM_relationship_row
Union_message_type

Subject Area: MET Information model

The information model defines the content of messages exchanged with HL7. Classes, connections, attributes, and states are the primary building blocks of the information model. Classes provide abstractions of the objects represented by the model. The semantic relationships between classes are expressed using connections. The three types of connections are Generalization-specialization, Whole-part, and Instance. Attributes are the facts applicable to the objects of the class, and states capture changes that trigger events have upon the subject classes of the information model.

Contains classes: **Alias_name**
Association
Attribute

Attribute_domain_constraint
Attribute_type
Class
Composite_aggregation
Data_type
Generalization_relationship
HL7_committee
State
State_transition
Subject_area
Subject_class
V23_data_type
V23_fields

Subject Area: MET Interaction model

The interaction model specifies the information flows that are needed to support the use cases defined in the use case model.

It includes the information flows or interactions, the trigger events, the application roles that send and receive the interactions and scenarios that provide an interaction trace for a series of events.

Contains classes: **Application_role**
HL7_committee
Interaction
Interaction_model_category
Interaction_sequence
Storyboard
Storyboard_example
Trigger_event
Use_case
Use_case_sequence

Subject Area: MET Message element types

The message element type model expresses the semantic relationship between the meta-model elements that define the type structure used to represent HL7 information structures for communications.

Contains classes: **Choice_branch**
Choice_MET
Collection_MET
Common_message_element_type
Composite_MET
Hierarchical_message_description
Message_element_type
Message_MET
MET_component
Model
Primitive_MET
Version_MET

Subject Area: MET Message information model

The message information model subject area is that sub-set of the message specification model than includes all of the elements that make up a MIM.

Contains classes: **Association**
Attribute
Class
Composite_aggregation
Data_type
Generalization_relationship
Hierarchical_message_description
Message_information_model
MIM_aggregation
MIM_association
MIM_attribute
MIM_attribute_domain_constraint
MIM_class
MIM_generalization
MIM_relationship
MIM_state
State

Subject Area: MET Message specification model

The message specification model maps the information content of the information model into the abstract and concrete message specifications needed to communicate between computer applications.

It includes: the message information model which is the sub-set of the information model needed to support a set of messages; the hierarchical message description that maps the information content of the MIM into a set of message formats; and the message element type model which describes the type structure used to convey messages.

Contains classes: **Choice_branch**
Choice_MET
Collection_MET
Common_message_element_type
Composite_MET
Data_type
Hierarchical_message_description
HL7_committee
HMD_attribute_row
HMD_class_row
HMD_domain_constraint
HMD_notation
HMD_other_row
HMD_relationship_row
HMD_row
Message_element_type
Message_information_model
Message_MET
Message_row_control
Message_type
MET_component
MIM_aggregation
MIM_association
MIM_attribute
MIM_attribute_domain_constraint
MIM_class
MIM_generalization

MIM_relationship
MIM_state
Model
Primitive_MET
Project
Refined_message_information_model
RMIM_attribute_row
RMIM_class_row
RMIM_notation
RMIM_note
RMIM_other_row
RMIM_relationship_row
RMIM_row
RMIM_state_row
Union_message_type
Version_MET

Subject Area: MET Model identification and scope

The components that define the overall model, the project and the domain information models that support the project.

Contains classes: **HL7_committee**
Model
Project

Subject Area: MET Refined message information model

Contains classes: **Message_element_type**
Message_information_model
MIM_attribute
MIM_class
MIM_relationship
MIM_state
Model
Refined_message_information_model
RMIM_attribute_row
RMIM_class_row
RMIM_notation
RMIM_note
RMIM_other_row
RMIM_relationship_row
RMIM_row
RMIM_state_row

Subject Area: MET Use case model

The use case model is a collection of actors, use cases and scenarios that comprise a high level functional analysis of healthcare. For HL7, the purpose of this analysis is to the requirements for messaging between computer applications . The Use Case Model documents the institutional, medical, and business practices that the message(s) being created will support.

Contains classes: **Actor**
HL7_committee
Use_case

Use_case_category
Use_case_relationship

Subject Area: MET Vocabulary domain model

Defines the structure and relationships for vocabulary domains that are used to constrain coded information model attributes in the information and message design models.

Contains classes: **Attribute**
Attribute_domain_constraint
Code_subsystem
Code_system
Coded_term
Derivative_domain
Domain_version
Enumerated_domain
HMD_domain_constraint
LOINC_link
MIM_attribute
MIM_attribute_domain_constraint
RMIM_attribute_domain_constraint
RMIM_attribute_row
Vocabulary_domain

Subject Area: **Meta 1 Use case and Interaction models**

Defined for graphing purposes only.

Contains classes: **Actor**
Application_role
Class
Interaction
Interaction_model_category
Interaction_sequence
Message_type
Model
State_transition
Storyboard
Storyboard_example
Subject_class
Trigger_event
Use_case
Use_case_category
Use_case_relationship
Use_case_sequence

Subject Area: **Meta 2 Information model**

Defined for graphing purposes only.

Contains classes: **Alias_name**
Application_role
Association
Attribute
Attribute_domain_constraint
Attribute_type
Class
Composite_aggregation
Data_type
Generalization_relationship
Hierarchical_message_description
HL7_committee
MIM_aggregation

MIM_association
MIM_attribute
MIM_class
MIM_generalization
MIM_relationship
MIM_state
Model
Project
State
State_transition
Subject_area
Subject_class
Trigger_event
Use_case
V23_data_type
V23_fields

Subject Area: Meta 3 Data type and Domain models

Contains classes: Attribute
Attribute_domain_constraint
Attribute_type
Code_subsystem
Code_system
Coded_term
Composite_data_type
Data_type
Data_type_alias
Data_type_category
Data_type_component
Data_type_generalization
Derivative_domain
Domain_version
Enumerated_domain
Generic_type_parameter
HMD_domain_constraint
LOINC_link
MIM_attribute
MIM_attribute_domain_constraint
Primitive_data_type
V23_data_type
V23_field_segment
V23_fields
V23_segments
Vocabulary_domain

Information model in: **HL7_V3_Meta-Model**

Classes in: **HL7_V3_Meta-Model**

Class: **Actor**

Is part of: **Model**

Associated with: **Use_case**
 Use_case_category

Description of: **Actor**

An actor is a role played by someone or something that interacts directly with the elements represented by the classes in the information model. With one exception, actors cannot represent information systems. The exception is a special actor with the literal name "some information system". The name is chosen to reinforce the notion that use cases are not built on a priori knowledge of the functionality of specific healthcare information systems.

Composition for: **Actor**

in (1,1) :: Model :: has (0,n)

The relationship between actors and the models of which they are a part.

Associations for: **Actor**

participates_in (0,n) :: Use_case :: involves (1,n)

included_in (0,n) :: Use_case_category :: includes (0,n)

Attributes of: **Actor**

description : DescriptiveText

A short informative statement that allows people to determine, with certainty, whether a particular real world role is an instance of the actor.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

name : NameString

The actors in the model are each given a unique name. The actor name is a singular noun or noun phrase.

Class: **Alias_name**

Associated with: **Attribute**
 Class

Description of: **Alias_name**

Classes and attributes in the information model may have one or more alias names. These allow for special uses in HL7, such as short names, and for translations to other languages.

Associations for: **Alias_name**

names (0,1) :: Attribute :: has_alias (0,n)

names (0,1) :: Class :: has_alias (0,n)

Attributes of: **Alias_name**

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

name : NameString

The alias name for the linked element.

use : Enumerated

A code indicating the use of this alias.

Class: **Application_role**

Is part of: **Model**

Associated with: **Interaction**
Interaction
Interaction_model_category
Subject_class

Description of: **Application_role**

Application roles are abstractions that name roles that may be played by health-care information system components when sending or receiving HL7 messages. Thus they are a defined set of responsibilities with respect to interactions. The role may have responsibility to send or receive one or more interactions. The application role and its responsibilities may form the basis for establishing conformance specifications for a standard.

Application roles should be stereotyped with respect to their responsibilities for information about a subject class. The technical committee should start its thinking about application roles from the perspective that there are three fundamental purposes for message exchange, three basic "messaging modes". These are:

Declarative - This includes messages that are sent with the intent of conveying information from one party to another. For example, a healthcare provider might send a message every time a person is registered as a patient with that provider.

Imperative - This includes messages that direct or request a party to do something. For example, a healthcare provider might send a message to a laboratory every time the provider needs the laboratory to perform a test. Note, that even though the message must include information about the test and the patient the test is for, the primary purpose of the message is to request that the test be done.

Interrogative - This includes messages that ask for information, that ask a question. For example, a healthcare provider might send a message to an MPI mediator asking whether the MPI mediator has information about a specific person.

Application roles will have stereotyped names constructed as <subject class> <relationship>. These stereotypes are specific to the messaging modes.

- For the declarative mode, the typical roles are "declarer" and "recipient"
- For the imperative model, the typical roles are "placer" and "filler"
- For the interrogative mode, the typical roles are "questioner" and "answerer"

There is no requirement that a Technical Committee create all of these "<subject class><relationship>" roles. Nor is it limited to these possibilities. But it is expected that the committee will consider these stereotypes.

Composition for: **Application_role**

in (1,1) :: Model :: has (0,n)

The relationship between application roles and the models of which they are a part.

Associations for: **Application_role**

receives (0,n) :: Interaction :: received_by (1,1)

A reference to the application role that is responsible for receiving the message involved in this interaction. The receiving role must be prepared to accept the message and to fulfill the receiver responsibility.

sends (0,n) :: Interaction :: sent_by (1,1)

The sending role has responsibilities to recognize the trigger event for the interaction and to cause the appropriate message to be sent.

included_in (0,n) :: Interaction_model_category :: includes (0,n)

relates_to (1,1) :: Subject_class :: subject_of (0,n)

Links each Application role to the Subject class for which it plays a role. The nature of this relationship is stereotyped as discussed in the description for the Application_role.

Attributes of: **Application_role**

description : DescriptiveText

The text that describes the application role and summarizes the interaction responsibilities that are part of that role.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

identifier : IdentifierString

An identifier assigned to the application role. The identifier is unique within the scope of the model in which the application role is defined. In HL7, committees manage the unique identifiers for their application roles, and concatenate the committee identifier as "Cnn_<identifier>."

name : NameString

A name assigned to the application role. The name is unique within the scope of the model in which the application role is defined.

Class: Association

Associated with: **Class**
 Class
 MIM_association

Description of: **Association**

An association between classes depicts the occurrence of a reference attribute used to connect class instances (objects). The associated objects can be of the same or different classes. When the association is defined one of the two classes is designated the "source class" and the other the "target" class. These designations are necessary to unambiguously define an association, but the designations have no semantic implications about the roles of the associated classes within the information model being defined. The selection of which class to label as "source" is arbitrary.

Associations for: **Association**

has_source (1,1) :: Class :: is_source (0,n)

A reference to the class from which the association perspective is captured.

has_target (1,1) :: Class :: is_target (0,n)

A reference to the class that is the target of the association.

included_in (0,n) :: MIM_association :: includes (1,1)

Attributes of: **Association**

description : DescriptiveText

A short informative statement that describes the relationship between the classes connected by the association.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

Note, the version data must fit within the range of the applicable versions for both classes to which this element is attached.

inverse_name : NameString

A short action phrase that specifies the role of the destination class in the association. Each association between the same pair of classes must have a unique inverse name.

name : NameString

A short action phrase that specifies the role of the source class in the association. Each association between the same pair of classes must have a unique name.

source_multiplicity : MultiplicityString

A set of values and value ranges indicating the number of source class instances involved in the connection. In value ranges the minimum shall be zero or more and the maximum shall be equal to or greater than the minimum. The maximum number may be expressed as unlimited.

target_multiplicity : MultiplicityString

A set of values and value ranges indicating the number of destination class instances involved in the connection. In value ranges the minimum shall be zero or more and the maximum shall be equal to or greater than the minimum. The maximum number may be expressed as unlimited.

Class: **Attribute**

Is part of:	Class
Associated with:	Alias_name Attribute_domain_constraint Attribute_type Data_type MIM_attribute Subject_class V23_data_type V23_fields V23_segments

Description of: **Attribute**

Attributes in the information model are the major source of the data content used in HL7 communications. Attributes are abstractions of the data captured about classes. Attributes capture separate aspects of the class and take their values independent of one another. Attribute domain specifications are captured in datatypes.

Composition for: **Attribute**

in (1,1) :: Class :: has (0,n)

The relationship between attributes and the classes of which they are a part.

Associations for: **Attribute**

has_alias (0,n) :: Alias_name :: names (0,1)

constrained_by (0,1) :: Attribute_domain_constraint :: constrains (0,n)

is_of_type (1,1) :: Attribute_type :: types (0,n)

A reference between an attribute and its attribute type. The attribute type code must also be the terminal component of the attribute name.

is_of_type (0,1) :: Data_type :: types (0,n)

A link between an attribute and the datatype that has been assigned to it. An attribute is assigned a datatype the first time that it is used in an HMD, and retains that type thereafter.

included_in (0,n) :: MIM_attribute :: includes (1,1)

is_state_attribute_for (0,1) :: Subject_class :: has_state_attribute (1,1)

The state attribute of a class contains a value indicating the current state of the class. In the event that the class has concurrent states, the attribute must be a set of state values.

had_V23_type (1,1) :: V23_data_type :: typed (0,n)

Provides an indication of the data type used in Version 2.x for a particular attribute, if such prior usage has been identified.

based_on (0,n) :: V23_fields :: is_source_for (0,n)

Provides a linkage for an information model attribute to its equivalent version 2.x field, if such linkage exists and has been identified.

stems_from (0,n) :: V23_segments :: source_of (0,n)

Many attributes are traced to equivalent content in HL7 Version 2.x. This connection is secondary to the path that traces an attribute to an HL7 field to a segment. It is provided for modelers who wish to specify particular segments for information model attributes.

