HL7 QUALITY ASSURANCE/DATA MODELING COMMITTEE

HL7 Version 3

Message Development Framework

Version 2.1
July, 1996

Table of Contents

Figures

Tables

1. Forward

This document is to be used by members of the Health Level 7 (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.

It is an understatement to say that the developers have drawn heavily on the work done by CEN TC251 WG 3[1]. In many ways, this manual is a re-statement of WG3's document Method for the Development of Healthcare Messages.

1.1 Acknowledgments

To be added.

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. It includes material and references to guide the message developer, and to help ensure the quality of the HL7 standard.

1.3 Introduction

This document leads a Technical Committee Project Team 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 Model[2] 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.

This methodology is not just a set of notations, but a systematic process for taking the message developer from original requirements to implementation.[3] The development life cycle is divided into three distinct phases: Requirements, Analysis, and Implementation. During each phase different aspects of the healthcare environment[4] 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[5]. Every Technical Committee will define and carry out projects to model messages for use in its sub-domain.

Each project must go through the phases defined in this document, but not all projects will be at the same phase at any given time. Each phase builds on the results of the previous phase. The process

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

[2]. 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.

[3] . 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.

[4] . Insofar as it is significant to HL7 messaging.

[5]. In HL7 Version 3, the first task of each Technical Committee is to

of developing messages using this methodology is presented here as a series of steps, but completing the phases usually involves 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 a detailed description of modeling methods, please consult the respective method textbooks.

This methodology defines the process of message development through generation of models and graphs during the following stages:

Table 1. Three Stages of the Message Development Methodology

Stage I Functional Analysis (Requirements)	Modeling Business Processes The Functional Analysis Stage seeks to provide a description of the domain's business processes through development of the Use Case Model The Use Case Model is built using use case diagrams to capture the project scope, and to allow full definition of the requirements that the set of messages are designed to support. This definition provides a basis for assuring the quality of the later stages of message development, and will lay the basis for defining the desired ways that applications using HL7 should interact.
Stage II Structural Analysis (Analysis)	The Structural Analysis Stage encompasses the construction of two models. The Information Model defines the the data that messages will carry, and the Interactions Model provides definition of the specific interactions (information flows) that are needed to support the functional requirements. The interactions identify the messages that will be needed, and allow specification of the interaction profiles that HL7 using systems can claim conformance to. Building the Information Model The Information Model is built using class diagrams. and state transition diagrams. The Information Model defines. Creation of the Information Model provides for consistent and clear definition of the contents of a message or a group of related messages. Use of the HL7 Reference Information Model provides for a more efficient message development process. Harmonization with the HL7 Reference Information Model will ensure consistency of definition among the contents of all the HL7 messages. Building the Interaction Model The Interaction Model is built using interaction diagrams to describe the specific interactions (information flows) needed to support the messaging requirements. The service roles that applications will use to satisfy the use case scenarios described in the Use Case Model are developed within this model. The interactions depicted in the Interaction diagram provide a template for the flow of messages that can flow between HL7 using applications. This will allow specification of the interaction profiles that applications using HL7 can claim conformance to.

Stage III Specification (Implementation)	Creating Hierarchical and Implementable Message Specifications During Specification Stage, the models created during the previous stages are translated into specifications suitable for implementation. This stage draws on the previous models, and supports the definition of the messages that HL7 communication partners will transmit. It supports precise definition of the allowable contents of the message(s) associated with each HL7 trigger event. It also provides a well defined "abstract specification" that allows creation of an implementable message specification for each Version 3 messaging syntax.

There are four types of models that are created or modified in order to create a message during these three stages[6]. This document organizes the message development process around those models, as indicated in the diagram below.

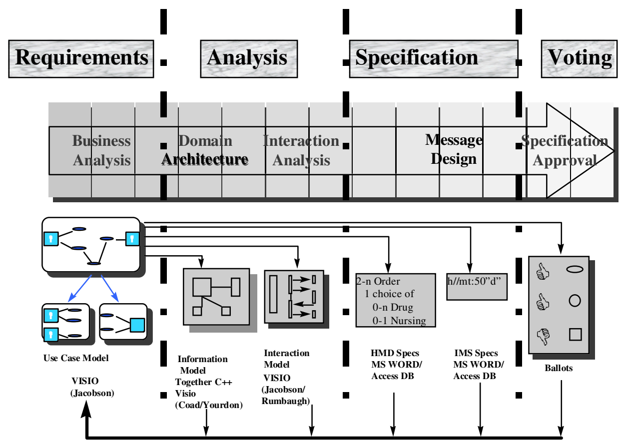


Figure 1. HL7 Message Development Project Model

---

[6] . The reader will note 3 stages and 4 models. This is because the order of development for the Information and Interaction models is not fully determined. HL7 message developers will probably iterate between development of these models, and different development projects will differ in which model they start with first. As a general rule, work should strart with the Information Model since it focuses attention on the data that will be exchanged in messages. QA/DM feels that, since the data tends to be the more stable aspect of the healthcare environment, it will be easier to reach consensus on its definition.

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.

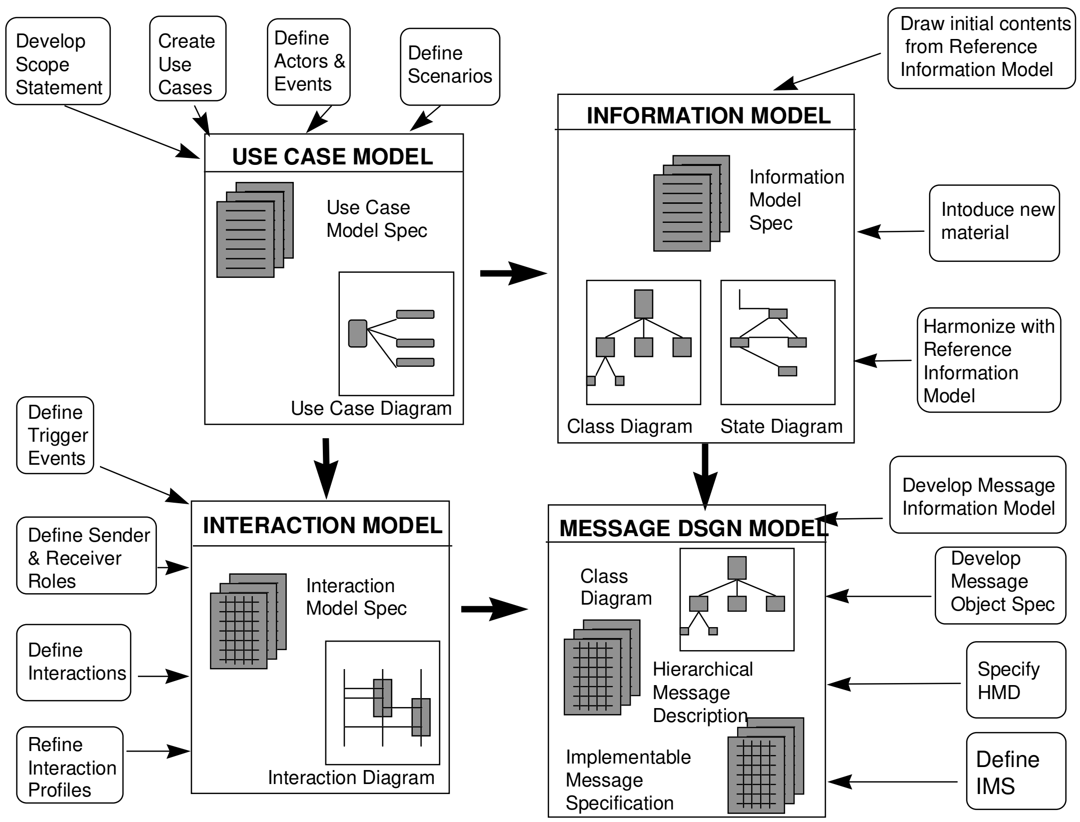


Figure 2. Project Life Cycle and its Models

The diagram depicts the steps the project must perform when designing a message. The models which are developed represent the collections of the work products of message development.

The process defined by this Message Development Framework provides for

- precise definition of functionality that will be provided in messages
- consistent data definition across the entire standard
- rigorous definition of the contents of each message that is defined.

1.4 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

1.5 Document Structure

This document guides the Technical Committee project team through the process of developing HL7 messages. Each step in the HL7 message development process results in the definition of a model. The step, and the resulting model(s) are described in more detail in the chapters that follow. Message developers should execute these steps in the proscribed order. However, the Quality Assurance/Data Modeling Committee does expect that message developers will need to iterate through previous steps to maintain consistency, and to fill in more detail as their level of understanding grows.

In the document, each step of the process is described in its own multi-part section. Each section contains the following elements:

Overview:

A description of the step, and its function within the message development process.