Attributes of: **Attribute**

description : DescriptiveText

A short informative description of the Class characteristic captured by the Attribute.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

Note, the version data must fit within the range of the applicable versions for the class of which this element is a part.

inclusion : Boolean

An indication of whether the inclusion of the attribute is mandatory in all HL7 HMDs and messages. Setting the inclusion to mandatory in the information model is deprecated.

name : NameString

Singular nouns are used for attribute names. Attribute names are unique within the class they describe and within the set of attributes inherited by the class they describe. The terminal element of the name shall indicate the attribute type for the attribute.

repeatability : Boolean

Indicates whether this attribute may repeat when it is included in the message structure of a hierarchical message description. The default is false, non-repeating.

Class: Attribute domain constraint

Associated with: **Attribute**
 Vocabulary_domain

Description of: Attribute domain constraint

Constrains a coded attribute to a particular vocabulary domain.

For any class, the special attribute status_cd has as its domain all of the states of the class.

Associations for: Attribute domain constraint

constrains (0,n) :: Attribute :: constrained_by (0,1)

links_domain (1,1) :: Vocabulary_domain :: is_constraint (0,n)

Attributes of: Attribute domain constraint**strength : String**

The strength of the constraint is either CWE (coded with exceptions) or CNE (coded, no exceptions). If no value is given, CWE is the default.

Class: Attribute type

Associated with: **Attribute**
 Data_type

Description of: Attribute type

An indication of the form of the attribute, and of its usage. The use of attribute type words in attribute names aids in creating uniformity in the names, helps to avoid unintentional redundancy, and adds clarity to the model.

Associations for: Attribute type

types (0,n) :: Attribute :: is_of_type (1,1)

A reference between an attribute and its attribute type. The attribute type code must also be the terminal component of the attribute name.

implemented_by (0,1) :: Data_type :: implements (1,n)

Each Attribute type may be implemented by one or more data types.

Attributes of: **Attribute_type**

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

name : NameString

The full name of the attribute type.

short_name : NameString

The representation of the attribute type that is appended to the name of the attribute to indicate the attribute type.

usage : DescriptiveText

A brief description of the intended usage of this attribute type.

Class: **Choice_branch**

Associated with: **Choice_MET**
 Message_element_type
 Version_MET

Description of: **Choice_branch**

A choice branch is provides one alternative for a choice MET or a version MET. The branch includes a tag used to identify the branch when it is instantiated and a type that represents the branch.

Associations for: **Choice_branch**

is_choice_for (0,1) :: Choice_MET :: has_choices (1,n)

Each branch of a Choice_MET includes a type. One of these choices is used in a Choice MEI. A choice branch must be part of a version MET or a choice MET, but not a part of both.

is_of_type (1,1) :: Message_element_type :: types (0,n)

Each choice branch must be typed by an MET.

is_choice_for (0,1) :: Version_MET :: has_choices (1,n)

Each branch of a Version_MET includes a type. All of these choices are used in a Choice MEI. A choice branch must be part of a version MET or a choice MET, but not a part of both.

Attributes of: **Choice_branch**

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

tag_value : Enumerated

The value that identifies a particular choice. The tag values must be unique within a single choice MET or version MET.

The branch must have a unique tag value whether or not that value governs the selection of the choice in an MEI. Whether this is the governing value for a particular HMD is determined by the value_type attribute. If that attribute has a value of "Other" then this value rules. If the value_type is "State" then a set of "states" will govern the choice. The state values are linked to the choice branch by the mandatory association to a choice node which contains the states as choice values.

value_type : Enumerated

The value type determines whether this choice branch is governed by the tag value, or whether it is governed by a set of choice values linked to the (mandatory) associated choice node, which values are determined by associations to states in the MIM.

The presently allowed values for this element are "State" or "Other" where "Other" draws the value from tag value.

Class: **Choice_MET**

Subtype of: **Message_element_type**

Associated with: **Choice_branch**

Description of: **Choice_MET**

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

Associations for: **Choice_MET**

has_choices (1,n) :: Choice_branch :: is_choice_for (0,1)

Each branch of a Choice_MET includes a type. One of these choices is used in a Choice MEI. A choice branch must be part of a version MET or a choice MET, but not a part of both.

Class: **Class**

Supertype of: **Subject_class**

Is part of: **Model**

Composite of: **Attribute**

Associated with: **Alias_name**
 Association
 Association
 Composite_aggregation
 Composite_aggregation
 Generalization_relationship
 Generalization_relationship
 MIM_class
 Subject_area
 Subject_area

Description of: **Class**

An abstraction of a set of real-world things (objects) such that all of the objects have the same characteristics and all instances are subject to and conform to the same rules. Classes are the people, places, roles, things, and events about which information is kept.

Composition for: **Class**

has (0,n) :: Attribute :: in (1,1)

The relationship between attributes and the classes of which they are a part.

in (1,1) :: Model :: has (0,n)

The relationship between classes and the models of which they are a part.

Associations for: **Class**

has_alias (0,n) :: Alias_name :: names (0,1)

is_source (0,n) :: Association :: has_source (1,1)

A reference to the class from which the association perspective is captured.

is_target (0,n) :: Association :: has_target (1,1)

A reference to the class that is the target of the association.

is_composite (0,n) :: Composite_aggregation :: has_composite (1,1)

A reference to the class that is the composition the relationship.

is_part (0,n) :: Composite_aggregation :: has_part (1,1)

A reference to the class that is the part class in the relationship.

is_subtype (0,n) :: Generalization_relationship :: has_subtype (1,1)
The linkage between a generalization relationship and the subtype that participates in that connection.

is_super_type (0,n) :: Generalization_relationship :: has_super_type (1,1)
The linkage between a generalization relationship and the supertype that participates in that connection.

included_in (0,n) :: MIM_class :: includes (1,1)

appears_in (0,n) :: Subject_area :: includes (1,n)
The linkage between a Subject area and each of the Classes that are in that Subject area.

primarily_resides_in (0,1) :: Subject_area :: holds (1,n)
The linkage between a Class and the Subject area that is its primary residence. This must be established if a Class resides in more than one Subject area.

Attributes of: **Class**

description : DescriptiveText

A short informative statement that allows one to tell, with certainty, whether a particular real world thing is an instance of the Class as conceptualized in the information model.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

isAbstract : Boolean

This variable indicates whether or not this Class is an abstract class. An abstract class is a class that can not be instantiated and is customarily the generalization class in a generalization/specialization structure.

name : NameString

The Classes in the information model are given a unique name. The Class name is a singular noun or noun phrase.

Class: Code_subsystem

Subtype of: **Vocabulary_domain**

Is part of: **Code_system**

Associated with: **Code_subsystem**
 Code_subsystem

Description of: Code subsystem

Many code systems have subsystems. These are, e.g., axes in SNOMED, classes, and subclasses in hierarchical codes such as ICD. The structure of coding systems is idiosyncratic.

Composition for: Code subsystem

of (1,1) :: Code_system :: has (0,n)

Associations for: Code subsystem

has_substructure (0,n) :: Code_subsystem :: is_substructure (0,1)

A code sub-system may be further decomposed in a hierarchical fashion by implementing this association.

is_substructure (0,1) :: Code_subsystem :: has_substructure (0,n)

A code sub-system may be further decomposed in a hierarchical fashion by implementing this association.

Class: Code system

Subtype of: **Vocabulary_domain**

Composite of: **Code_subsystem**

Description of: Code system

A system is a published code system by an organization, that defines it, publishes it, maintains it, and thus guarantees for its usefulness and continuity.

ORGANIZATION: e.g. WHO, College of American Pathologists, ISO, IANA, and HL7 of course.

NAME: e.g. ICD, ICPM, ICPC, SNOMED, 639-1, MIME types, ...

Composition for: Code system

has (0,n) :: Code_subsystem :: of (1,1)

Attributes of: Code system

organization : String

The name of the organization that maintains the code system. Examples are WHO, College of American Pathologists, ISO, IANA, and HL7.

Class: Coded term

Is part of: **Domain_version**

Associated with: **Enumerated_domain**
Vocabulary_domain

Description of: **Coded_term**

This class is not expected to be exhaustively instantiated. Its purpose is not to copy the lists of terms of all external code sets. Rather it will be used for those codes that must be enumerated to define a particular HL7 domain.

It can also be used to capture and manage HL7-defined code sets.

Composition for: **Coded_term**

in_version (1,1) :: Domain_version :: has (0,n)

Associations for: **Coded_term**

is_part_of (0,n) :: Enumerated_domain :: includes (0,n)

An enumeration is a set of terms.

element_of (1,1) :: Vocabulary_domain :: has_set_of (0,n)

This relationship indicates that a vocabulary domain is a set of terms and every term belongs to one vocabulary domain.

Through subsystems and derived vocabulary domains, any term can be member of more than one vocabulary domain.

Attributes of: **Coded_term**

code : String

The text string used within the coding system to identify this concept.

definition : DescriptiveText

A textual representation of the meaning of this entry as it is represented in the coding system from which it came.

Any term may have a definition stored. For the terms that are referenced from external coding systems, the definition will not be included..

These terms may be used to maintain HL7 code tables, in which case, the definition is mandatory.:

A freshly defined HL7 code table would be an Enumeration domain (refers to itself as a "base"). All items in the enumeration would be terms with definition attached to them.

HL7_concept_id : String

the unique item identifier assigned by HL7 to this concept. This concept identifier is globally unique to a concept throughout all HL7 tables. That is, if the concept male occurred in another vocabulary domain in addition to the Gender domain, it would again have an item identifier of "1". If a universal terminology of medicine becomes available, the universal concept identifier from that terminology could be used in place of this HL7 assigned identifier.

status : CodedElement

The status of this entry within this domain table. The values for Status come from the vocabulary domain EditStatus. Some values for status are Proposed, Rejected, Active, Obsolete, and Inactive.

title : String

A textual name for the term.

version_out : String

The version number of the table at which this entry was deleted. A blank Vout value means that the row continues to exist in the current version of the table.

Class: Collection MET

Subtype of: **Message_element_type**

Associated with: **Message_element_type**

Description of: **Collection MET**

Provides for the inclusion of a collection of other METs as a component of an MET.

The collection may be a List (ordered collection of instances), a Set (unordered collection of unique instances) or a Bag (unordered collection of instances which might not be unique).

Associations for: **Collection MET**

collects (1,1) :: Message_element_type :: collected_by (0,n)

Each collection MET collects elements of a single type.

Attributes of: **Collection MET**

cardinality : MultiplicityString

Provides the minimum and maximum number of occurrences in the collection.

type : Enumerated

Determines whether the collection is a List, a Set, or a Bag.

Class: Common message element type

Subtype of: **Composite_MET**

Description of: **Common message element type**

The common message element type is composite MET that has been declared as a public type available for assignment or substitution, as appropriate in an HMD.

OpenIssue: This class may require additional attributes.

Class: Composite aggregation

Associated with: **Class**
 Class
 MIM_aggregation

Description of: Composite aggregation

An association between classes that depicts the relationship between a composite aggregate class and its component parts.

Associations for: Composite aggregation

has_composite (1,1) :: Class :: is_composite (0,n)

A reference to the class that is the composition the relationship.

has_part (1,1) :: Class :: is_part (0,n)

A reference to the class that is the part class in the relationship.

included_in (0,n) :: MIM_aggregation :: includes (1,1)

Attributes of: Composite aggregation

composite_to_part_phrase : NameString

A short phrase representing the nature of the relationship from the perspective of the composite class. Example phrases are: "includes", "contains", "consists of", "has as parts". Other phrases may also be used. If no phrase is specified, the phrase "contains" will be assumed.

description : DescriptiveText

A short informative description of the composite aggregation connection.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

Note, the version data must fit within the range of the applicable versions for both classes to which this element is attached.

part_multiplicity : MultiplicityString

A set of values and value ranges indicating the number of part class instances involved in the aggregation. In value ranges the minimum shall be zero or more and the maximum shall be equal to or greater than the minimum. The maximum number may be expressed as unlimited.

part_to_composite_phrase : NameString

A short phrase representing the nature of the relationship from the perspective of the part class . Example phrases are: "is included in", "is contained in", "is part of", and "is a

component of". Other phrases may also be used. If no phrase is specified, "is part of" will be assumed.

Class: **Composite_data_type**

Subtype of: **Data_type**
Composite of: **Data_type_component**
Associated with: **Composite_MET**

Description of: **Composite_data_type**

A composite data type consists of one or more named and typed components.

Composition for: **Composite_data_type**

contains (1,n) :: Data_type_component :: belongs_to (1,1)

Associations for: **Composite_data_type**

specifies (1,1) :: Composite_MET :: specified_by (0,1)

Each composite data type has an associated composite MET that is used to represent it in messages.

Class: **Composite_MET**

Subtype of: **Message_element_type**
Supertype of: **Common_message_element_type**
 Message_MET
Composite of: **MET_component**
Associated with: **Composite_data_type**

Description of: **Composite_MET**

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, although many may be of the same type.

Composition for: **Composite_MET**

has_parts (1,n) :: MET_component :: is_part_of (1,1)

Associations for: **Composite_MET**

specified_by (0,1) :: Composite_data_type :: specifies (1,1)

Each composite data type has an associated composite MET that is used to represent it in messages.

Attributes of: **Composite MET**

is_segment : Boolean

This attribute is intended to capture the definition of a "segment" MET for use in a segment-positional syntax (ER7).

OpenIssue: This attribute is redundant to the association to a MIM_class. If the association is present, then it is a segment. Otherwise, not.

Class: **Data_type**

Supertype of: **Composite_data_type**
 Data_type_alias
 Generic_type_parameter
 Primitive_data_type

Is part of: **Model**

Associated with: **Attribute**
 Attribute_type
 Data_type_alias
 Data_type_category
 Data_type_component
 Data_type_generalization
 Data_type_generalization
 Generic_type_parameter
 Generic_type_parameter
 Generic_type_parameter
 MIM_attribute

Description of: **Data_type**

Datatypes are used to express the type of an attribute or of a data type component. A data type may be composite, primitive, an alias or a generic type parameter.

A generic type parameter contains a parameter that is part of the definition of a generic data type. Each generic type parameter is part of the definition for a single generic type.

A composite data type contains one or more components.

An alias provides an alternative name, alternate type code and additional description for another data type.

A primitive data type is a data type that is defined entirely by its specification. A primitive data type may have generic type parameters.

A generic data type is a type that has one or more generic type parameters. It provides a pattern for instantiating a specific, usually composite, data type.

Composition for: **Data_type**

in (1,1) :: Model :: has (0,n)

The relationship between data types and the models in which they are first defined.

Associations for: **Data_type**

types (0,n) :: Attribute :: is_of_type (0,1)

A link between an attribute and the datatype that has been assigned to it. An attribute is assigned a datatype the first time that it is used in an HMD, and retains that type thereafter.

implements (1,n) :: Attribute_type :: implemented_by (0,1)

Each Attribute type may be implemented by one or more data types.

has_alias (1,n) :: Data_type_alias :: is_alias (1,1)

Provides for one type to represent a simple alias for another.

resides_in (0,n) :: Data_type_category :: contains (0,n)

types (0,n) :: Data_type_component :: is_of_type (1,1)

Each component is linked to a single type either directly or through a generic type parameter.

is_subtype (0,n) :: Data_type_generalization :: has_subtype (1,1)

Each data type generalization includes a single sub-type.

is_supertype (0,n) :: Data_type_generalization :: has_supertype (1,1)

Each data type generalization provides sub-types for a single super-type..

allowed_for (0,n) :: Generic_type_parameter :: has_allowed_types (0,n)

Determines the set of types that a generic type parameter may implement.

defined_by (0,n) :: Generic_type_parameter :: defines (1,1)

Relationship between a Generic Type Parameter and the Generic type for which it is a parameter.

types (0,n) :: Generic_type_parameter :: has_instance_type (0,1)

This relationship defines the particular instantiation type for a generic instance.

is_working (0,n) :: MIM_attribute :: has_working (0,1)

If a MIM attribute is associated with a RIM attribute that does not have an assigned data type, the MIM attribute may be assigned an alternative, working data type. This association must not be instantiated otherwise.

Attributes of: **Data_type**

description : DescriptiveText

A detailed description or specification for the data type. All such descriptions are assumed to also reference a broader specification of data types.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

is_internal : Boolean

A data type may be defined as being "internal". An internal type is used only to define other composite data types. Internal types shall not be used in messages. For example, we define a type Binary that contains pure raw data bits, and that is used only by Multimedia Enabled Free Text.

name : NameString

The formal name for the data type.

type_code : String

The formal code assigned by the Control/Query Committee for this datatype. This is the representation of the data type that appears as the data type for attributes of the information model and data type components in the data type model.

Class: Data_type alias

Subtype of: **Data_type**

Associated with: **Data_type**

Description of: **Data_type alias**

An alias provides an alternative name, alternate type code and additional description for another data type. Its most common use is to provide a specific "user-friendly" name for a collection of other data types.

Associations for: **Data_type alias**

is_alias (1,1) :: Data_type :: has_alias (1,n)

Provides for one type to represent a simple alias for another.

Class: Data_type category

Is part of: **Model**

Associated with: **Data_type**
Data_type_category
Data_type_category
HL7_committee

Description of: **Data_type category**

A data type category collects data types that represent similar real world concepts, or are represented in a similar fashion.

Composition for: **Data_type_category**

in (1,1) :: Model :: has (0,n)

The relationship between data type categories and the models of which they are a part.

Associations for: **Data_type_category**

contains (0,n) :: Data_type :: resides_in (0,n)

is_nested_in (0,1) :: Data_type_category :: nests (0,n)

nests (0,n) :: Data_type_category :: is_nested_in (0,1)

maintained_by (1,1) :: HL7_committee :: maintains (0,n)

Attributes of: **Data_type_category**

description : DescriptiveText

The description of the data type category expresses the unifying concept that causes a set of data types to be included in this category.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

name : NameString

The name given to the data type category.

Class: **Data_type_component**

Is part of: **Composite_data_type**

Associated with: **Data_type**

Description of: **Data_type_component**

A component is an element of a data type that may be valued when the data type is used in HL7 communications. The component takes its type from a data type that is any of - primitive, composite, or generic type parameter.

A component of a composite data type is like a variable, i.e. it has a name and a type. The type can be declared to be included by reference instead of by value. This is useful if you know such a component mentions an instance that is already mentioned elsewhere in the communication. In languages such as Java, where objects are always handled through references this does not make any difference.

Composition for: **Data_type_component**

belongs_to (1,1) :: Composite_data_type :: contains (1,n)

Associations for: **Data_type_component**

is_of_type (1,1) :: Data_type :: types (0,n)

Each component is linked to a single type either directly or through a generic type parameter.

Attributes of: **Data_type_component**

description : DescriptiveText

The description of a component should include its role within the composite of which it is a part and its relationships, if any, to other components of the same composite.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

is_reference : Boolean

The component type can be declared to be included by reference instead of by value. This is useful if you know such a component mentions an instance that is already mentioned elsewhere in the communication. In languages such as Java, where objects are always handled through references this does not make any difference.

name : NameString

The formal name of the data type component.

Class: **Data_type_generalization**

Associated with: **Data_type**
 Data_type

Description of: **Data_type_generalization**

Types can maintain an inheritance relationship with each other. We explicitly allow (and use) "multiple inheritance". However, we do use inheritance as a way to specialize subtypes from general super-types. Rather we go the other way. Abstract generalized types are used to categorize the concrete types in different ways. Thus one can get hold of all types that have a certain property of interest.

For instance, we define the generalized type Quantity to subsume all quantitative types. This is used to define one type Ratio as a ratio of any two quantities.

Similarly, we define a data type Interval that is a continuous subset of any type with an order relation. All types with an order relation are subsumed under OrderedType. Note that not all quantities are ordered (e.g. vectors are not) and there may be non-quantities that have an order relationship (ordinals, e.g. military ranks).

Associations for: **Data type generalization**

has_subtype (1,1) :: Data_type :: is_subtype (0,n)

Each data type generalization includes a single sub-type.

has_supertype (1,1) :: Data_type :: is_supertype (0,n)

Each data type generalization provides sub-types for a single super-type..

Attributes of: **Data type generalization**

description : DescriptiveText

A statement of the nature of the generalization relationship.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

Class: **Derivative domain**

Subtype of: **Vocabulary_domain**

Associated with: **Vocabulary_domain**

Description of: **Derivative domain**

A derivative is the vocabulary domain that results from applying a particular operator to two or more other vocabulary domains.

A derivative vocabulary domain is usually a subset of another set. Defining derivatives inclusion or exclusion of one domain with another. Derivatives can exist on different levels: e.g., in the realm of an institution (hospital department), a country (USA), a treaty (EU), and so on. Thus, each derivative can be defined on another derivative in a larger realm (e.g., a German hospital narrows the domain defined by the EU that in turn may be a subset of a WHO code).

While system and subsystem are defined externally, derivatives must be explicitly defined.

Associations for: **Derivative domain**

has_operands (2,n) :: Vocabulary_domain :: is_operand_for (0,n)

Relationship between a derivative, its operator, and the operands (other domains) to which the operator is applied.

Attributes of: **Derivative domain**

operator : Enumerated

The operator that defines the combinatorial rule applied to two or more operand domains to arrive at the derivative domain. Operators include union (+) and difference (-). Additional constructs may be provided.

Class: Domain version

Composite of: **Coded_term**
 LOINC_link
 Vocabulary_domain

Description of: Domain version

Captures each update of the vocabulary domain tables in a version, including the when the editing took place, who performed it, and comments as to what was done.

Composition for: Domain version

has (0,n) :: Coded_term :: in_version (1,1)

has (0,n) :: LOINC_link :: in_version (1,1)

has (0,n) :: Vocabulary_domain :: in_version (1,1)

Attributes of: Domain version

comment : DescriptiveText

A summary of why the edits were made, and what was done.

edit_dttm : DateTime

The date and time that the edit session began

editor_id : String

An identifier of the person or group responsible for the edits.

version : String

The version number of the edit session. This number is incremented by 1 each time a new edit session takes place. The version number is used as the value of Vin and Vout as appropriate to track which table entries in the vocabulary definition tables were added, modified, or deleted during the session.

Class: Enumerated domain

Subtype of: **Vocabulary_domain**

Associated with: **Coded_term**
 Vocabulary_domain

Description of: Enumerated domain

An enumerated domain is defined by an enumerated set of code terms. While large code systems are impractical to enumerate, and while some are not enumerable at all, enumeration is useful for small domains.

Associations for: **Enumerated domain**

includes (0,n) :: Coded_term :: is_part_of (0,n)

An enumeration is a set of terms.

populates (1,1) :: Vocabulary_domain :: includes (0,n)

An enumeration includes terms from a particular domain. In the case where the enumeration is the entire domain, this relationship is self-referential.

Class: **Generalization relationship**

Associated with: **Class**
 Class
 MIM_generalization

Description of: **Generalization relationship**

Generalization is a relationship between a class and subtypes of the class. A supertype can be associated with more than one subtype. Each of the subtypes associated with a single supertype is mutually exclusive. A subtype may be associated with more than one supertype. The hierarchy or lattice of generalizations is called a generalization relationship. The subtype inherits the attributes, and associations, and of all of its supertypes.

Associations for: **Generalization relationship**

has_subtype (1,1) :: Class :: is_subtype (0,n)

The linkage between a generalization relationship and the subtype that participates in that connection.

has_super_type (1,1) :: Class :: is_super_type (0,n)

The linkage between a generalization relationship and the supertype that participates in that connection.

included_in (0,n) :: MIM_generalization :: includes (1,1)

Attributes of: **Generalization relationship**

description : DescriptiveText

A short informative description of the Generalization relationship.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

Note, the version data must fit within the range of the applicable versions for both classes to which this element is attached.

Class: **Generic type parameter**

Subtype of: **Data_type**

Associated with: **Data_type**
Data_type
Data_type

Description of: **Generic type parameter**

A generic type parameter contains a parameter that is part of the definition of a generic data type. Each generic type parameter is part of the definition for a single generic type.

The most common form of generic type parameter provides an instantiation type, drawn from two or more types.

Other generic type parameters constrain the generic type, such as the collection type and multiplicity for a Collection.

Associations for: **Generic type parameter**

defines (1,1) :: Data_type :: defined_by (0,n)

Relationship between a Generic Type Parameter and the Generic type for which it is a parameter.

has_allowed_types (0,n) :: Data_type :: allowed_for (0,n)

Determines the set of types that a generic type parameter may implement.

has_instance_type (0,1) :: Data_type :: types (0,n)

This relationship defines the particular instantiation type for a generic instance.

Attributes of: **Generic type parameter**

value : String

Establishes the content for a generic type parameter that defines a property of a generic type other than an instantiation type.

Class: Hierarchical message description

Is part of: **Model**

Composite of: **HMD_row**
Message_type

Associated with: **Message_element_type**
Project

Description of: **Hierarchical message description**

A structure that completely defines the structure of a set of messages, and reflects the relationship of the elements of these messages to components of the Refined Message Information Model from which it derives and the Message Element Types that it defines or uses.

Composition for: **Hierarchical message description**

contains (1,n) :: HMD_row :: is_part_of (1,1)

A reference to the HMD that contains each HMD row.

contains (1,n) :: Message_type :: is_part_of (1,1)

Each message structure is contained in a single HMD.

in (1,1) :: Model :: has (0,n)

The relationship between hierarchical message descriptions and the models in which they are first defined.

Associations for: **Hierarchical message description**

defines (0,n) :: Message_element_type :: defined_in (1,1)

Each MET must be defined in one HMD.

supports (1,1) :: Project :: implemented_by (0,n)

A MIM should support a single project.

Attributes of: **Hierarchical message description**

description : DescriptiveText

A short textual description of the messages covered in the HMD..

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

identifier : IdentifierString

Arbitrary identifier assigned by Technical Steering Committee. Committees may assign an interim identifier that starts with the committee's identifier, as "Cnn_<identifier>" so long as this composite identifier is unique.

name : NameString

The HMDs in the model are each given a name.

Class: **HL7 committee**

Associated with: **Data_type_category**
Interaction_model_category
Message_information_model
Model
Project
Subject_area
Use_case_category

Description of: **HL7_committee**

Unique identifier assigned to each of the Technical Committees and Special Interest Groups of the HL7 Working Group.

Associations for: **HL7_committee**

maintains (0,n) :: Data_type_category :: maintained_by (1,1)

maintains (0,n) :: Interaction_model_category :: maintained_by (0,1)

defines (0,n) :: Message_information_model :: defined_by (1,1)

prepares (0,n) :: Model :: prepared_by (1,1)

Links a model to the committee that prepared it.

responsible_for (0,n) :: Project :: responsibility_of (1,1)

Establishes the relationship between committees and the projects for which that committee is responsible.

maintains (0,n) :: Subject_area :: maintained_by (0,1)

maintains (0,n) :: Use_case_category :: maintained_by (0,1)

Attributes of: **HL7_committee**

facilitator : String

Name of the individual who facilitates modeling for this committee.

id : IdentifierString

Assigned committee identifier.

mission : DescriptiveText

The approved mission statement or charter for this committee.

name : String

The name of the Technical Committee or Special Interest Group.

Class: HMD_attribute_row

Subtype of: **HMD_row**

Associated with: **RMIM_attribute_row**

Description of: **HMD_attribute_row**

An attribute row represents a single attribute in the R-MIM.