Procedures:

A list of the procedures to be followed to accomplish this step. This includes the inputs to each stage, a step by step recipe for each stage of the process, and the outputs that need to be produced..

Documentation:

A discussion of how the model contents should be represented.

Discussion and Helpful Hints:

Comments on how to accomplish the step, and/or discussion of the importance of different aspects of the job. This section is designed to provide assistance to members of the HL7 Working Group as they attempt to accomplish the procedures for each step.

Criteria:

Definition of different criteria for judging the quality of the work that has been done to accomplish the step. This section provides guidelines for the quality assurance process. It should be used during the process of message development to ensure quality, as well as after the fact.

The **toy model** is included at the end of the document to serve as an illustration of the different components of the message development process. Pointers to the relevant part of this model are included within the contents of each section.

2. Use Case Model

The purpose of functional analysis is to understand a problem before designing its solution. For HL7, this implies understanding the requirements for messaging and the Use Case Model documents these requirements. It documents the institutional, medical, and business practices that the message(s) being created will support. The CEN 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.”[7]

2.1 Overview

The Use Case Model lays the foundation for the other models, and it provides the functional content they draw upon. Jacobson's classical "use-case" approach is used to discover and analyze where HL7 messages will be needed. Based on the Use Case Model, candidate objects are identified, like Problem, Status, Person (who reports the problem) and so on, that will be identified and represented in the Information Model. In the Interactions Model, the list of the information flows that will need to be supported by HL7 messages is created by considering the actors in the Use Case Model as potential sender and receiver roles for the interactions that are needed to support use cases..

The reader should keep in mind, that a particular class diagram describing object relationships makes sense in a specific "use case" context. If analysts look at certain objects in isolation, there will be an unlimited possibility of describing them, because the purpose those objects are supposed to serve is not known.

The Use Case Model also provides a medium for communications with users of the standard. For Jacobson,

> “Another important characteristic of the requirements model is that we can discuss it with the users and find out their requirements and preferences. This 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 {*set of messages}*.”[8]

When the domain person performs a business function using a particular application, that application may need to communicate with other applications in order to accomplish a task or tasks required by that function. The contents of this communication are the messages that are being sent/received. These messages can be discovered using a method called Use Case modeling, and, within the healthcare environment, this is the area that HL7 is interested in. Use case modeling is used to describe the business processes which cause the messages to be exchanged between applications at different sites. Use cases view the business processes from the end user’s (a.k.a. domain expert) point of view.

---

7. MDHM, Page 25

8. OOSE, Page 129.
I have inserted {*set of messages}* in the message to replace the word “system” in the original. It is important to remember, that, when you read works such as Jacobson’s for background on how to develop HL7 messages, that the “system” discussed in the book is the set of messages we are developing and the way they are used, NOT the actual systems that use the messages.

In this document, the term “system” will refer to the set of messages and service profiles defined within HL7 Version 3.0 to support the communications needs of healthcare applications.

The figure shows steps in the development of the Use Case Model.[9]

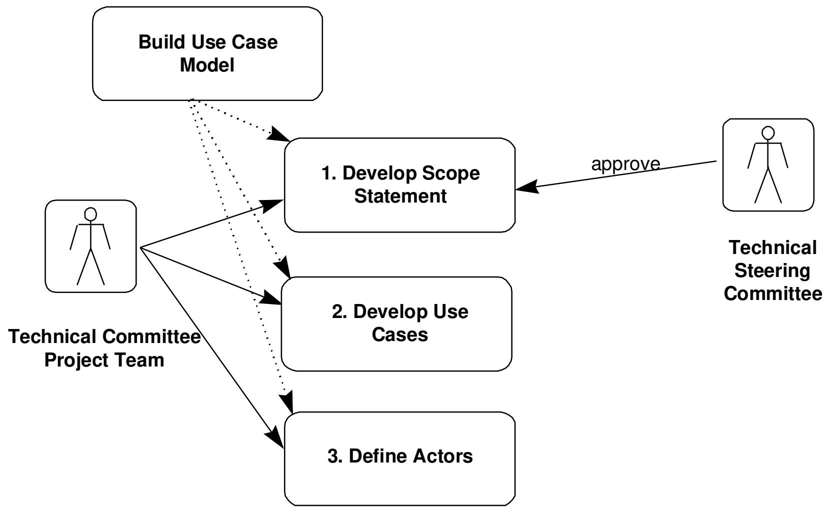


Figure 3 Use Case Modeling

The diagram shows the initiating role of the Technical Committee as the primary actor in development of the Use Case Model. The Technical Steering Committee must approve the Scope Statement as a means of guiding the development of the HL7 Standard.

The fundamental contents of the Use Case Model are provided by the text descriptions of the use cases. Use case diagrams provides a graphical representation of use cases, which is very helpful when developing and describing use cases. The use case diagram is a graph showing actors performing actions on the system.

- **UseCase**

  A use case defines a specific behavior or function that is important to the project team. A use case describe the possible sequences of interactions between the system and one or more actors in response to some initial stimulus by one of the actors. Each interaction starts with an initial event from the actor and proceeds through a series of events between the actor, the system, and possibly other actors, until the interaction initiated by the original event reaches its logical conclusion. When performing this task, the HL7 project team should look for use cases that allow a particular actor to carry out a task of measurable value, ensuring that use cases are neither too large nor too small.

- **Actors**

  Actors include both humans and other quasi-autonomous things, such as machines, computer tasks, and applications. More precisely, an actor is a communication role played by such an entity. Entities that are indirectly connected to the system are not actors, since their needs must be communicated to the system through an actor.

Use cases split the business into two parts: external users of the business and the internal functions, tasks, and processes that deliver value. Use cases are used to identify objects. An object may

---

[9]. The figure for the Use Case Model is an example of a use case diagram. Each Use Case describes a function that is carried out, and is depicted as a rounded rectangle, and its description is included within the rectangle. The party that caries out the function is depicted as a stick figure. The decomposition of one use case into many use cases is shown by the lines connecting the highlevel use case to its components.

participate in many use cases. With use cases, the analyst creates different models for each of these domains. The HL7 project team should start by identifying external objects and processes. These will lead to internal objects and processes.

2.2 Procedures

The Use Case Model captures requirements by describing the business processes from the domain expert's perspective. Follow these procedures in building the Use Case Model.

Inputs:

- A group of message developers who are ready to work,
- the HL7 Reference Information Model,
- existing models and documents produced by other technical committees.

Follow these steps:

1. **Develop Scope Statement:** Develop an initial problem description and write a Scope Statement that defines what is to be done. The scope statement is a high level use case that encompasses the entire project/domain. Review the scope statement with the HL7 Technical Steering Committee, and get approval by that body. The Technical Steering Committee shall ensure that the project/domain is relevant, feasible, and unique. Revise the scope statement as needed.

2. **Create Use Cases:** Work with domain specialists and existing standardization efforts to discover use cases of interest. The scope statement for the project should be considered as the first use case. Other use cases are developed by expanding and more fully defining the content of the scope statement. In defining a use case, group together all transactions that are "similar". Structure the use cases to bring out the functional relationships between the use cases. The identification of uses cases is often iterative. Once the picture 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 in detail, it is not unusual to uncover unclear issues in the expression of the scope statement. This allows it to be corrected at an early stage. Document each use case in sufficient detail to show the high level content of information flows, and prerequisites for information exchange. A use case is specified informally in words. Using the standard use case template, state the purpose of each use case, the initial event that starts it, and the termination condition.

3. **Identify Actors and Events:** Identify actors associated with the use case to provide the proper context. Verify that the actors that appear have been properly documented. For each actor, determine the fundamentally different and relevant activities that the actor can carry out. If there are too many, approach things at a higher level. Identify the initial event that starts each use case. This event is graphically represented by a line connecting the actor to a use case.

4. **Define Scenarios:** List a prototypical scenario that describes a typical example of the use case. If there are variations, list additional prototypical scenarios or describe the variations in words. Feel free to use ordinary language; a use case is not meant to be highly formal. Identify and describe all the exceptions that are logically associated with a given use case.

5. **Review Model:** Determine whether existing use cases are simple enough. The HL7 project team should ask itself this question: "Have we arrived at the trigger event we need to support

messages and interactions?" If not decompose them into their constituent use cases.[10] Verify that the documented set of use cases covers all the functional requirements for standardization within the project scope.

Outputs:

- Approved statement of project scope
- Use cases documented using the recommended graphical and literary expression provided.

2.3 Documentation

There is a simple diagrammatic technique for developing use cases[11]. Actors are represented by a stylized human figure, and the use case by an oval with its name inscribed within. Use this representation when developing use cases. This is especially valuable in group brain-storming sessions. The technical committee should also provide a written expression of the use case. This description generally contains 3 components: a) an action phrase that indicates the actor and the function the use case expresses, b) definition of the function being expressed, and c) further detail, as needed, to indicate dependencies of the function on other use cases or circumstances.[12]

Procedures for use cases have been documented by the JWG-CDM in the Data Modeling Framework. Sections of that document are included as an appendix to this document.

2.4 Discussion and Helpful Hints

The use case model lays the foundation for defining HL7 messages and their content. It is important to strike a balance between laying a firm foundation, and getting bogged down in the initial stages of message development.

2.4.1 Scope

Defining scope is the starting point for a project. It defines the area of healthcare functionality that needs to be supported by HL7 messaging. The scope will be used to confirm the need for standards development, and to ensure that the work is properly allocated within the HL7 Working Group.

Consider the need for information interchange when the HL7 Technical Committee develops the scope for a project. A new project should take up a current between a number of parties in healthcare.

Consider whether the scope is too broad. If the scope is too broad, the group will get bogged down, and have difficulty in developing messages in a reasonable time. On the other hand, taking an HL7wide perspective, too narrow a scope may lead to providing multiple solutions for a single problem.

---

[10]. One of the problems of modeling is knowing when to stop. When dealing with complex problems, there are often a great number of use cases that can produce an essentially infinite number of usage scenarios, especially with today's graphical user interfaces and event-driven systems. Too few use cases will result in an inadequate specification, while too many use cases will lead to functional decomposition and the scattering of objects and classes. Analysts must limit their analysis to the most obvious or important scenarios. The risks associated with use case modeling can be mitigated through knowledge, training, and avoiding an over enthusiastic usage.

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

[12]. Refer to the "toy model" included within this document for use case examples. Note, also how use cases are used within this Message Development Framework.

Determine whether the proper resources will be available. It is important to draw on functional expertise from a number of sources.

2.4.2 Use Cases

A use case is a textual description of specific situations within the healthcare environment in which communication between healthcare entities is needed. The use case describes the actors who are important within the project scope, and the activities that those actors carry out. Use cases are developed to fill out, and more fully define the functional requirements indicated by the Project Scope.

For TC 251, "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."[13]

Define use cases in terms of end users as opposed to information systems. The use case should provide a description of the necessary interactions from the perspective of the users of healthcare information systems.

Don't make the use case too complex. Remember that a use case can be decomposed into multiple related use cases. In some cases, when use cases are decomposed into more detailed use cases, a component can be "shared" by more than one high level use case. This is Jacobson's "uses" relationship. Remember that decomposing a complex use case into simpler ones is a good tool for reducing the complexity of a single use case. Re-using "component" use cases decreases the complexity of the use case model as a whole.

Remember that, for HL7, the "system" being defined is a group of messages that work together, not an end-user application. The use cases are used to identify trigger events for messages. Once use case development has reached the point of discovering trigger events, stop. Do not decompose the use cases any further.

Given that use cases are informal and do not carry strong semantics, the purpose of developing them is to force the analyst to define the requirements of the project at hand. Use cases are a way of specifying functionality in a manageable way, organized into categories of functionality from a user's point of view. By trying to enumerate the actors and the use cases, there is less chance of forgetting important messaging requirements.

2.4.3 Actors

An actor is a healthcare party which is involved in providing or receiving information within the context of the use case. This may be a person such as a healthcare practitioner, an organization, or a role played by a computer system such as order processsor, results reporter, etc. During the Structural Analysis stage, when building the Interaction Model, these actors will be reviewed to see if they have roles as senders or receivers of messages.

[13] MDHM, Page 15

A single actor can interact with the health care environment in fundamentally different ways. Each form of interaction with the health care environment system is called a *use case*. An actor initiates a use case. Actors can be primary or secondary. Primary actors are actors who initiate the primary activities in the use case. An example of a primary actor can be Caregiver who creates a lab order. Secondary actors are actors who participate in a use case which was initiated by the primary actor. An example of a secondary actor would be a lab technician who records the lab test results for an order that was initiated by a Caregiver, since the lab technician is not needed if there are no orders. It can be said, that secondary actors react to the use case that was initiated by the primary actor. Actors are used to identify trigger events. A trigger event is implied by a relationship (connection) between an actor and a use case. Since a single actor may participate in many use cases, it may be the initiator of several trigger events.

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.

To identify the actors, discuss the requirements from the actor's perspective. It will also be valuable to carry on discussions with those users who have experience functioning as actors, asking a number of questions, such as:

- What are the main tasks of each actor?
- Will the actor have to read/write/change any of the system information?
- Will the actor have to inform the system about outside changes?
- Does the actor wish to be informed about unexpected changes?

As noted above, 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. Initially, it is important to focus on those actors who initiate the activity within a 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. However, as the model is developed to the point of discovering trigger events, the focus shifts to the actors which are the immediate cause of information flows; these actors are application roles.

2.4.4 Scenario

A use case specifies a set of related usage *scenarios*, each of which captures a specific course of interactions that take place between one or more actors and the system. Therefore, the description of an individual use case typically can be divided into a basic course and zero or more alternative courses. There should be a finite number of use cases for a system, whereas there are usually an infinite number of possible scenarios (Naturally, only a finite number of scenarios should be documented.). In is important to document scenarios during the development of the use case model, since the scenarios provide a concrete illustration of the functionality that the use case describes. Multiple scenarios will serve to document the fact that a single use case can support different, but related, functions.

It is important to note that scenarios can be related to high level as well as low level or leaf level[14] use cases. In those cases where the scenario is related to a high level use cases, it will be important, at a later stage of message development, to align the scenario to a group of related interactions (For more information on this issue, see Interaction Model, Page 25.

2.5 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.

- Scope definition has been written down.
- Scope definition has been approved by HL7 Technical Steering Committee.
- Responsibility for the project has been assigned within the HL7 Working Group.
- Use cases do not fall beyond the defined scope. Conversely, the requirements for messaging that people expect from this project must be expressed as use cases.
- Use cases are clearly written, and express the need for communication, the information being communicated, and the parties involved.

[14]. If one thinks of all the use cases for a specific project as decomposed from the scope statement, they will appear as a tree structure- it can be upside down or right-side up depending on how it is visualized. In this analogy, those use cases that are not further decomposed are the "leaf level" use cases.

3. Information Model

3.1 Overview

HL7 messages support communications between applications in the context of a specific healthcare problem domain that supplies the underlying semantics used by the applications that exchange messages. The domain objects that are discovered as this model is built carry the semantics of a problem domain in terms meaningful to domain experts. Health care field has a terminology, and jargon words usually identify important concepts that are often domain objects. Domain objects are found and modeled without considering the needs of a particular application.

The Hl7 message development process involves several different information models. Each plays a role during the process, and all share the same modeling methodology. These models are:

- **Domain Information Model** expresses the information content for a single project. This model is developed by the technical committee during the Information Model stage of the process.

- **Reference Information Model** provides a coherent shared information model that contains all the data content relevant to HL7 messages. This model is maintained by the QA/DM Technical Committee, and grows through harmonization with domain information models supplied by the HL7 technical committees.

- **Message Information Model** contains the information content for one or more related messages. This model is developed by the technical committee from the Reference Information Model during the Message Design Model stage of the process.

The domain information model[15] provides a well expressed definition of the contents of messages. Its content is drawn from the relevant use cases. In the same way that development of the interaction model provides a more rigorous expression of the information exchange implied by the use cases, the information model provides a more rigorous expression of the data that is exchanged.

Each HL7 project will build a domain information model as part of the second stage of message development. However, the first step in doing so is to draw relevant material from the HL7 Reference Information Model. Whenever possible, development should draw on the work that has already been done, rather than starting all over again for each project. When the information model has been developed, any new or changed material should be harmonized with the HL7 Reference Information Model.

The Quality Assurance/Data Modeling Committee will support development of the Information Model through its work maintaining the Reference Information Model. QA/DM will help the Technical Committee identify relevant material in the Reference Information Model, and will manage the process of harmonization between the Domain Information Model and the Reference Information Model. The Harmonization process will involve identifying proposed changes to the Reference Information Model, determining the Technical Committees which would be affected by these changes, and driving the process of reaching agreement on new material for the Reference Information Model.

[15]. The terminology here sometimes causes problems. We use a CLASS diagram to model OBJECTS, and call this the INFORMATION MODEL.