Associations for: **HMD_attribute_row**

defined_by (1,1) :: RMIM_attribute_row :: defines (0,n)

Class: **HMD_class_row**

Subtype of: **HMD_row**

Associated with: **RMIM_class_row**

Description of: **HMD_class_row**

There is one class row in an HMD. This row is the root of the HMD.

Associations for: **HMD_class_row**

defined_by (1,1) :: RMIM_class_row :: defines (0,n)

Class: **HMD_domain_constraint**

Associated with: **Message_row_control**
Vocabulary_domain

Description of: **HMD_domain_constraint**

Constrains a coded HMD attribute row to a particular vocabulary domain. Links each coded attribute in an HMD to the code domain that may be used to value it.

For any class, the special attribute status_cd has as its domain all of the states of the class. In an HMD domain specification, the special domain name '@state' can substitute for the domain name. If held is a valid state, <@state> and <@state - (held)> are valid domain specifications.

Associations for: **HMD_domain_constraint**

constrains (0,n) :: Message_row_control :: has_domain (0,1)

links_domain (1,1) :: Vocabulary_domain :: is_constraint (0,n)

Attributes of: **HMD_domain_constraint**

realm : String

May specify the realm of applicability for this attribute row.

strength : String

The strength of the constraint is either CWE (coded with exceptions) or CNE (coded, no exceptions). If no value is given, CWE is the default.

Class: **HMD_notation**

Associated with: **Message_row_control**
RMIM_note

Associations for: **HMD_notation**

annotates (1,1) :: Message_row_control :: has_notation (0,n)

links_note (1,1) :: RMIM_note :: is_notation (0,n)

Attributes of: **HMD_notation**

type : Enumerated

Indicates the type of the note. Types include:

RV : Required value

CP : Conditional Presence

CN : Constraint

DM : Domain

CT : Comment

Class: **HMD_other_row**

Subtype of: **HMD_row**

Associated with: **RMIM_other_row**

Description of: **HMD_other_row**

Links an 'other' R-MIM row into a message.

Associations for: **HMD_other_row**

defined_by (1,1) :: RMIM_other_row :: defines (0,n)

Class: **HMD_relationship_row**

Subtype of: **HMD_row**

Associated with: **HMD_row**
RMIM_relationship_row

Description of: **HMD_relationship_row**

An relationship row represents a single association, aggregation or inheritance relationship traversed in the R-MIM graph walk.

Associations for: **HMD_relationship_row**

has_parent (1,1) :: HMD_row :: is_parent (0,n)

defined_by (1,1) :: RMIM_relationship_row :: defines (0,n)

Class: **HMD_row**

Supertype of: **HMD_attribute_row**
HMD_class_row
HMD_other_row
HMD_relationship_row

Is part of: **Hierarchical_message_description**

Associated with: **HMD_relationship_row**
Message_row_control

Description of: **HMD_row**
The rows of an HMD.

Composition for: **HMD_row**

is_part_of (1,1) :: Hierarchical_message_description :: contains (1,n)
A reference to the HMD that contains each HMD row.

Associations for: **HMD_row**

is_parent (0,n) :: HMD_relationship_row :: has_parent (1,1)

controlled_by (0,n) :: Message_row_control :: controls (1,1)
Each message row control controls the presence of one unsubsumed HMD row in the message structure of which the message row control is a part.

Attributes of: **HMD_row**

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

MET_source : Enumerated

Indicates the source of the MET - reuse a definition from this HMD, CMET or data type.

nest_level : Integer

Indicates the nesting level of the row in the HMD.

Class: Interaction

Is part of: **Model**

Associated with: **Application_role**
Application_role
Interaction
Interaction
Interaction_model_category
Interaction_sequence
Message_type

Trigger_event

Description of: Interaction

An association between a specific message (information transfer), a particular trigger event that initiates or triggers the interaction, and the roles that send and receive the interaction. An interaction is a single, one-way transfer of information. Within itself, an interaction may not specify a return message. An interaction may, however, establish a responsibility for the receiver of its message, and this responsibility may require that the receiver initiate a particular trigger event/interaction subsequent to the receipt. That follow-on interaction may have the effect of continuing or completing a transaction that requires two or more linked message exchanges.

Composition for: Interaction

in (1,1) :: Model :: has (0,n)

The relationship between interactions and the models of which they are a part.

Associations for: Interaction

received_by (1,1) :: Application_role :: receives (0,n)

A reference to the application role that is responsible for receiving the message involved in this interaction. The receiving role must be prepared to accept the message and to fulfill the receiver responsibility.

sent_by (1,1) :: Application_role :: sends (0,n)

The sending role has responsibilities to recognize the trigger event for the interaction and to cause the appropriate message to be sent.

initated_by_receiver (0,1) :: Interaction :: responsible_for (0,n)

A reference to an interaction that the receiver of the message must initiate once receipt of the message is acknowledged. This is an optional element in that there may no follow-on responsibility. Transactions can be established through a chain of receiver responsibilities for individual interactions.

responsible_for (0,n) :: Interaction :: initated_by_receiver (0,1)

A reference to an interaction that the receiver of the message must initiate once receipt of the message is acknowledged. This is an optional element in that there may no follow-on responsibility. Transactions can be established through a chain of receiver responsibilities for individual interactions.

included_in (0,n) :: Interaction_model_category :: includes (0,n)

is_linked_by (0,n) :: Interaction_sequence :: links (1,1)

transfers (1,1) :: Message_type :: transferred_by (1,n)

Each interaction shall include a link to a single message structure that the interaction will transfer.

initiated_by (1,1) :: Trigger_event :: initiates (0,n)

A reference to the trigger event that triggers or initiates this interaction.

Attributes of: **Interaction**

description : DescriptiveText

Provides a description of the data content of the interaction, usually expressed in terms of the class instances that are expected to be part of the message sent by the interaction.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

identifier : IdentifierString

An identifier assigned to the interaction. The identifier should be unique within the scope of the model in which the interaction is defined. In HL7, committees manage the unique identifiers for their interactions, and concatenate the committee identifier as "Cnn_<identifier>."

Class: **Interaction_model_category**

Is part of: **Model**

Associated with: **Application_role**
HL7_committee
Interaction
Interaction_model_category
Interaction_model_category

Description of: **Interaction_model_category**

A major category of information represented in the interaction model. An aggregation of interrelated interactions and application roles. A category allows portions of a large model to be viewed as a whole thereby eliminating some complexity involved in understanding a large model.

Composition for: **Interaction_model_category**

in (1,1) :: Model :: has (0,n)

The relationship between interaction model categories and the models of which they are a part.

Associations for: **Interaction_model_category**

includes (0,n) :: Application_role :: included_in (0,n)

maintained_by (0,1) :: HL7_committee :: maintains (0,n)

includes (0,n) :: Interaction :: included_in (0,n)

nested_in (0,1) :: Interaction_model_category :: nests (0,n)
Interaction model categories may be nested.

nests (0,n) :: Interaction_model_category :: nested_in (0,1)
Interaction model categories may be nested.

Attributes of: **Interaction_model_category**

description : DescriptiveText

Short informative text describing the interaction model category so as to be clear what type of interactions and application roles it includes.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

name : NameString

The name given to the interaction model category. The identifier for the committee defining this category is prepended to the name as Cnn.

Class: **Interaction_sequence**

Is part of: **Storyboard**

Associated with: **Interaction**

Description of: **Interaction_sequence**

Captures the sequence in which a particular interaction is included in a storyboard.

Composition for: **Interaction_sequence**

is_part_of (1,1) :: Storyboard :: contains (0,n)

Each storyboard is made up of a sequence of interactions, a sequence of use cases, or both.

Associations for: **Interaction_sequence**

links (1,1) :: Interaction :: is_linked_by (0,n)

Attributes of: **Interaction_sequence**

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

sequence_number : Integer

The order in which the interaction participates in the Storyboard.

Class: **LOINC_link**

Is part of: **Domain_version**

Associated with: **Vocabulary_domain**

Description of: **LOINC_link**

A linkage from a particular vocabulary domain to a particular LOINC code.

Composition for: **LOINC_link**

in_version (1,1) :: Domain_version :: has (0,n)

Associations for: **LOINC_link**

links_domain (0,1) :: Vocabulary_domain :: equates_to (0,n)

Attributes of: **LOINC_link**

LOINC_code : String

The LOINC code being equated to the vocabulary domain.

LOINC_version : String

The version of LOINC being referenced.

status : CodedElement

The status of the item. The values for Status come from vocabulary domain EditStatus. Some values for status are Proposed, Rejected, Active, Obsolete, and Inactive.

version_out : String

The version number of the table at which this entry was deleted. A blank Vout value means that the row continues to exist in the current version of the table.

Class: **Message_element_type**

Is Abstract Class

Supertype of: **Choice_MET**
Collection_MET
Composite_MET
Primitive_MET
Version_MET

Is part of: **Model**

Associated with: **Choice_branch**
Collection_MET
Hierarchical_message_description
MET_component
RMIM_row

Description of: **Message element type**

This is an abstract generalization for particular message element types (MET). It is also known simply as a type. A MET is a specification of the values that a message element can take on in its instances. It is the basic unit of structure for HL7 Version 3 messages and related information structures.

Composition for: **Message element type**

in (1,1) :: Model :: has (0,n)

The relationship between Message element types and the models in which they are first defined.

Associations for: **Message element type**

types (0,n) :: Choice_branch :: is_of_type (1,1)

Each choice branch must be typed by an MET.

collected_by (0,n) :: Collection_MET :: collects (1,1)

Each collection MET collects elements of a single type.

defined_in (1,1) :: Hierarchical_message_description :: defines (0,n)

Each MET must be defined in one HMD.

types (0,n) :: MET_component :: is_of_type (1,1)

Each MET component is typed by a single MET.

types (0,n) :: RMIM_row :: has_type (0,1)

Attributes of: **Message element type**

ballot_version : IdentifierString

The version of HL7 models under which this MET was first balloted, with its HMD.

definition_cd : Enumerated

Indicates whether this MET is defined within an HMD, as a CMET, a type for a data type, or is being re-used in an HMD.

formal_name : NameString

The formal name (sometimes called the "long name") is a very descriptive name that generally is the same as it appears in the source. Common sources for formal names are class, attribute, or association names from the Message Information Model, Object Views from the Message Object Diagram, Common Message Element Type definitions, and Data Type definitions.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

name : NameString

The name (sometimes called the "short name") is intended to be a mnemonic abbreviation of the formal name that is more tractable for programming, for reference as types in another MET and for use in Implementation Technology Specifications.

Class: Message information model

Composite of: **MIM_attribute**
 MIM_class
 MIM_relationship
 MIM_state

Associated with: **HL7_committee**
 Model
 Refined_message_information_model

Description of: Message information model

Is a subset of the HL7 Reference Information Model (RIM) that contains only those classes, attributes, states and relationships necessary to support the messages to be specified for a particular project. The acronym for message information model is MIM.

The elements of the MIM may have their optionality and/or cardinality constrained beyond the levels set in the RIM. For example, if a connection is optional in the RIM, it may be constrained to be mandatory in the MIM. Similarly, an attribute that may repeat any number of times in the RIM may be constrained in the MIM to a specific number of repeats.

In selecting classes, attributes, states and connections for inclusion in a MIM, a Technical Committee should follow the following process and rules. The TC is selecting the elements of a MIM that will contain: Classes; Attributes (perhaps with constrained optionality); States; Instance connections (perhaps with constrained cardinality); and whole part connections (perhaps with constrained cardinality).

The selection rules are:

- a) all selections must come from the RIM or from a single DIM that is a subset of the HL7 RIM;
- b) for every attribute selected, the committee must also select the class of which that attribute is a part;
- c) for every state selected, the committee must also select the class of which that state is a part;
- d) all state transitions between states selected for inclusion in the MIM shall be deemed members of the MIM, too.
- e) for every instance connection and whole part connection that is selected, the committee must also select both of the classes that are connected by that connection; and
- f) any class that is selected must meet one of the following two criteria:
 - i) the class contains at least one selected attribute, or
 - ii) the class participates in two selected connections (instance or whole part).

Composition for: **Message information model**

contains (1,n) :: MIM_attribute :: is_part_of (1,1)

A MIM is a container holding links to individual components of the information model.

contains (1,n) :: MIM_class :: is_part_of (1,1)

A MIM is a container holding links to individual components of the information model.

contains (1,n) :: MIM_relationship :: is_part_of (1,1)

A MIM is a container holding links to individual components of the information model.

contains (0,n) :: MIM_state :: is_part_of (1,1)

A MIM is a container holding links to individual components of the information model.

Associations for: **Message information model**

defined_by (1,1) :: HL7_committee :: defines (0,n)

draws_from (1,1) :: Model :: is_basis_for (0,n)

A reference to the model (RIM version) from which all of the elements of the MIM will be drawn.

refined_by (0,n) :: Refined_message_information_model :: refines (1,1)

Each R-MIM refines a single MIM.

Attributes of: **Message information model**

ballot_version : IdentifierString

The version of HL7 models in which this MIM was first balloted. Balloting will freeze the contents of the MIM. Future changes to the MIM must be made with a new MIM, established under a new id.

description : DescriptiveText

Describes the general set of messages (HMDs) that will be built from this MIM.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

id : IdentifierString

Arbitrary identifier assigned by Technical Steering Committee. Committees may assign an interim identifier that starts with the committee's identifier, as "Cnn_<identifier>" so long as this composite identifier is unique.

name : NameString

A brief descriptive name for the MIM. This may be the same as the name of the supported project.

Class: **Message MET**

Subtype of: **Composite_MET**

Description of: **Message MET**

A message MET types all of the messages in an HMD.

Class: **Message row control**

Is part of: **Message_type**

Associated with: **HMD_domain_constraint**
 HMD_notation
 HMD_row

Description of: **Message row control**

An element of a message type that controls the use of a particular HMD row in messages defined by the parent message type. The message row control has one subtype that relates to HMD attribute rows.