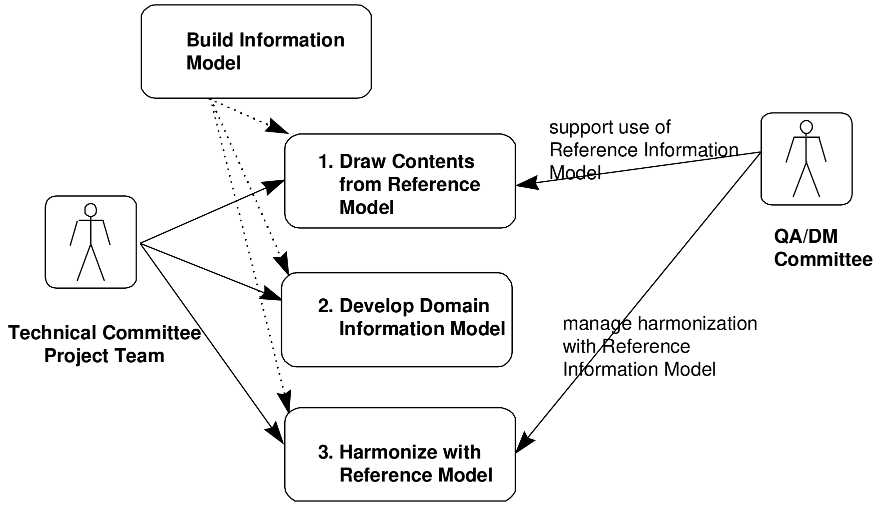


Figure 4 Information Model

The format of the information model is based on the Data Modeling Framework developed by the JWG-CDM. This provides a common language that can be used for comparing the data models developed by different SDOs.

3.2 Procedures

Inputs: Use cases to indicate the data content that is required.

I. **Draw initial contents from Reference Information Model.** Before going on to develop a class from the beginning, it is important to review the relevant sections of the HL7 Reference Information Model to see if this area has already been covered. Any classes that are needed, and already exist within the Reference Information Model should be drawn from there instead of being re-invented.

II. **Introduce new material**. This includes several steps, and is based on the identification and full description of classes. Review use cases to understand which classes will be needed in addition to those supplied by the Reference Information Model. Think of important concepts, terminology, and jargon in the problem domain. The entities that are important as groupings of information within the use case are candidate classes.

A. Provide clear textual definitions of each class.

B. Organize classes into a class model showing their relationships. Also discover Gen/Spec and Whole/Part structures. Determine which instance connections are required to support the meanings expressed within the use cases. Determine which classes are represented as Class & Objects, and which are simply Classes.

C. For each class, determine which attributes are needed to support the use cases. Provide names, definitions, and data types for the attributes.

D. Model object behavior. Build State Diagrams for classes that exhibit complex behavior. It may be helpful to build object's *life history model* showing the operations that take the object between fundamentally different life states.

III. **Harmonize with the HL7 Reference Information Model**. The HL7 Reference Information Model expresses the fact that full use of the standard requires consistency between parts of the standard. This consistency starts with use of common data definitions within the standard and extends to common ways of defining and representing information flows. Consolidating the model for a project with the Reference Information Model, guarantees that this consistency is maintained.

A. Identify the classes that appear in both models.

B. Examine each duplicated class. Are there differences in: definition: list of attributes, list of Gen/Spec, Whole/Part, and/or instance connection, instance connection description or cardinality?

C. Review each difference which is found to see whether the Reference Information Model should change after the consolidation. If the Reference Information Model will not be changed, this decision must be reviewed with the HL7 sub-group responsible for the project. If the Reference Information Model needs to be changed, this needs to be reviewed with the HL7 sub-groups responsible for models that contain the affected objects.

D. Review of changes or non-changes needs to follow the consensus principle and respect the fact that the functional content of the standard is developed by the message development work groups within HL7.

Outputs 3: Harmonized domain information model, updated HL7 Reference Information Model

3.3 Documentation

The procedures for documenting class diagrams and state transition diagrams have been documented by the JWG-CDM in the Data Modeling Framework. Sections of that document are included as an appendix to this document.

3.4 Discussion and helpful hints

The JWG Framework contains discussion on how to diagram and document data models. It also contains a discussion of state modeling. There are also many texts that provide opinions on this subject. This can be reviewed for guidance on the modeling process.

When the HL7 project team examines use cases to extract classes, it should look at the nouns that appear as subjects and objects within the use cases. Many of these will appear over and over, with accompanying descriptive information.

OO diagrams that show the model from a high-level perspective are the ideal tool for developing and providing an initial view of the model. When diagrams are created while building this model, it is important to organize them so each diagram doesn't have too many or too few classes.

3.4.1 Drawing from the HL7 Reference Information Model

The HL7 Reference Information Model provides a substantial foundation for efforts to extend the scope of the standard. The first step in developing the domain information model for a project is to draw relevant contents from the Reference Information Model.

Classes[16] within the model should represent significant "real world" persons, places, things, or concepts. The first step in model building should be to brainstorm to identify the basic subjects, e.g., persons, places, things, concepts, that, given the context of the use case model, are important. The modeling group should make sure that it spends time getting a list of classes or potential classes **before** it tries to develop an object model or even attempt to develop class definitions. The first step is to consider which subject areas within the Reference Information Model are relevant to the project. Then, the preliminary class list can be compared to the list of objects in the Reference Information Model.

3.4.2 Development of the domain information model

In many cases, people will want to start the modeling exercise at this point. This methodology, however requires development of the use case model first Following the methodology will allow better definition of the context for the class modeling in advance of actually diving in. Standards developers should consider, as a candidate class, any concept/thing about which information is communicated between systems. It is important for modeling groups to put substantial time and effort into naming and defining classes, before doing the detailed work of attribute, service, and object state definition.

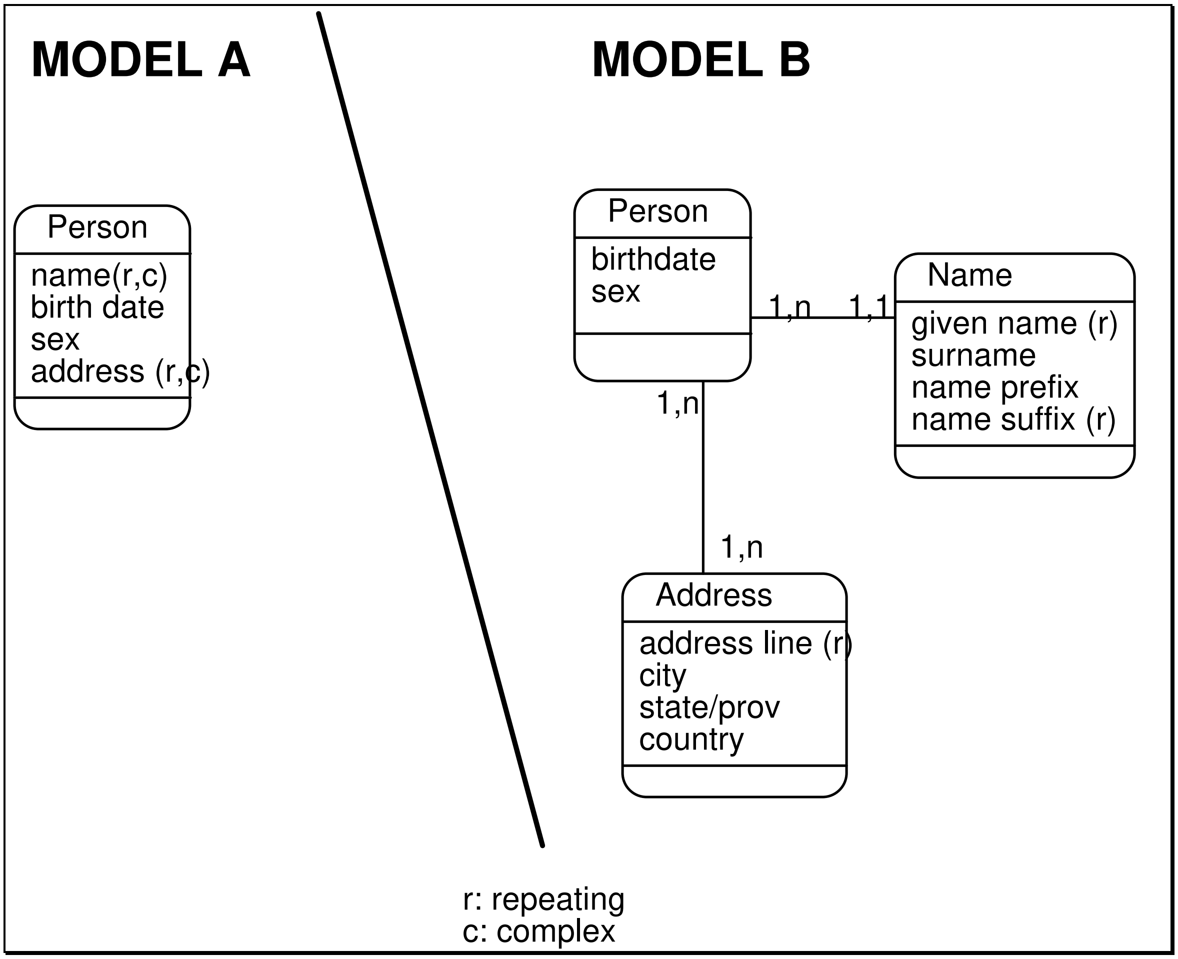
3.4.3 Classes

Classes are identified by examination of the use cases that have been developed. Those concepts which are important for the use case, and/or for which information is captured, are candidate classes. Therefore, the definition of a class must be drawn from functional rules and practices which are documented in the use case model.

3.4.3.1 Class or Attribute?

Initially candidate classes should be listed without worrying about whether they really should be modeled as classes or as attributes. However, once the process of defining and fully specifying a class starts, it is important to consider whether or not to define a given concept/thing as a class or as an attribute of a class. The conceptual framework provides the structures to define complex and/or repeating attributes so that many concepts/things can be represented as one or the other. For example, if the developer is modeling a person and age, sex, name(s), and address are important attributes to consider, the two models below, at least, are relevant:

16. Refer to Peter Coad & Ed Yourdon's Object Oriented Analysis for a thorough discussion on how to build object models. We do not attempt to replicate that material here. This clause will bring up points that Coad & Yourdon do not cover, or that are outside of the scope of their modeling paradigm.



Modelers need to consider the following issues to make this decision:

- Representing concepts/things as attributes hides some of the complexity of the model. On the one hand, it makes it easier to grasp the essential ideas being put forward; on the other hand it can hide important issues from the casual user. Today's HL7 seeks to hide complexity by allowing repeating and compound attributes. These attributes are assigned a datatype that defines their internal structure. This retains a precise definition of the data structure while not complicating the class structure and diagram. In general, this approach, as represented by the left hand side of the diagram is preferred.

- Objects within a class can have instance connections and object state histories. If it is important to show complex relationships between object/things than they have to be modeled as classes. For example, Model B above supports the idea that more than one person can share an address. In Model A, with address as an attribute of person, there is now way to show this.

Classes in the model do not have to represent something that is kept as a persistent object within system databases, although, those things that are stored are certainly candidates for being classes within the model. In general, it is important not to focus on the content of databases that people use or would like to use.

3.4.4 Class Contents & Structure

Once a preliminary set of classes and attributes is developed, it is important to check on the work to date to validate that each class has as simple a structure as possible, i.e., does not include attributes that should be assigned to other classes.[17] The objective is to produce a more robust model that corresponds more closely to the structure of the world that it represents. This will make the model less subject to change.

The refinement process parallels relational normalization and is designed to create classes that have desirable properties that simplify data access and representation. It is possible, and often desirable to create attributes which repeat and which have a complex internal structure. This is done by defining suitable datatypes, and the reasons for doing this are discussed above.

Therefore:

- While an attribute of a class may repeat, groups of attributes may not repeat together (excluding the trivial case in which all attributes can be said to repeat. Sometimes repeating groups are not self-evident from the way attributes are defined. For example, classes and attributes can be defined to capture a child's school experience.[18] The student class could include such attributes as "first grade teacher:, "second grade teacher", ... "twelfth grade teacher". In this case the repeating group "child's teacher" is rather thinly disguised by listing teachers according to the class they teach. It would be better to have a separate class "teacher" and create a relationship between it and the student. What happens if the model developer needs to capture the child's college and graduate education experience? If an attribute takes on only one value now, but might become a repeating group in the future, the cautious course of action is to pull it out and give it its own object. In the school system example, the designer could define the information for a classroom and include the type of computer students use as an attribute related to the classroom. However, if students gain access to multiple kinds of computer, this structure breaks down.

- All the attributes within the class must be functionally dependent on the unique attribute This makes sure that the attributes describe the class, not some other class inserted within the first. In the school example, it might be important to keep data on the colleges that twelfth graders apply to. If this information is within the twelfth grader object, and includes relevant data such as the college location and the average SAT score of new students, there is a violation of third normal form. It would be better to define a separate college class and relate it to the students applying to that college. If this isn't done, it is harder to define messages for the case in which college related data, such as the college location, is changed. If a twelfth grader decides to change the colleges s/he is applying to, much unneeded college information will have been entered.

To summarize. It is valid to define attributes that have a complex structure and that repeat. This makes it possible to hide some of the complexity inherent in the model from a casual view. Model developers must balance the contrasting perspectives of normalization and masking complexity. However, if model builders wish attributes to repeat or to otherwise violate normalization principles, this should be done explicitly in the attribute definition process.

3.4.5 Generalization & Specialization

The fundamental structure of a data model is expressed by Gen/Spec hierarchies. These show how classes can be related to or “types of:” each other.

Use inheritance properly! Improper use of inheritance tend to fall into three categories:

---

[17]. This exercise is similar to the normalization process carried on in relational database analysis, and has a similar objective. However classes are not relational tables and do not have the same requirements.

[18] . Throughout this framework document, the concepts introduced are amplified by developing an example based on a school or other setting. The document purposely does not use a healthcare related setting in order to focus the reader's attention on the ideas that the example is trying to show rather than the example itself.

1. "Has" relation with no "is" relation. Inheritance is a reflective relation. Test to see if the subclass "is" indeed a member of a superclass. The example of misguided use are: APPLE_PIE inheriting from APPLE and from PIE. The test of "is" APPLE_PIE a subtype of an APPLE clearly indicates the error in the logic. The intent her, obviously, was to use the inheritance to avoid repeating some common attributes. The correct inheritance would be that APPLE_PIE is a type of PIE and it contains APPLES, DOUGH, SUGAR, etc.

2. Taxomania. Avoid beginners enthusiasm. Do not use unnecessary intermediate classes in the inheritance hierarchy as a substitute for simple Boolean properties (such as gender class instead of a gender attribute). Introduce a new class only if it provides significant new or modified functionality in the form of query or command operations—features.

3. Convenience inheritance. Don't reuse the class simply to reuse its features, without the proper "is" relationship between the corresponding abstractions.

3.4.6 Instance Connections

The presence of instance connections and their cardinality can be inferred from the use cases and from group discussion within the project team. Instance connections are needed to express cardinality restrictions within the use cases, or to carry the semantic linking between two classes. The Information Model does have an integrity constraint that there must be a path of instance connections that can be traced between any two classes within the model.

Attributes and instance connections provide the detailed information that fleshes out the model. At this level, attributes and instance connections allow exact specification of what is meant by a class and of the constraints that are enforced across classes. Attribute values are typically constitute the data that is communicated between systems. Instance connections indicate much of the context for and meaning of those attribute values.

An instance connection "is a model of problem domain mapping(s) that one Object needs with other Objects, in order to fulfill its responsibilities".[19] You need to add instance connections to the model to express either:

- **integrity requirements**: The instance connection links objects to show that business rules require a specific cardinality relationship between the objects. E.g., A baseball model might include "team" and "stadium". The instance connection between them could show that a team cannot be defined within the model with out identification of one and only one stadium as the location where it plays its home games. Therefore, inter-systems communication of team information when a new team is created must include information about the home stadium.

- **scenario access requirements**: The instance connection links objects because scenarios specify that the attributes for one class can only be meaningfully processed in conjunction with the existence or non-existence of the other object. E.g., practices in the baseball world may require that, when transmitting the score of a game, the identity of the team whose home stadium the game is played in may be known.

A instance connection between two objects can always be expressed bi-directionally. In other words, the constraint expressed by the instance connection "Every baseball player is designated as playing one or more positions" implies the converse statement "Each position can be designated as being played by one or more players."

---

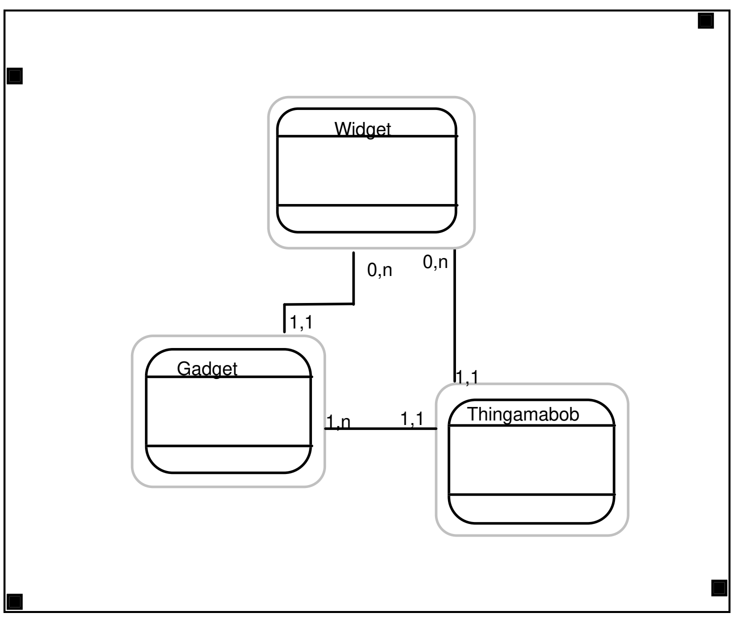
[19]. Object Oriented Analysis, Page 127

Instance connections in object diagrams show the range of allowable values on each end of the relationship, from zero to many. The possible cardinalities are listed below.