Composition for: **Message row control**

included_in (1,1) :: Message_type :: includes (0,n)

A reference to the message structure of which each message row control is a part.

Associations for: **Message_row_control**

has_domain (0,1) :: HMD_domain_constraint :: constrains (0,n)

has_notation (0,n) :: HMD_notation :: annotates (1,1)

controls (1,1) :: HMD_row :: controlled_by (0,n)

Each message row control controls the presence of one unsubsumed HMD row in the message structure of which the message row control is a part.

Attributes of: **Message_row_control**

conformance : Enumerated

Describes the requirement of information systems to send, or receive and process, this message element in order to claim conformance to the HL7 messaging standard defined by this message type. Values are: Required (R) and Not Required (N).

default_value : String

A notation that captures the default value for this row in the message type. It provides a value that a sending system may insert when creating a message instance if it has no other value to use.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

inclusion : Enumerated

Shows whether a message element may appear and if it may be null.. A mandatory or required message element may appear within an optional message element. If the outer message element, which is optional, actually appears in a message instance, any mandatory inner element must appear Possible values are: Mandatory (M), and Optional (O)..

repetitions : MultiplicityString

Describes whether the message element may repeat. Shows the minimum and maximum number of repetitions for this row (and its subordinates) in this message structure. It is not consistent to have a minimum number of repetitions of zero unless the Inclusion value is Optional.

Class: **Message_type**

Supertype of:	Union_message_type
Is part of:	Hierarchical_message_description
Composite of:	Message_row_control
Associated with:	Interaction Union_message_type

Description of: **Message_type**

A message type is part of an HMD. It defines the specific information transfer that occurs in an interaction to meet the requirements of use cases. It is a set of constraints applied to the message elements defined in the HMD. These are represented by a set of columns in the HMD. The content for those columns is specified by the message row control instances that are parts of the message type.

The message type is an atomic unit in that the entire information content defined by the message type will be sent in an interaction, or no part of it will be sent. No further decomposition is possible.

Composition for: **Message_type**

is_part_of (1,1) :: Hierarchical_message_description :: contains (1,n)

Each message structure is contained in a single HMD.

includes (0,n) :: Message_row_control :: included_in (1,1)

A reference to the message structure of which each message row control is a part.

Associations for: **Message_type**

transferred_by (1,n) :: Interaction :: transfers (1,1)

Each interaction shall include a link to a single message structure that the interaction will transfer.

combined_in (0,1) :: Union_message_type :: combines (1,n)

Attributes of: **Message_type**

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

identifier : IdentifierString

Arbitrary, unique identifier assigned by Technical Steering Committee. Committees may assign an interim identifier that starts with the committee's identifier, as "Cnn_<identifier>" so long as this composite identifier is unique.

isCommonType : Boolean

Identifies that this message type structure is common to all of the message types in this HMD.

Class: **MET_component**

Is part of: **Composite_MET**

Associated with: **Message_element_type**

Description of: **MET component**

Elements that make up a composite MET.

Each MET component has a name and a type. The names must be unique within the composite. Each component of an element must have a different name, although many may be of the same type.

Composition for: **MET component**

is_part_of (1,1) :: Composite_MET :: has_parts (1,n)

Associations for: **MET component**

is_of_type (1,1) :: Message_element_type :: types (0,n)

Each MET component is typed by a single MET.

Attributes of: **MET component**

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

is_optional : Boolean

Indicates whether this component is optional within the MET composite of which it is a part.

locale : Enumerated

Locale is intended to be a code. The intent is that all the information in a component flagged for a particular locale is specific to the definition for that locale. This allows for locale-specific information to be added to a message type. A scheme for generating unambiguous locale IDs is required.

In theory, the locale 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.

long_name : NameString

The full name for the component

name : NameString

This is the "short name" for the component.

sequence : Integer

Provides the sequence number within the composite for use in a segment-positional syntax such as ER7.

Class: MIM aggregation

Subtype of: **MIM_relationship**

Associated with: **Composite_aggregation**

Description of: **MIM aggregation**

A linking element that includes an individual aggregation relationship from the RIM into a particular MIM.

Associations for: **MIM aggregation**

includes (1,1) :: Composite_aggregation :: included_in (0,n)

Attributes of: **MIM aggregation**

part_multiplicity : MultiplicityString

Expresses a multiplicity for the part in this aggregation that applies to all uses of the aggregation in this MIM, and that is a tighter constraint than that recorded in the RIM. This attribute must be valued.

Class: MIM association

Subtype of: **MIM_relationship**

Associated with: **Association**

Description of: **MIM association**

A linking element that includes one individual association from the RIM into a particular MIM.

Associations for: **MIM association**

includes (1,1) :: Association :: included_in (0,n)

Attributes of: **MIM association**

source_multiplicity : MultiplicityString

Expresses a multiplicity for the source class in this association that applies to all uses of the association in this MIM, and that is a tighter constraint than that recorded in the RIM. This attribute must be valued.

target_multiplicity : MultiplicityString

Optional attribute expresses a multiplicity for the target class in this association that applies to all uses of the association in this MIM, and that is a tighter constraint than that recorded in the RIM. This attribute must be valued.

Class: **MIM_attribute**

Is part of: **Message_information_model**

Associated with: **Attribute**
Data_type
MIM_attribute_domain_constraint
RMIM_attribute_row

Description of: **MIM_attribute**

Includes individual attributes from the RIM into a particular MIM, provided that the parent class for the attribute is also in the MIM..

Composition for: **MIM_attribute**

is_part_of (1,1) :: Message_information_model :: contains (1,n)

A MIM is a container holding links to individual components of the information model.

Associations for: **MIM_attribute**

includes (1,1) :: Attribute :: included_in (0,n)

has_working (0,1) :: Data_type :: is_working (0,n)

If a MIM attribute is associated with a RIM attribute that does not have an assigned data type, the MIM attribute may be assigned an alternative, working data type. This association must not be instantiated otherwise.

constrained_by (0,1) :: MIM_attribute_domain_constraint :: constrains (0,n)

has_dependent (0,n) :: RMIM_attribute_row :: based_on (1,1)

Each R-MIM attribute is based on one MIM attribute.

Attributes of: **MIM_attribute**

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

inclusion : Boolean

This attribute expresses whether the inclusion of the attribute in the HMD and message structures derived from the HMD is mandatory. If the inclusion of the attribute is mandatory in the RIM, it must also be mandatory in the MIM. The default is the inclusion constraint for this attribute in the RIM.

repeatability : Boolean

This attribute expresses whether the attribute may repeat in a message. The repeatability applies to all uses of the attribute in this MIM, and must be a tighter constraint than that recorded in the RIM. The default for this attribute is the repeatability assigned to the attribute in the RIM.

Class: MIM attribute domain constraint

Associated with: **MIM_attribute**
 Vocabulary_domain

Description of: MIM attribute domain constraint

Constrains a coded MIM attribute to a particular vocabulary domain.

For any class, the special attribute status_cd has as its domain all of the states of the class.

Associations for: MIM attribute domain constraint

constrains (0,n) :: MIM_attribute :: constrained_by (0,1)

links_domain (1,1) :: Vocabulary_domain :: is_constraint (0,n)

Attributes of: MIM attribute domain constraint

strength : String

The strength of the constraint is either CWE (coded with exceptions) or CNE (coded, no exceptions). If no value is given, CWE is the default.

Class: MIM class

Is part of: **Message_information_model**

Associated with: **Class**
 RMIM_class_row

Description of: MIM class

Includes individual classes from the RIM into a particular MIM.

Composition for: MIM class

is_part_of (1,1) :: Message_information_model :: contains (1,n)

A MIM is a container holding links to individual components of the information model.

Associations for: MIM class

includes (1,1) :: Class :: included_in (0,n)

has_dependent (0,n) :: RMIM_class_row :: is_based_on (1,1)

Attributes of: MIM class

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

Class: MIM generalization

Subtype of: **MIM_relationship**

Associated with: **Generalization_relationship**

Description of: **MIM_generalization**

A linking element that includes one individual generalizations from the RIM into a particular MIM.

Associations for: **MIM_generalization**

includes (1,1) :: Generalization_relationship :: included_in (0,n)

Class: **MIM_relationship**

Is Abstract Class

Supertype of: **MIM_aggregation**
MIM_association
MIM_generalization

Is part of: **Message_information_model**

Associated with: **RMIM_relationship_row**

Description of: **MIM_relationship**

Provides an abstract generalization for the MIM links to association, aggregation, and generalization connections in the RIM.

Composition for: **MIM_relationship**

is_part_of (1,1) :: Message_information_model :: contains (1,n)

A MIM is a container holding links to individual components of the information model.

Associations for: **MIM_relationship**

has_dependent (0,n) :: RMIM_relationship_row :: is_based_on (1,1)

Attributes of: **MIM_relationship**

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

Class: **MIM_state**

Is part of: **Message_information_model**

Associated with: **RMIM_state_row**
State

Description of: **MIM_state**

Includes an individual state into the MIM, provided that the parent class for the state is also in the MIM.

Composition for: **MIM_state**

is_part_of (1,1) :: Message_information_model :: contains (0,n)

A MIM is a container holding links to individual components of the information model.

Associations for: **MIM_state**

has_dependent (0,n) :: RMIM_state_row :: is_based_on (1,1)

includes (1,1) :: State :: included_in (0,n)

Attributes of: **MIM_state**

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

Class: **Model**

Composite of: **Actor**
 Application_role
 Class
 Data_type
 Data_type_category
 Hierarchical_message_description
 Interaction
 Interaction_model_category
 Message_element_type
 Refined_message_information_model
 Storyboard
 Subject_area
 Trigger_event
 Use_case
 Use_case_category

Associated with: **HL7_committee**
 Message_information_model

Description of: **Model**

This class in the meta-model contains the elements necessary to uniquely define an aggregate model, to establish its provenance and scope, and to link it to each of the elements that make up that model.

A model is a collection of subject areas, scenarios, classes, attributes, use cases, actors, trigger events, interactions, etc. that depict the information needed to specify HL7 Version3 messages. This model is further divided into four specific models - a use case model, an information model, an interaction model, and a message design model..

Composition for: **Model**

has (0,n) :: Actor :: in (1,1)

The relationship between actors and the models of which they are a part.

has (0,n) :: Application_role :: in (1,1)

The relationship between application roles and the models of which they are a part.

has (0,n) :: Class :: in (1,1)

The relationship between classes and the models of which they are a part.

has (0,n) :: Data_type :: in (1,1)

The relationship between data types and the models in which they are first defined.

has (0,n) :: Data_type_category :: in (1,1)

The relationship between data type categories and the models of which they are a part.

has (0,n) :: Hierarchical_message_description :: in (1,1)

The relationship between hierarchical message descriptions and the models in which they are first defined.

has (0,n) :: Interaction :: in (1,1)

The relationship between interactions and the models of which they are a part.

has (0,n) :: Interaction_model_category :: in (1,1)

The relationship between interaction model categories and the models of which they are a part.

has (0,n) :: Message_element_type :: in (1,1)

The relationship between Message element types and the models in which they are first defined.

contains (0,n) :: Refined_message_information_model :: is_part_of (1,1)

has (0,n) :: Storyboard :: in (1,1)

The relationship between scenarios and the models of which they are a part.

has (0,n) :: Subject_area :: in (1,1)

The relationship between subject areas and the models of which they are a part.

has (0,n) :: Trigger_event :: in (1,1)

Sets relationship between model elements and the models of which they are a part.

has (0,n) :: Use_case :: in (1,1)

The relationship between use cases and the models of which they are a part.

has (0,n) :: Use_case_category :: in (1,1)

The relationship between use case model categories and the models of which they are a part.

Associations for: **Model**

prepared_by (1,1) :: HL7_committee :: prepares (0,n)

Links a model to the committee that prepared it.

is_basis_for (0,n) :: Message_information_model :: draws_from (1,1)

A reference to the model (RIM version) from which all of the elements of the MIM will be drawn.

Attributes of: **Model**

description : DescriptiveText

A short narrative describing the scope and intent of the model.

developing_org : String

A short form identifier of the organization responsible for the publication and maintenance of the model. . For HL7, this name shall be "HL7."

last_modified_date : Date

The date the model was last modified by the model developing organization.

modelID : NameString

A unique identifier assigned to the model by the developing organization. In HL7, these identifiers will be assigned by the Modeling and Methodology Committee.

name : NameString

A descriptive title for the model. The name in combination with the version number shall be unique within the set of models developed by any particular model developing organization.

version_number : VersionNumber

A number showing the release level of the model. The version number, in combination with the name, shall be unique for all public releases of the model.

Class: **Primitive data type**

Subtype of: **Data_type**

Associated with: **Primitive_MET**

Description of: **Primitive_data_type**

A primitive data type is a data type that is defined entirely by its specification. A primitive data type may have generic type parameters.

Associations for: **Primitive_data_type**

types (0,n) :: Primitive_MET :: is_of_type (1,1)

A Primitive MET gains its own type from a Primitive data type.

Class: **Primitive_MET**

Subtype of: **Message_element_type**

Associated with: **Primitive_data_type**

Description of: **Primitive_MET**

A message element type that does not contain other message elements. A primitive MET may only be defined as part of a data type MET (DMET).

Associations for: **Primitive_MET**

is_of_type (1,1) :: Primitive_data_type :: types (0,n)

A Primitive MET gains its own type from a Primitive data type.

Class: **Project**

Associated with: **Hierarchical_message_description**
HL7_committee

Description of: **Project**

The specification of a particular, coherent set of messages and events based around one (or a few) Subject Classes. A project is undertaken to address a current need for standardizing information that flows between a number of parties in healthcare. A project also defines a ballot package that might be advanced independently of other HL7 ballot packages.

Associations for: **Project**

implemented_by (0,n) :: Hierarchical_message_description :: supports (1,1)

A MIM should support a single project.

responsibility_of (1,1) :: HL7_committee :: responsible_for (0,n)

Establishes the relationship between committees and the projects for which that committee is responsible.

Attributes of: **Project**

ANSI_PINS_date : Date

The date on which the project scope was published in the ANSI PINS system.

id : IdentifierString

Arbitrary identifier assigned by Technical Steering Committee.

name : NameString

Brief descriptive name for the project.

scope : DescriptiveText

The approved scope statement for the project. It defines the area of healthcare functionality that needs to be supported by HL7 messaging and is a high level use case that encompasses the entire project.

TSC_approval_date : Date

Date on which the project was approved by the TSC.

Class: Refined message information model

Is part of:	Model
Composite of:	RMIM_note RMIM_row
Associated with:	Message_information_model

Description of: Refined message information model

The Refined message information model (R-MIM) is a hybrid between a class model and an object model. Each class, attribute and association in the MIM is part of the R-MIM.

In addition, classes in the MIM may be cloned in the R-MIM. That is, they may be represented more than once. This is done to allow the messages to be tailored to the specific needs of different instances of a class. For example, both the Person class has roles for both patients and doctors. By and large, the attributes and associations of Person that are important to the patient role are different from those important to the doctor role. Cloning allows these differences to be represented explicitly in the R-MIM.

When a class is cloned, the clone must be given a unique name, and the original may be re-named, as well. Both the clone and the original may be constrained (beyond the constraints of the MIM) independently. Constraints involve removing attributes, tightening association cardinality, discarding associations and reducing vocabulary domains.

The R-MIM is displayed both as a UML diagram, and in a two-level tabular format in which each class and clone is a primary row, and the attributes and associations of those classes comprise the second-level rows. Selected attributes of the meta-model provide 'layout' information for the tabular format.

Composition for: **Refined message information model**

is_part_of (1,1) :: Model :: contains (0,n)

contains (0,n) :: RMIM_note :: part_of (1,1)

contains (1,n) :: RMIM_row :: part_of (1,1)

Associations for: **Refined message information model**

refines (1,1) :: Message_information_model :: refined_by (0,n)
Each R-MIM refines a single MIM.

Attributes of: **Refined message information model**

first_node_id : IdentifierString
Identifies the first class node in the tabular display of the R-MIM.

history : CompoundHx
A compound data element that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

identifier : IdentifierString
A unique identifier for the R-MIM.

Class: **RMIM attribute domain constraint**

Associated with: **RMIM_attribute_row**
Vocabulary_domain

Description of: **RMIM attribute domain constraint**
Constrains a coded RMIM attribute row to a particular vocabulary domain.

For any class, the special attribute status_cd has as its domain all of the states of the class.

Associations for: **RMIM attribute domain constraint**

constrains (0,n) :: RMIM_attribute_row :: constrained_by (0,1)

links_domain (1,1) :: Vocabulary_domain :: is_constraint (0,n)

Attributes of: **RMIM attribute domain constraint**

strength : String
The strength of the constraint is either CWE (coded with exceptions) or CNE (coded, no exceptions). If no value is given, CWE is the default.

Class: **RMIM attribute row**

Subtype of: **RMIM_row**

Associated with: **HMD_attribute_row**
MIM_attribute
RMIM_attribute_domain_constraint

Description of: **RMIM_attribute_row**
Expresses the presence of selected attributes in the R-MIM.

Associations for: **RMIM_attribute_row**

defines (0,n) :: HMD_attribute_row :: defined_by (1,1)

based_on (1,1) :: MIM_attribute :: has_dependent (0,n)
Each R-MIM attribute is based on one MIM attribute.

constrained_by (0,1) :: RMIM_attribute_domain_constraint :: constrains (0,n)

Class: **RMIM_class_row**

Subtype of: **RMIM_row**

Associated with: **HMD_class_row**
MIM_class
RMIM_relationship_row
RMIM_row
RMIM_row

Description of: **RMIM_class_row**
Expresses the presence of selected classes or clones thereof in the R-MIM.

Associations for: **RMIM_class_row**

defines (0,n) :: HMD_class_row :: defined_by (1,1)

is_based_on (1,1) :: MIM_class :: has_dependent (0,n)

is_related_by (0,n) :: RMIM_relationship_row :: has_distal_class (0,1)

is_active_parent (0,n) :: RMIM_row :: has_active_parent (1,1)
Shows the active parent relationship for an inherited row. Reflects the combined effects of inheritance and cloning.

is_true_parent (0,n) :: RMIM_row :: has_true_parent (1,1)
Establishes the true parent for each association and attribute. Is required because the act of cloning precludes determining this association through the information model or MIM.

Attributes of: **RMIM_class_row**

first_attribute_row_id : IdentifierString

Pointer to the first child attribute row for this class row.

first_relation_row_id : IdentifierString

Pointer to the first child relationship row for this class row.

Class: **RMIM_notation**

Associated with: **RMIM_note**
 RMIM_row

Associations for: **RMIM_notation**

links_note (1,1) :: RMIM_note :: is_notation (0,n)

annotates (1,1) :: RMIM_row :: has_notation (0,n)

Attributes of: **RMIM_notation**

type : Enumerated

Indicates the type of the note. Types include:

RV : Required value

CP : Conditional Presence

CN : Constraint

DM : Domain

CT : Comment

Class: **RMIM_note**

Is part of: **Refined_message_information_model**

Associated with: **HMD_notation**
 RMIM_notation

Description of: **RMIM_note**

A note may be placed on any row of an R-MIM or of a Hierarchical Message Definition. These notes clarify the semantic intent of the Technical Committee.

Composition for: **RMIM_note**

part_of (1,1) :: Refined_message_information_model :: contains (0,n)

Associations for: **RMIM_note**

is_notation (0,n) :: HMD_notation :: links_note (1,1)

is_notation (0,n) :: RMIM_notation :: links_note (1,1)

Attributes of: **RMIM_note**

number : Integer

Notes within an HMD are identified by number.

subject : String

Captures the subject, a column of the HMD, to which the note applies.

Class: **RMIM_other_row**

Subtype of: **RMIM_row**

Associated with: **HMD_other_row**
RMIM_row

Description of: **RMIM_other_row**

Expresses the presence of special rows in the R-MIM. There are two types of such additional rows:

item : Represents the individual elements of the message that make up a Set or List of elements, as required by repeating attributes or associations.

stc : Represents the presence of a sub-component of a data type in the message. These components are exposed in to allow the expression of constraints against them.

Associations for: **RMIM_other_row**

defines (0,n) :: HMD_other_row :: defined_by (1,1)

has_parent (1,1) :: RMIM_row :: is_parent (0,n)

Each 'other' node arises as a result of some other node, its parent.

Attributes of: **RMIM_other_row**

otherType : Enumerated

Coded value for the type of the other row. See definition of the RMIM_other_row class for the code values and their meaning.

Class: **RMIM_relationship_row**

Subtype of: **RMIM_row**

Associated with: **HMD_relationship_row**
MIM_relationship
RMIM_class_row
RMIM_relationship_row
RMIM_relationship_row

Description of: **RMIM_relationship_row**

Expresses the presence of selected association nodes (UML roles) in the R-MIM. Each relationship in the MIM and R-MIM produces two rows in the tabular R-MIM, one for the appearance of each end of the relationship in one of the R-MIM classes or clones.

Associations for: **RMIM_relationship_row**

defines (0,n) :: HMD_relationship_row :: defined_by (1,1)

is_based_on (1,1) :: MIM_relationship :: has_dependent (0,n)

has_distal_class (0,1) :: RMIM_class_row :: is_related_by (0,n)

has_other_half (1,1) :: RMIM_relationship_row :: is_other_half (0,1)

Each RMIM relationship row may be paired with a second such row to comprise a complete relationship.

is_other_half (0,1) :: RMIM_relationship_row :: has_other_half (1,1)

Each RMIM relationship row may be paired with a second such row to comprise a complete relationship.

Attributes of: **RMIM_relationship_row**

blocked : Boolean

Expresses whether this half-relationship is intended to be followed in building an HMD. This value is not normative. It may be over-ridden. When the R-MIM is diagrammed in UML, the presence of a blocked path is shown by making the UML role for the other end of the relationship unnavigateable.

Class: **RMIM_row**

Is Abstract Class

Supertype of: **RMIM_attribute_row**
RMIM_class_row
RMIM_other_row
RMIM_relationship_row
RMIM_state_row

Is part of: **Refined_message_information_model**

Associated with: **Message_element_type**
RMIM_class_row

RMIM_class_row
RMIM_notation
RMIM_other_row

Description of: **RMIM_row**

The R-MIM is modeled by the Rows that make up its tabular expression.

Composition for: **RMIM_row**

part_of (1,1) :: Refined_message_information_model :: contains (1,n)

Associations for: **RMIM_row**

has_type (0,1) :: Message_element_type :: types (0,n)

has_active_parent (1,1) :: RMIM_class_row :: is_active_parent (0,n)

Shows the active parent relationship for an inherited row. Reflects the combined effects of inheritance and cloning.

has_true_parent (1,1) :: RMIM_class_row :: is_true_parent (0,n)

Establishes the true parent for each association and attribute. Is required because the act of cloning precludes determining this association through the information model or MIM.

has_notation (0,n) :: RMIM_notation :: annotates (1,1)

is_parent (0,n) :: RMIM_other_row :: has_parent (1,1)

Each 'other' node arises as a result of some other node, its parent.

Attributes of: **RMIM_row**

cardinality : MultiplicityString

Expresses the minimum and maximum number of occurrences for an association or an attribute in the context of the R-MIM.

conformance : Enumerated

Expresses the constraints on this row for conformance testing. Possible values are:

R: required for conformance

(blank): not required for conformance

NP: not permitted to appear in his message variant (only used in message row controls, not in R-MIM).

constraint : DescriptiveText

Captures constraints and notes for a given row in the R-MIM and HMD. The type of note and its contents must both be expresses. The types provided for include:

Required value : states a value and, in the presence of repetition, how many times the value can/must appear

Conditional Presence : states a value that must or must not be present based on the value of another element or sub-component that is higher in the HMD

Constraint : a verbal expression of a constraint

Domain : a domain specification, as described in the vocabulary chapter

Comment : any general comment; this label should not be used for items that can be described with any of the other labels.

default_update_mode : String

A coded value drawn from the possible modes of updating that occur when an attribute is received by a system that already contains values for that attribute. The update modes and their codes are:

R : replace (this is the default)

D : delete

I : ignore

NA : not applicable

V : verify: confirm that it exists

K : key: when creating an element store it; when updating an element confirm that it exists.

The following codes apply when updating individual items in a set:

ESA : edit set: add item

ESC : edit set: change item

ESD : edit set: delete item

ESAC : edit set: add or, if the item exists, change item

default_value : DescriptiveText

The default value for this attribute. If this field is blank or Null, the default is 'NULL'. If the field contains 'No' then no default specified. Otherwise, the field contains the value of the default and if the value is an integer it must be enclosed in quotes.

history : CompoundHx

A compound data element that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

mandatory : Boolean

If this attribute has a value of "True" then this message element must have a non-null value in order for the receiver to process the message.

name : NameString

The name of the row in the R-MIM. Rows representing attributes should not be renamed.

next_sibling_ID : IdentifierString

Points to the next sibling node in the tabular R-MIM.

previous_sibling_id : IdentifierString

Points to the previous sibling node in the tabular R-MIM.

short_name : NameString

A shortened version of the name for the row.

update_mode_set : String

A set of utterances from the list of values for 'default_update_mode.'. The sender may change the update mode instance by instance to any of the values in this list.

Class: RMIM_state_row

Subtype of: **RMIM_row**

Associated with: **MIM_state**

Description of: **RMIM_state_row**

Expresses the presence of selected states in the R-MIM.

Associations for: **RMIM_state_row**

is_based_on (1,1) :: MIM_state :: has_dependent (0,n)

Class: State

Is part of: **Subject_class**

Associated with: **MIM_state**
State
State
State_transition
State_transition

Description of: **State**

The identification of a unique combination of attribute value(s) and connections which are of interest about a subject class.

The enumerated States for each Subject class must include an "Initial state" from which some designated state transition moves the class to one or more active states.

Composition for: **State**

in (1,1) :: Subject_class :: has (0,n)

This relationship between states and the subject classes of which they are a part.

Associations for: **State**

included_in (0,n) :: MIM_state :: includes (1,1)

has_substate (0,n) :: State :: is_substate_of (0,1)

States may be defined as sub-states of a parent, provided that all of the states for a given class have unique names.

is_substate_of (0,1) :: State :: has_substate (0,n)

States may be defined as sub-states of a parent, provided that all of the states for a given class have unique names.

ends (0,n) :: State_transition :: ends_in (1,1)

is_start_of (0,n) :: State_transition :: starts_from (1,1)

Attributes of: **State**

description : DescriptiveText

A short informative description of the State.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

Note, the version data must fit within the range of the applicable versions for the class of which this element is a part.

name : NameString

The name of this particular state which shall be unique within this class.

predicate : String

The condition or set of conditions that when true about the attribute(s) and connections of the parent subject class identifies that the subject class is in the declared state.

Class: **State transition**

Associated with: **State**
 State
 Trigger_event
 Use_case

Description of: **State transition**

Captures the semantics of transitions from state-to-state for the Subject classes. Note that a legal transition may return to the same state from which it started.

State transitions also are a critical link between leaf-level use cases from which the transitions stem, and the trigger events identified with the transitions.

Associations for: **State transition**

ends_in (1,1) :: State :: ends (0,n)

starts_from (1,1) :: State :: is_start_of (0,n)

identified_by (0,1) :: Trigger_event :: identifies (1,n)

This connection from state transition to trigger event is only optional because we do not expect all possible trigger events to be defined in HL7. Nevertheless, any state transition that is described in a use case must be linked to a trigger event.