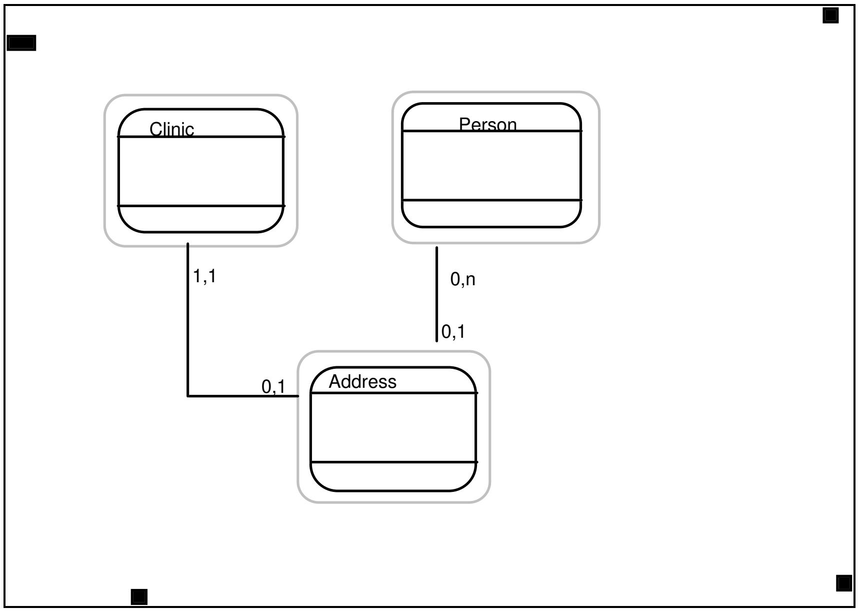
- **one to one**: E.g., A person has one and only one mother.
- **zero to many**: E.g., A person has anywhere from zero to many children.
- **one to many**: E.g., A book is written by one to many persons.
- **many to many**: E.g., An employer has many employees and an employee can work for more than one employer.

It is important to record cardinalities because they illuminate the fundamental relationships between the classes. Furthermore, when it is time to define communications between systems, cardinality between classes will often indicate how to model information flows.

When the model is being created, developers and reviewers should look out for "redundant relationships." For example, in the diagram below, a thingamabob is related to one and only one gadget, and a gadget is related to one and only one widget. Therefore, the instance connection between the widget and the thingamabob is redundant. It doesn't add any information. Object models should not contain such relationships because they can lead to inconsistencies in abstract messages or other communications instances. Note, however, that the redundancy here is based on the fact there is a mandatory instance connection between the thingamabob and the gadget. and between the gadget and the widget. If either of these were optional, then the widget-thingamabob instance connection would be needed.



In some cases, models will create instance connections with one-to-one cardinality between objects. Win this happens, the instance connection should be carefully reviewed. Usually, it means that a single object has been split into two classes. In some cases, the cardinality has been improperly defined and should really be 1:n or n:m. There may be situations where this is not the case. The most likely situation is that an object is "shared" with two others, and has 1:1 cardinality in its relationship with one of them. For example: The model may include classes for "person", "clinic", and "address". A person can have many addresses, a clinic can have only one, while the attributes of the person's address and the clinic's address are the same. In such a case the following model could result:



The implicit constraint that an address is related either to a clinic or to a person must also be imposed.

3.4.7 Attributes

An attribute is a discrete piece of information related to each occurrence of an object. Objects are assigned attribute values based on the information that needs to be communicated between systems. Since compound and repeating attributes are allowed, the attribute can take on many of the characteristics of an object. However, it does not have instance connections or services independent of the class it "belongs" to.

Modelers often distinguish between derived and primitive data. Primitive data is defined as the most basic data that is represented within the model. Primitive data cannot be decomposed any further within the model. On the other hand, derived data is "calculated" or "derived" from the primitive data. In general, the model should not contain derived attributes, but should be limited to primitive data[20]. This simplifies the model and the inter-system communication that is being supported. For example, a person's age is often important for healthcare applications, but this information can be derived from the current date and the person's birth date, both of which are in the model. It is better to allow calculation of such fields rather then include them in the model. One implication of this is that attributes should not be duplicated across the model. If an attribute value is needed for a use case along with the attribute values of a particular object, an instance connection should be created or relied on instead of duplicating the attribute in the object. For example, the lineup for a particular game will indicate the player's in the line up by giving the identifier of each player. If the name of the player is needed. this is done by documenting the relationship between the lineup for each game and the player's in the lineup, not by making player name an attribute of lineup.

3.4.7.1 Generic or Specific Attributes

Attributes can be represented in many different ways. A very common issue is whether to define generic or specific attributes for information about a class that may repeat. The case of a person’s identifiers is a good example. A person may have one or more of many different identifiers. They may have a social security number, driver’s license number, medical record number, DEA number, etc. The issue is: Should these be represented by many specific attributes, one for each identifier, or by a pair of attributes that capture the identifier type and value? The generic format is more compact

[20]. One way to handle derived data is to define the procedure by which the attribute value is calculated as a service of the relevant class.

and easier to extend - if a new identifier is discovered, then a new value needs to be added to the domain of the identifier type, as opposed to creating a new attribute. On the other hand, constraints related to the value of the attribute can be more difficult to express. However, Hl7 has decided to emphasize the importance of making it easier to express and extend the information model. Therefore, in general, it is preferable to use the generic format for attributes which can repeat in this way.

3.4.8 Datatypes

Each attribute in the model must be associated with a specific datatype. The datatype defines the rules for determining whether a specific value is valid or not for a particular attribute. This allows for precise definition of the contents of a particular attribute or data element; this precise definition is one of the basic features that allow standards to effectively support interfaces between applications. HL7 information models should use the rules developed by the JWG-CDM for describing datatypes.

The datatype structure can be used to represent composite and repeating data as is done in HL7 today. Person name, and address are good examples of the kind of data that can be represented with the appropriate datatype. "List" datatypes can also be constructed. This allows the representation of data elements which have coded values, and provides a place to record the valid codes for the data element.

Since the datatypes define the building blocks that model components are built from, datatype harmonization is a fundamental issue for standards convergence under the HISPP umbrella. In recognition of this fact, the MSDS Subcommittee on Common Data Types has drawn up a document, ANSI HISPP MSDS Common Data Types, which discusses some datatypes which apply to very commonly used attributes such as date/time and person name. Standards developers should refer to this document for guidance in defining datatypes. Whenever possible, the datatypes defined by the Subcommittee should be used.

3.4.9 States

The state model for a class describes the temporal evolution of an object of a given class in response to stimuli (events). State diagrams are formal specifications of the behavior of a class. In practice, most of the objects within the information do not undergo "interesting"[21] state changes, so only a minority of classes require their dynamic behavior to be modeled.

A state diagram is a directed graph of states (nodes) connected by transitions (directed arcs). A state diagram describes the life story of objects of a given class. It describes all possible ways in which the objects respond to events from other objects.

A state variable is an attribute that is valid while the object is in the state. State variables can be accessed and modified by operations within the state, including entry and exit operations. State variables are attributes of the object described by the state diagram.

All objects have states. Even the simplest object life cycle can be conceived in terms of state: it has one state, called "active." Creation into the active state, and deletion of an active instance, are the allowed state transitions in such a model. In many cases, in which the functionality being modeled needs to retain information over extended periods of time, there will be two object states: "active" (which allows creation), and "inactive" (which allows deletion and retrieval, but no services that connect other class instances to an inactive instance.) Defining this kind of state model supports scenarios in which an object can be retrieved, but not updated.

[21]. The objects drawn from every class can be created, destroyed (deleted), and updated. Coad and Yourdon have the notion of "algorithmically simple" state transitions for these state changes. For such classes, there is no need to document the state model. It is not interesting because it is already known.

Modelers care about an object's state if an instance of the object, on its life cycle from creation to deletion, passes through a period in which the transformations allowed on it differ from those allowed in another period of its life. (Such a period is called a "state.") Such transformations include:

- the attributes that are required/allowed/not allowed during the state
- the services that are required/allowed/not allowed during the state (update, delete)

A state transition is a response to an event received by an object which is in certain state. A transition may cause the object to change to another state. Transitions leading from the current state of an object are eligible to be triggered by an event received by the object.