Note that because of the above condition, and because of the condition on the instance connection from a use case to a state transition, there is a mandatory (1,1) relationship from any leaf-level use case to one trigger event.

The relationship from trigger event to state transition is (1..*) because although the event will probably drive the Subject class to a single state (e.g. "Canceled") and the transitions may start from several states. Thus one trigger event may identify many transitions.

captured_in (0,n) :: Use_case :: describes (0,1)

A leaf-level use case describes the events that result in a single state transition of the subject class.

Although the instance connection from use case to state transition is optional, this is only because not all use cases are leaf-level. At the leaf-level, each use case should describe one and only one state transition.

Attributes of: **State transition**

description : DescriptiveText

A short, informative description of the state transition.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

Note, the version data must fit within the range of the applicable versions for both of the states to which this transition links.

label : NameString

A word or phrase that reflects the event that causes the transition. The name shall be unique with respect to the state from which the transition starts. This label may be the same as the name of the trigger event identified with the Transition.

Class: Storyboard

Is part of: **Model**

Composite of: **Interaction_sequence**
 Storyboard_example
 Use_case_sequence

Description of: Storyboard

A storyboard is a statement of health care relevant events defined as a sequence of leaf-level use cases or interactions. The storyboard provides one set of use case instances that the modeling committee expects will typically occur in the domain.

A storyboard may also be expressed as a subset of the interaction model in which case the representation includes all interactions that are implied by the trigger events associated with the sequence of use cases or are implied by the sender and receiver responsibilities of those interactions. Usually, an interaction diagram is constructed to show a group of interactions for a single storyboard.

The collection of storyboards that HL7 may publish does not limit the ways in which HL7 can be applied; other combinations of trigger events, interactions, and application roles that are consistent with the interaction model may also be used.

Composition for: Storyboard

contains (0,n) :: Interaction_sequence :: is_part_of (1,1)

Each storyboard is made up of a sequence of interactions, a sequence of use cases, or both.

in (1,1) :: Model :: has (0,n)

The relationship between scenarios and the models of which they are a part.

exemplifies (0,n) :: Storyboard_example :: is_part_of (1,1)

Each storyboard example provides a real world example for a single storyboard..

contains (0,n) :: Use_case_sequence :: contains (1,1)

Each storyboard is made up of a sequence of interactions, a sequence of use cases, or both.

Attributes of: Storyboard

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

identifier : IdentifierString

An identifier assigned to the storyboard to simplify references to it. The identifier should be unique within the scope of the model in which it is defined. In HL7, committees manage the unique identifiers for their storyboards, and concatenate the committee identifier as "Cnn_<identifier>."

name : NameString

A short phrase that provides a descriptive title for the storyboard.

purpose : DescriptiveText

The purpose for which the storyboard was created. Frequently it describes the generic set of actions that the storyboard represents.

Class: Storyboard example

Is part of: **Storyboard**

Description of: **Storyboard example**

Provides a real-world example of the sequence of events captured in a Storyboard.

Composition for: **Storyboard example**

is_part_of (1,1) :: Storyboard :: exemplifies (0,n)

Each storyboard example provides a real world example for a single storyboard..

Attributes of: **Storyboard example**

description : DescriptiveText

A narrative example from the real world that describes a set of events represented by the sequence of use cases that make up the Storyboard which this example exemplifies.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

identifier : IdentifierString

A unique identifier assigned to the storyboard example.

Class: Subject area

Is part of: **Model**

Associated with: **Class**
 Class
 HL7_committee
 Subject_area
 Subject_area

Description of: **Subject area**

A major category of information represented in the information model. An aggregation of interrelated classes. A subject area allows portions of a large model to be viewed as a whole thereby eliminating some complexity involved in understanding a large model.

Subject areas will also be used by the Methodology and Modeling Committee of HL7 to designate Domain Information Models (DIM), class stewardship responsibilities, and classes of interest to a particular committee.

Composition for: **Subject_area**

in (1,1) :: Model :: has (0,n)

The relationship between subject areas and the models of which they are a part.

Associations for: **Subject_area**

holds (1,n) :: Class :: primarily_resides_in (0,1)

The linkage between a Class and the Subject area that is its primary residence. This must be established if a Class resides in more than one Subject area.

includes (1,n) :: Class :: appears_in (0,n)

The linkage between a Subject area and each of the Classes that are in that Subject area.

maintained_by (0,1) :: HL7_committee :: maintains (0,n)

is_nested_in (0,1) :: Subject_area :: nests (0,n)

The linkage between two subject areas where one of the two is nested within the other.

nests (0,n) :: Subject_area :: is_nested_in (0,1)

The linkage between two subject areas where one of the two is nested within the other.

Attributes of: **Subject_area**

description : DescriptiveText

Short informative text describing the subject area so as to be clear what type of Classes it includes.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

name : NameString

The name given to the subject area. A subject area name is often the plural form of the name of the central or dominant class within the subject area. Subject area names shall be unique within a given model.

Class: **Subject_class**

Subtype of: **Class**

Composite of: **State**

Associated with: **Application_role**
 Attribute
 Trigger_event
 Use_case

Description of: **Subject_class**

A specialization of Class that is used in HL7 to identify those classes that are the focus for a set of Use Cases, Trigger events and/or Application Roles.

Composition for: **Subject_class**

has (0,n) :: State :: in (1,1)

This relationship between states and the subject classes of which they are a part.

Associations for: **Subject_class**

subject_of (0,n) :: Application_role :: relates_to (1,1)

Links each Application role to the Subject class for which it plays a role. The nature of this relationship is stereotyped as discussed in the description for the Application_role.

has_state_attribute (1,1) :: Attribute :: is_state_attribute_for (0,1)

The state attribute of a class contains a value indicating the current state of the class. In the event that the class has concurrent states, the attribute must be a set of state values.

is_driven_by (1,n) :: Trigger_event :: affects (1,1)

Reflects the fact that although a trigger event may cause multiple state transitions, all of these transitions will be within the states of a single subject class.

subject_of (0,n) :: Use_case :: describes (0,1)

Links a leaf-level use case to its Subject Class.

Class: **Trigger_event**

Is part of: **Model**

Associated with: **Interaction**
 State_transition
 Subject_class

Description of: **Trigger_event**

An occurrence in the health care domain, or within the systems that support this domain, that causes information to be exchanged in the domain or between systems. Trigger events are initiators of Interactions.

Each Trigger event is tied to one State transition for one of the Subject classes in the model. In turn, the State transition traces to a leaf-level Use case from which the transition stems, and the leaf-level use case defines the context for the trigger event.

Composition for: **Trigger event**

in (1,1) :: Model :: has (0,n)

Sets relationship between model elements and the models of which they are a part.

Associations for: **Trigger event**

initiates (0,n) :: Interaction :: initiated_by (1,1)

A reference to the trigger event that triggers or initiates this interaction.

identifies (1,n) :: State_transition :: identified_by (0,1)

This connection from state transition to trigger event is only optional because we do not expect all possible trigger events to be defined in HL7. Nevertheless, any state transition that is described in a use case must be linked to a trigger event.

Note that because of the above condition, and because of the condition on the instance connection from a use case to a state transition, there is a mandatory (1,1) relationship from any leaf-level use case to one trigger event.

The relationship from trigger event to state transition is (1..*) because although the event will probably drive the Subject class to a single state (e.g. "Canceled") and the transitions may start from several states. Thus one trigger event may identify many transitions.

affects (1,1) :: Subject_class :: is_driven_by (1,n)

Reflects the fact that although a trigger event may cause multiple state transitions, all of these transitions will be within the states of a single subject class.

Attributes of: **Trigger event**

dependency : String

If the occurrence of the trigger event is dependent upon the state of one or more objects in the domain or upon the prior occurrence of a different trigger event, this dependency will be expressed in this textual component.

description : DescriptiveText

The text that describes the trigger event. When viewed along with the description of the State transitions which the event identifies, this description must have sufficient detail that the event can be reliably recognized when it occurs.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

identifier : IdentifierString

An identifier assigned to the trigger event. The identifier is unique within the scope of the model in which the trigger event is defined. In HL7, committees manage the unique identifiers for their trigger events, and concatenate the committee identifier as "Cnn_<identifier>."

name : NameString

A name assigned to the trigger event. The name is unique within the scope of the model in which the trigger event is defined.

Class: Union message type

Subtype of: **Message_type**

Associated with: **Message_type**

Associations for: **Union message type**

combines (1,n) :: Message_type :: combined_in (0,1)

Class: Use case

Is part of: **Model**

Associated with: **Actor**
State_transition
Subject_class
Use_case_category
Use_case_relationship
Use_case_relationship
Use_case_sequence

Description of: **Use case**

A use case is a summary of health care relevant events and related information system events that reflect the usage of the information in the information model and related business models. Use cases describe the interactions and information interchanges that occur in the healthcare domain and the events that cause these interchanges.

Use cases may be expressed at various levels. A use case may be a parent to several child use cases. In this circumstance, the interactions ascribed to all of the children constitute the complete interaction of the parent. The decomposition to child use cases should stop when the resulting use case involves a single actor and a single interaction in response to a single stimulus - an atomic or leaf level use case. Note that a use case may be a child of more than one parent, but must not be defined such that a trace up the parental tree from a child will run into the same child (recursion).

At the lowest level of decomposition, each leaf level use case should link to only a single state transition which, in turn, links to a trigger event that initiates interactions. If the linkage is to multiple such transitions or events, further decomposition should be considered.

Composition for: Use case

in (1,1) :: Model :: has (0,n)

The relationship between use cases and the models of which they are a part.

Associations for: Use case

involves (1,n) :: Actor :: participates_in (0,n)

describes (0,1) :: State_transition :: captured_in (0,n)

A leaf-level use case describes the events that result in a single state transition of the subject class.

Although the instance connection from use case to state transition is optional, this is only because not all use cases are leaf-level. At the leaf-level, each use case should describe one and only one state transition.

describes (0,1) :: Subject_class :: subject_of (0,n)

Links a leaf-level use case to its Subject Class.

included_in (0,n) :: Use_case_category :: includes (0,n)

is_source_for (0,n) :: Use_case_relationship :: links_source (1,1)

Links a use case relationship to the source of the relationship.

is_target_in (0,n) :: Use_case_relationship :: links_target (1,1)

Links the use case relationship to its target or destination.

is_linked (0,n) :: Use_case_sequence :: links (1,1)

Attributes of: Use case

description : DescriptiveText

The text that describes the use case and provides the details necessary to understand the events that are involved in the use case.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

identifier : IdentifierString

An identifier assigned to the use case. The identifier is unique within the scope of the model. In HL7, committees manage the unique identifiers for their use cases, and concatenate the committee identifier as "Cnn_<identifier>."

name : NameString

A short phrase that provides a descriptive name for the use case. The name should be unique within the scope of use cases defined by a particular committee.

Class: Use case category

Is part of: **Model**

Associated with: **Actor**
 HL7_committee
 Use_case
 Use_case_category
 Use_case_category

Description of: Use case category

A major category of information represented in the use case model. An aggregation of interrelated actors and use cases. A category allows portions of a large model to be viewed as a whole thereby eliminating some complexity involved in understanding a large model.

Composition for: Use case category

in (1,1) :: Model :: has (0,n)

The relationship between use case model categories and the models of which they are a part.

Associations for: Use case category

includes (0,n) :: Actor :: included_in (0,n)

maintained_by (0,1) :: HL7_committee :: maintains (0,n)

includes (0,n) :: Use_case :: included_in (0,n)

nested_in (1,1) :: Use_case_category :: nests (0,n)

Use case categories may be nested in a hierarchy.

nests (0,n) :: Use_case_category :: nested_in (1,1)

Use case categories may be nested in a hierarchy.

Attributes of: Use case category

description : DescriptiveText

Short informative text describing the use case category so as to be clear what type of use cases it includes.

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

name : NameString

The name given to the use case category. The identifier for the committee defining this category is prepended to the name as Cnn.

Class: Use case relationship

Associated with: **Use_case**
 Use_case

Description of: Use case relationship

Use cases maintain a variety of relationships. This class captures all such flavors. See the use case model chapter of the MDF for details of the stereotypical relationships.

Associations for: Use case relationship

links_source (1,1) :: Use_case :: is_source_for (0,n)

Links a use case relationship to the source of the relationship.

links_target (1,1) :: Use_case :: is_target_in (0,n)

Links the use case relationship to its target or destination.

Attributes of: Use case relationship

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

stereotype : String

Identifies the stereotype for the use case relationship. A blank or null value is a simple generalization relationship, meaning that the target use case Use Case 1 adds additional behavior to the source use case. A value of "extends" means that the target use case adds additional behavior to the source use case at a specified Variation Point. A value of "includes" means that the target uses the source as part of its execution.

Class: Use case sequence

Is part of: **Storyboard**

Associated with: **Use_case**

Description of: Use case sequence

Captures the sequence in which a particular use case is enacted in a storyboard.

Composition for: Use case sequence

contains (1,1) :: Storyboard :: contains (0,n)

Each storyboard is made up of a sequence of interactions, a sequence of use cases, or both.

Associations for: Use case sequence

links (1,1) :: Use_case :: is_linked (0,n)

Attributes of: Use case sequence

history : CompoundHx

This is a compound data type that holds the id and previous_ID for history, and the first_ver and last_ver for versioning.

sequence : Integer

The order in which the use case participates in the Storyboard.

Class: V23 data type

Associated with: **Attribute**
 V23_fields

Description of: V23 data type

The definition of the data type as specified in Figure 2.2, Chapter 2 of HL7 Version 2.3.

Associations for: V23 data type

typed (0,n) :: Attribute :: had_V23_type (1,1)

Provides an indication of the data type used in Version 2.x for a particular attribute, if such prior usage has been identified.

types (0,n) :: V23_fields :: is_of_type (1,1)

Expresses that each field in V2.3 is typed with a data type.