3.4.10 Harmonization with the HL7 Reference Information Model

The final step in developing the information model is to harmonize its contents with the HL7 Reference Information Model. Once the project team is confident that the domain information model meets its requirements as defined in the use case model, it needs to be combined with the HL7 Reference Information Model. The domain information model can be compared with the Reference Information Model both in terms of form and content. The purpose is to combine the domain information model contents into the Reference Information Model so as to create an improved Reference Information Model.. The new Reference Information Model should include the contents of the old one and of the merged in domain information model. It should resolve inconsistencies and discard redundant classes, structures and instance connections. Another important output of this process is a harmonized domain information model which can be used both in further development, e.g., the interaction model and message model, and as a basis for later message development projects.

3.5 Criteria

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

- The project team must be able to generate the Information Model in the literary expression adopted by HL7. This can be done most easily using Together C++. However the requirement holds whatever the tool used to maintain the model, whether that tool is a CASE tool or a word processor.
- Classes and attributes should be defined in concrete terms.
- The important concepts from the use cases, and all the concepts that will need to be expressed in messages should appear within the data model as classes, instance connections, or attributes.
- The elements of the candidate model, whether they be classes, instance connections, or attributes should be incorporated into the Reference Information Model. They should either be added or shown to already exist in the model.
- If an element of the candidate model is not added to the Reference Information Model (which implies that it is not “needed” or functionally relevant), the appropriate changes need to be made to the use cases for the Reference Information Model
- Ensure that the model contains nothing that is not also in, or implied by (this refers to instance connection cardinalities) the Reference Information Model.

- Verify that the classes, attributes, instance connections, and instance connection cardinalities that are included can be justified by the use cases.

4. Interaction Model

4.1 Overview

The interaction model provides a way to specify the information flows that are needed to support the use cases that have been defined to support the scope statement. It provides names for these information flows or interactions. It defines the trigger event for the interaction, and the actions which interaction receivers should take when an interaction is received. It defines interaction profiles that aggregate communication roles (as sender and receiver) across multiple interactions. This information is needed to guide the development of message content and structure. It is also needed for defining and structuring HL7 conformance claims.

Interactions are discovered by examining use cases, especially leaf level use cases. They are defined when a use case requires, in at least some cases, that information flow from one actor to another. The actions that interactions must take, the “receiver responsibility”, is drawn from the requirements of the use case. If scenarios are developed, as examples or instances of use cases, the interactions can be drawn from the scenarios. When a scenario is based on a high level use case, multiple interactions may be needed to support it.

Trigger events are discovered within the use case and information models. Examination of leaf level use cases shows potential trigger events represented as the connection between the primary actor and the use case. It identifies the action that the actor is taking. Trigger events can also be discovered by examining the state transitions that are defined for classes within the information model. Each state transition should be traceable to one or more trigger events. (See “States” on Page 22)[22]

Communications roles are related to the actors involved in the use case. For each interaction the role of sender and of receiver is filled by an interaction profile which represents an interface to the computer based application or system that are end points to the information flows (interactions) that are needed to support the use cases in the Use Case Model.

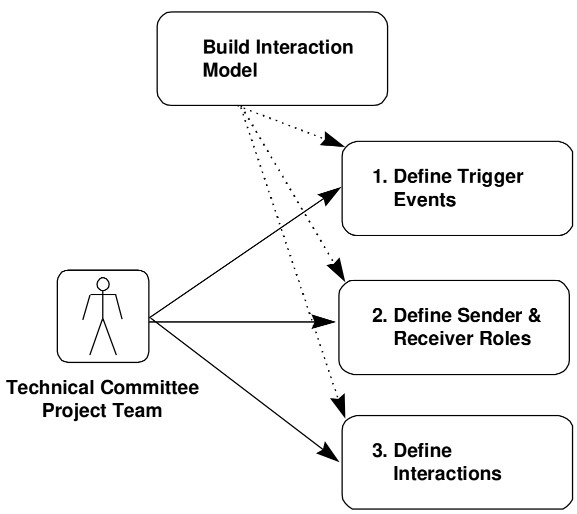


Figure 5 Interaction Model

22. Also note that every state transition should be traceable to one or more trigger events. In this context, it is important to remember that this methodology allows state transitions that do not change the state of an object, for example an update.

The Interaction Model employs interaction diagrams to more precisely define the communications needs that will be supported by HL7 messages. The set of interactions that flow from a single trigger event illustrate scenarios that implement a use case.

Scenarios are "threads" or sequences of events that occur between actors in order to support a high level use case. Each scenario includes actors, the use case which is being supported, and the sequence of interactions to carry it out. An interaction diagram is constructed to show a group of interactions for a single scenario. The interaction diagram is a graph that shows interaction profiles as vertical bars and arrows which represent the communication between these service profiles, i.e. interactions or messages. An interaction is drawn as a labeled arrow between interaction profiles.

4.1.1 Interaction Profile

Definition of the communication roles filled by an interaction profile provides the bases for conformance claims. The behavior mandated for an interaction profile is summarized from the sender and receiver roles defined for each interaction. Therefore, systems or applications which use HL7 can claim conformance to an information profile. The specific claim is that the system or application supports one or more HL7 interaction profiles. This implies the ability to send and or receive the interactions the profile is related to. As a result, it is important to ensure that the interaction profile indicates a functionally meaningful group of sender and receiver roles.

4.2 Procedures

Inputs: The lowest level (aka leaf level) use cases from the use case model, class states from the information model.

1. **Define trigger events**. Each "leaf" level use case for a project is a candidate trigger event. Consider, for each one, whether it is a trigger event. If it isn't, perhaps it should be removed from the model. Assign a name to the trigger event; this should be the same name as the one given the use case. In some cases, a trigger event is not valid for an application unless a specific interaction has been received, or an object that the interaction refers to is in the appropriate state. This needs to be defined.

2. **Determine sender and receiver roles**. Provide a label for the "kind" of computer application will send the message. It may be the case that there are different kinds of sender roles that handle a trigger event. This really means that, for a given trigger event, there will be a different interaction, i.e., different message contents, for each sender role . In this case, the model developers must list each sender role, and define a different interaction for each. For each trigger event and sender (if there are multiple senders for a trigger event), define which kind of application will receive the interaction that is sent.

3. **Define the interaction.** Given the trigger event and other information such as the sender and receiver system, give a name to 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.

4. **Refine interaction profiles**. The content of an interaction profile consists of all the communication roles - as sender and as receiver - that are defined for a specific named interaction profile. This definition 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 interaction profiles emerge..

Outputs:
Interaction grid that lists the required elements for each interaction, interaction profile table that shows,

for each application type designated with a sender or receiver role, which interactions it must be able to send and receive.

4.3 Documentation

Interaction flows and sender and receiver roles are defined to provide a more formal description of the use cases. In particular, this provides a clear description of the essence of the use cases from HL7' perspective. This set of models defines the specific instances of information exchange that will need to be supported by HL7 message descriptions.

The interaction diagram is usually developed for the interactions associated with a single scenario, which itself is drawn from a use case. Within the diagram, the interaction profiles that appear are shown by vertical bars, and the interactions are shown as arrows that go from the sending profile to the receiving one. Trigger events are listed on the left hand side, and are connected to the interaction they are associated with. When two interactions are linked because one must be sent by the receiver of the first, the interaction profile shows a rectangle or "block", that links the two. (Jacobson calls this an "activity".)

Use this table as a model for developing the interaction model[23].

Interaction ID				
Interaction Name				
Trigger Event Name				
Sender				
Event Dependency				
Receiver				
Receiver Responsibility				

The interaction profile table is even more simple. It is a table which lists the sender and receiver roles defined in the interaction grid, and which indicates, for each role (either sender or receiver) that appears in the interaction grid. the IDs of the interactions that it must either send or receive.

Sender/Receiver Role	Must Send	Must Receive

4.4 Discussion and helpful hints

The interaction model provides a place to determine and specify the information flows that the messages under development will support. Construction of this model draws on the use case model for its basic material. It also draws on the information model for confirmation that the trigger events needed are being defined. This model also lays the groundwork for defining meaningful conformance claims for HL7 users. Here are some points to help in developing the model.

4.4.1 Trigger Events

Defining trigger events is a key step within the HL7 process. Trigger events define the circumstances under which messages are sent, and are just as critical for allowing systems to co-operate as the contents of the messages themselves. Trigger events are discovered, in the first place through the use case models. Each use case is initiated by an actor, and that point of beginning which is shown

[23]. Refer to the Appendix of this document for a description of the implementation model's contents that is drawn from JWG-CDM Framework, Annex B.

on the use case diagrams as a connection between the actor and the use case is a potential trigger event. This is particularly true of the leaf level use cases.