Attributes of: V23 data type

data_type_category : String

The type of data type, such as alphanumeric, or chapter-specific.

data_type_code : String

The two- or three-character code for the data type. e.g. CM

data_type_name : String

The name for the data type as specified in the HL7 Chapter 2 table.

notes_format : String

The format of the data type, particularly for compound data types.

Class: V23 field segment

Associated with: **V23_fields**
 V23_segments

Description of: **V23_field_segment**

Links the fields in HL7 V2.3 to the segments in which those fields appear. Source is the HL7 data dictionary.

Associations for: **V23_field_segment**

positions (1,1) :: V23_fields :: populate (0,n)

Links each field to the segment(s) in which it is used and the sequential position in which it appears.

is_in (1,1) :: V23_segments :: contains (1,n)

Aggregates the fields that make up each segment.

Attributes of: **V23_field_segment**

sequence : Integer

The sequence number at which the element or field appears in the segment.

Class: **V23_fields**

Associated with: **Attribute**
 V23_data_type
 V23_field_segment

Description of: **V23_fields**

Contains all of the fields (data elements) specified in HL7 Version 2.x, as captured in the data dictionary published by HL7.

Associations for: **V23_fields**

is_source_for (0,n) :: Attribute :: based_on (0,n)

Provides a linkage for an information model attribute to its equivalent version 2.x field, if such linkage exists and has been identified.

is_of_type (1,1) :: V23_data_type :: types (0,n)

Expresses that each field in V2.3 is typed with a data type.

populate (0,n) :: V23_field_segment :: positions (1,1)

Links each field to the segment(s) in which it is used and the sequential position in which it appears.

Attributes of: V23 fields

description : DescriptiveText

The description of this field.

element : Integer

The unique numeric identifier assigned by HL7 for this field.

field_name : String

The name of the field in the HL7 Data Dictionary.

table : Integer

The number of the HL7 table that provides values for this field, if the field is table-based.

Class: V23 segments

Associated with: **Attribute**
 V23_field_segment

Description of: V23 segments

Contains all of the segments specified in HL7 Version 2.3, as captured in the data dictionary published by HL7.

Associations for: V23 segments

source_of (0,n) :: Attribute :: stems_from (0,n)

Many attributes are traced to equivalent content in HL7 Version 2.x. This connection is secondary to the path that traces an attribute to an HL7 field to a segment. It is provided for modelers who wish to specify particular segments for information model attributes.

contains (1,n) :: V23_field_segment :: is_in (1,1)

Aggregates the fields that make up each segment.

Attributes of: V23 segments

name : String

The name of the segment.

segment : String

The three-character segment identifier.

Class: Version MET

Subtype of: **Message_element_type**

Associated with: **Choice_branch**

Description of: **Version_MET**

Provides for version-specific METs for inter-version compatibility. When it is used, all of its branches will be included in the message.

Associations for: **Version_MET**

has_choices (1,n) :: Choice_branch :: is_choice_for (0,1)

Each branch of a Version_MET includes a type. All of these choices are used in a Choice MEI. A choice branch must be part of a version MET or a choice MET, but not a part of both.

Class: **Vocabulary_domain**

Supertype of: **Code_subsystem**
Code_system
Derivative_domain
Enumerated_domain

Is part of: **Domain_version**

Associated with: **Attribute_domain_constraint**
Coded_term
Derivative_domain
Enumerated_domain
HMD_domain_constraint
LOINC_link
MIM_attribute_domain_constraint
RMIM_attribute_domain_constraint
Vocabulary_domain
Vocabulary_domain
Vocabulary_domain
Vocabulary_domain

Description of: **Vocabulary_domain**

A vocabulary domain is a specification of the allowable values for an attribute or component of a composite data type that has a coded datatype assigned to it. The vocabulary domain may be as simple as a reference to a table of enumerated values or it can be a collection of predicate statements.

Every collection of terms is a vocabulary domain, including the one element set, the empty set, an enumeration of a few terms, and all huge vocabulary systems, their subsystems and derivatives.

Composition for: Vocabulary domain

in_version (1,1) :: Domain_version :: has (0,n)

Associations for: Vocabulary domain

is_constraint (0,n) :: Attribute_domain_constraint :: links_domain (1,1)

has_set_of (0,n) :: Coded_term :: element_of (1,1)

This relationship indicates that a vocabulary domain is a set of terms and every term belongs to one vocabulary domain.

Through subsystems and derived vocabulary domains, any term can be member of more than one vocabulary domain.

is_operand_for (0,n) :: Derivative_domain :: has_operands (2,n)

Relationship between a derivative, its operator, and the operands (other domains) to which the operator is applied.

includes (0,n) :: Enumerated_domain :: populates (1,1)

An enumeration includes terms from a particular domain. In the case where the enumeration is the entire domain, this relationship is self-referential.

is_constraint (0,n) :: HMD_domain_constraint :: links_domain (1,1)

equates_to (0,n) :: LOINC_link :: links_domain (0,1)

is_constraint (0,n) :: MIM_attribute_domain_constraint :: links_domain (1,1)

is_constraint (0,n) :: RMIM_attribute_domain_constraint :: links_domain (1,1)

referred_in (0,n) :: Vocabulary_domain :: refers_to (0,n)

This "see also" link is used in conjunction with the comments field to direct the interested reader to other codes..

refers_to (0,n) :: Vocabulary_domain :: referred_in (0,n)

This "see also" link is used in conjunction with the comments field to direct the interested reader to other codes..

updated_by (0,n) :: Vocabulary_domain :: updates (0,1)

This is used for versioning. Actually the version numbering is less important than the maintenance of these "updates" links between versions.

updates (0,1) :: Vocabulary_domain :: updated_by (0,n)

This is used for versioning. Actually the version numbering is less important than the maintenance of these "updates" links between versions.

Attributes of: **Vocabulary domain**

description : DescriptiveText

A textual description of the vocabulary domain and its purpose

edit_note : DescriptiveText

Editor's notes for the domain.

id : IdentifierString

A unique, sequentially assigned number that identifies a vocabulary domain.

internal_ind : CodedElement

Indicates whether the domain is one that is maintained internally by HL7 in the primitive vocabulary domain enumeration tables, or whether the domain refers to a set maintained by an external organization.

name : String

A unique textual name for the vocabulary domain. The name is created using mixed case object oriented style names, without the use of white space or special punctuation. The name is generally singular. This name is used when the vocabulary domain is referenced by other vocabulary domain definitions. Examples of acceptable names are: Gender, OrderType, PatientType, AbnormalFlag, etc.

The name may be determined by the organization defining the system which includes this domain.

realm_of_use : Enumerated

A coded element that indicates the country/affiliate/jurisdiction for which this domain specification applies. This term allows multiple jurisdiction-specific domains to be related under a single over-arching domain.

The values for the realms of use column come from the RealmOfUse vocabulary domain.

status : CodedElement

The status of the item. The values for Status come from vocabulary domain EditStatus. Some values for status are Proposed, Rejected, Active, Obsolete, and Inactive.

version : VersionNumber

The version of the domain as assigned by HL7 or by the organization defining the system of which this domain is a part.

version_out : String

The version number of the table at which this entry was deleted. A blank Vout value means that the row continues to exist in the current version of the table.

Data type definitions in: **HL7_V3_Meta-Model**

Data type: **Boolean : Boolean**

Is a Primitive Data Type

Description of: **Boolean**

Boolean data

Data type: **CodedElement : CodedElement**

Is a Primitive Data Type

Description of: **CodedElement**

Coded data

Data type: **CompoundHx : CompoundHx**

Is a Composite Data Type

Description of: **CompoundHx**

This set of components is assigned to one attribute of most meta-model elements. These components serve to track the history of each element and designate the models in which the element is valid.

Components of: **CompoundHx**

firstVer : String

This component contains the model unique identifier (modelID) of the first model version in which this element was defined.

hxID : Integer

This component is used to track the version history of each element of the model. It contains the unique element identifier assigned to each model element. The values are assigned in the repository. Modelers should never change these values or assign new ones, but they may copy them to indicate element history.

lastVer : String

This component contains the model unique identifier (modelID) of the model for which this element ceased to be valid. A blank lastVer value means that the element is valid in the most recent HL7 models. If this value is valued, the element is no longer a member of the current RIM. Since model identifiers are monotonically increasing, a given element is valid from the model identified by firstVer up to but not including the model identified by lastVer.

prevHxID : Integer

If an element of the meta-model derives from a previously defined element, this component will be valued. It contains the unique identifier of the element's predecessor,

Data type: **Date : Date**

Is a Primitive Data Type

Description of: **Date**

Date data.

Data type: **DateTime : DateTime**

Is a Primitive Data Type

Description of: **DateTime**

DateTime data.

Data type: **DescriptiveText : DescriptiveText**

Is a Primitive Data Type

Description of: **DescriptiveText**

In most instances the information to be kept about model components includes provision for a textual description of the component. Experience in documenting such models has shown the value of structuring these descriptions in order to provide for reference to external documents, identification of open issues, explanation of modeling rationale, etc. Therefore, descriptions of model components shall be of type DescriptiveText, as follows.

A paragraph that is part of the regular description shall not begin with one of the reserved phrases. Paragraphs that begin with a reserved phrase are used to capture comments about the rationale for modeling, to capture open issues and to express external references. The reserved phrases and their usage are shown below:

Reserved phrase "Rationale:" allows the modeler to document the rationale or justification for the specification of a particular element. It may occupy one or more paragraphs, but only one modeling rationale component should appear for any given model element. The first paragraph of the rationale must begin "Rationale:" The rationale will continue to the end of the description or until another reserved phrase is encountered.

Reserved phrase "OpenIssue:" allows the modeler to identify and discuss any open issues that remain to be resolved with respect to the model element. It may occupy one or more paragraphs, and there may be multiple open issues for a model element. The first paragraph of each open issue must begin with "OpenIssue:" The open issue will continue to the end of the description or until another reserved phrase is encountered.

Reserved phrase "ExtRef:" provides the specification of a reference to an external document, either by name or by a URL reference. Multiple external references may be contained in a given description. The external reference must be a single paragraph that starts with "ExtRef:" and must either be the final paragraph of the description or it must be followed by another reserved phrase paragraph.

Data type: **Enumerated : Enumerated**

Is a Primitive Data Type

Description of: **Enumerated**

Enumerated data

Data type: **IdentifierString : IdentifierString**

Is a Primitive Data Type

Description of: **IdentifierString**

Various data model elements in the use case model and interaction model are required to have identifiers that are unique throughout the model. These elements will be specified as being represented by an IdentifierString.

Elements that use these identifiers are: application role, interaction, message, scenario, scenario example, trigger event, and use case.

An IdentifierString is a string that contains no embedded spaces and that is built from a limited character set. The IdentifierString may include any number of the following characters: upper or lower case alphabetic characters (A-Z and a-z); the digits (0-9); the dot character (.); the hyphen character (-); and the underscore character (_). These characters may be in any order, except that the first character of the IdentifierString shall be either a digit or an upper case alphabetic character.

Note: Because the dot character (.) is an allowed member of an IdentifierString, the IdentifierString cannot be used in defining fully qualified names for elements.

Data type: **Integer : Integer**

Is a Primitive Data Type

Description of: **Integer**

Integer data.

Data type: **MultiplicityString : MultiplicityString**

Is a Primitive Data Type

Description of: **MultiplicityString**

A set of values and value ranges including the minimum and maximum occurrence are required for associations and aggregations in the meta-model. A MultiplicityString shall be used to represent this set in both the literary and graphical expressions of the model. The MultiplicityString is a constrained string. It is built according to the following rules:

1. A MultiplicityString shall have at least a minimum and a maximum value.
2. A MultiplicityString may also include an open ended range at the upper end.
3. A MultiplicityString shall be expressed either as the single element "1" (the numeral one) or as a pair of elements separated by an ellipsis (..).

4. The elements making up a MultiplicityString shall be either zero, a positive integer, or the character "*".

5. If the character "*" appears in a MultiplicityString, there must be only a single occurrence, and that occurrence shall represent the set of all positive integers that are greater than the largest of the other integers in the same MultiplicityString.

6. The minimum value for the multiplicity shall be the smallest integer in the MultiplicityString, and may not be the character "*".

7. The maximum value for the multiplicity shall be the largest integer in the MultiplicityString, and must be greater than zero.

8. The elements making up a MultiplicityString should be ordered in ascending order, but are not required to be.

Data type: NameString : NameString

Is a Primitive Data Type

Description of: NameString

The NameString is a string that contains no embedded spaces and that is built from a limited character set. The NameString may include any number of the following characters: upper or lower case alphabetic characters (A-Z and a-z); the digits (0-9); the hyphen (-), and the underscore character (_). These characters may be in any order, except that the first character of the NameString shall be an alphabetic character.

The appropriate use of upper and lower case characters, and the inclusion of special characters allow data modelers to create easily readable strings for the noun- and verb-phrases required for many of these elements. (Examples might include "is_ordered_by" or "HealthCarePractitioner" or NameString.)

For clarity of reading, the initial character of a NameString should be lower case when used for names of attributes, labels of relationships, and labels for state transitions, and should be upper case in all other uses.. No matter what conventions are used with respect to capitalization, when NameStrings are compared for uniqueness, all alphabetic characters shall be treated as though they are lower case.

Data type: String : String

Is a Primitive Data Type

Description of: String

String data.

Data type: VersionNumber : VersionNumber

Is a Primitive Data Type

Description of: VersionNumber

A version number is a string comprised solely of the digit characters and the dot (.) character.

Data type categories for: **HL7_V3_Meta-Model**

Data type category: **MET Metamodel data types**

Specifies particular data types used in the meta-model. Other data types such as String, Boolean, Integer and Enumerated are not listed here.

Contains data types: **Boolean**
CodedElement
CompoundHx
Date
DateTime
DescriptiveText
Enumerated
IdentifierString
Integer
MultiplicityString
NameString
String
VersionNumber