In the second place, trigger events are discovered in the state behavior of classes in the information model. Each state transition defined for a class represents a perturbation of the class - a change in one or more attribute values. These changes are not spontaneous, they are caused by something, and that something is a trigger event. Therefore, inspection of the information model provides a source for discovering trigger events. Furthermore, when a preliminary list of trigger events has been achieved, this list should be compared to the state transitions of the information model to see if each trigger event can be associated with a trigger event. If not, either the model contains unjustified state transitions, or is missing trigger events. It is not necessary to guarantee consistency, because both models will be developing over time as HL7 grows. However, it is important to understand differences here.

In many cases, there will be just one interaction associated with a trigger event. This implies that the messaging for that trigger event is complete once the first interaction has been sent. In is important to note that this does not include the system response that indicates an interaction was received. That response is assumed for all interactions and does not have to be documented here.

In other cases, a trigger event can require multiple interactions. 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.

Normally a given sender role will appear for a number of trigger events. This reflects the fact that an application system will need to support multiple use cases to provide real value to its user.

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 so that each type can send a different interaction.

4.4.2 The Interaction Grid

The interaction grid provides a framework for documenting the entire interaction model. It is a table that includes a row for each interaction and contains columns for each of the interaction’s properties.

4.4.3 Interaction Profiles

An interaction profile defines the interface between systems and applications and the body of HL7 messages. It is a “type” that can send and/or receive a given interaction. The source for interaction profiles is the list of actors in the use case model, particularly the actors for the leaf level use cases. An interaction profile’s full definition consists of two lists: the interactions it can send, and the interactions it is expected to receive. This is shown in the profile table which is drawn from the interaction grid.

The profile table is constructed once the interaction grid is populated. Once this is done, it is important to consider the interaction profiles that have been defined to determine whether they are reasonable. In particular, whether the interaction profiles, as defined, provide a sound basis for conformance claims.

4.4.4 Interaction Diagrams

The interaction diagram provides a way to illustrate the relationship between interactions, and is a useful tool for developing this model. It is particularly helpful when working through a scenario based on a higher level trigger event to ensure that the relationship between multiple interactions has been properly worked out.[24]

4.4.5 Conformance Claims

A particular feature of HL7 Version 3 is that it lays the basis for reasonable conformance claims. The conformance claim is drawn from the definition of an interaction profile which offers a summary of the different sender and receiver roles that have been defined. In other words, the interaction grid defines those messages which an interaction profile is expected to be able to send, and to be able to receive. Ability to receive an interaction also includes the ability to carry out the receiver responsibility as defined for the interaction. This becomes a starting point for allowing users of HL7 to claim conformance to one or more HL7 interaction profiles.

4.5 Criteria

- Each use case must be represented within the interaction grid.

- The interaction grid must include the specified contents. It must contain sender roles, receiver roles, information flows and trigger events.

- The content of the interaction diagram should accurately express the content of the related scenario as documented for the use case the scenario is related to.

- The interaction profile table must include a row for each sender and receiver role that appears within the interaction grid. It must also include a cell entry for each time a sender and/or receiver role appears within the interaction grid.

[24]. See Jacobson and UML for a discussion of this construct.

5. Message Specification Model

5.1 Overview

In this stage the technical committee actually specifies the format of HL7 messages and relates them to interactions. To do so it goes through the steps illustrated below.

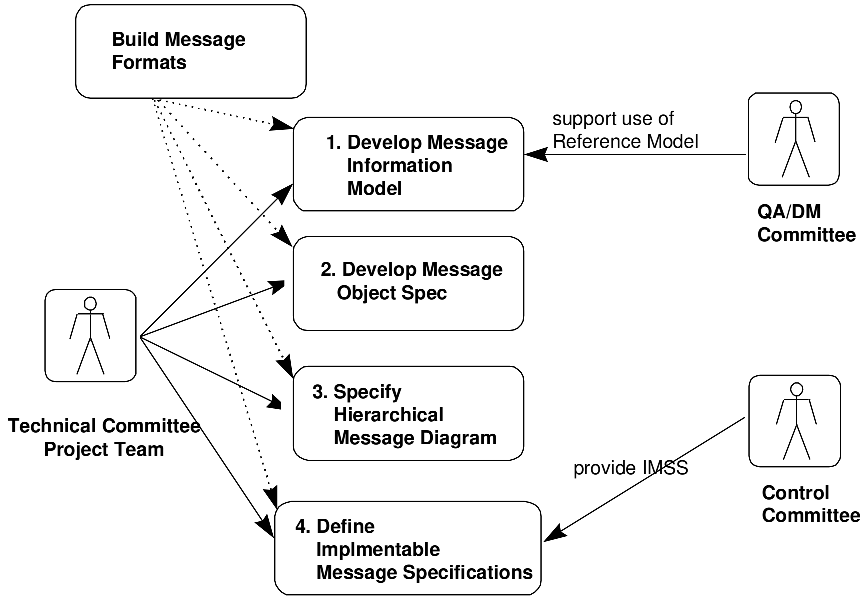


Figure 6 Message Description Model

1. It selects a subset of the Reference Information Model to serve as the information model of the specific domain of interest. This Message Information Model (MIM) is directly related to the Information Model produced in Stage II of the process.

2. It builds one or more Hierarchical Message Descriptions (HMD) to describe messages. Each HMD describes one or more messages. Each message may be used to describe the message format that is used for one or more interactions.

3. Specialists work with the Technical Committee to create sections in the Implementable Message Specifications (IMS) that correspond to the message formats described in the HMDs. There will be two Implementable Message Specifications, one for Printable Character Stream messages and one for messages transmitted using OLE and/or CORBA.

5.1.1 The Message Information Model

The Message Information Model (MIM) must be a subset of the Reference Information Model. Its purpose is to create a simplified representation of the contents of the Reference Information Model by selecting entities that are relevant to a logical group of messages. The MIM will be published with the messages to help users understand their structure and the definition of their data elements.

5.1.2 The Hierarchical Message Description

The Hierarchical Message Description is the primary definition of the contents of a message. In preparing it the technical committee

a) moves from the generality of class to the specificity of objects. If the same class appears several times in a message to represent different objects it is repeated in the description. For example, if a message describes a person who is a patient, another person who is next of kin and another person who is the referring physician, then the Person class may be repeated in the HMD once for each of those objects

b) describes for each object its optionality, its conditionality (conditions under which it may or may not be included in the message), and its cardinality (how many times it may recur in the message)

c) enumerates each attribute of the class that will appear as a data element in each object representing the class (we may send fewer and different attributes of Person when it is describing the referring physician than we do when it is describing the patient)

d) describes for each data element in the object its optionality, its conditionality (conditions under which it may or may not be included in the object within the message), and its cardinality (how many times it may recur in the object)

e) establishes an order in which the object views can be sequenced.

Because several objects in the message may represent substantially different sets of attributes of the same class, we call the representation of an object in the HMD an **object view**.

The HMD is represented as a table whose rows represent object views and data elements. A row that identifies an object view is labeled with the name of the class that it is drawn from A row representing a data element is labeled with the data element name and type.

Each column represents a message description--the complete listing of all the object views and data elements that may appear in the message. A column heading contains one or more Interaction IDs (from the Interaction Model) that are served by messages that have a common format.

The cell at the intersection of an object view row and a column describes its connection to at most one other object view and its optionality, cardinality, and conditionality. The cell at the intersection of a data element row and a column describes its optionality, cardinality, conditionality and the requirement for specific values.

5.1.3 Implementable Message Specification

The final step in message development is creation of the implementable message specification (IMS). This defines the rules for creating the actual message that will pass between systems using HL7. In order to create the IMS, you need to draw on the message design model which defines the rules for creating messages in a particular syntax supported by HL7. There may be multiple message design models, one for each messaging syntax that HL7 will support, e.g., ER7, OLE, CORBA. For each message design model, a different IMS can be created. This final stage of message construction will be highly mechanical. We may be able to automate all or part of it.

5.1.4 Optionality, Cardinality, Conditionality, and Required Value

These terms are used in the following way in Hierarchical Message Definition. This varies slightly from their usage in the information model.

5.1.4.1 Applied to Object Views

The *optionality* of an object view may be:

- **Required**. The object view must appear every time the message description is used for an Interaction (subject to the optionality implied by cardinality and conditionality, below.)

- **Site Specific.** The object view may appear when the message is used. This will be defined on a site-specific bases. See discussion on HL7 Conformance Claims *(needs to be written.)*

- **Not Permitted.** This object view may never occur when the message description is used for an interaction. (This option exists because the object view represented by the row of the HMD may be used in other message descriptions described by other columns of the same HMD.)

The *cardinality* of an object view refers to the number of times it may repeat in the message. It is specified as a pair of numbers specifying the lower bound and the upper bound, as in (1,m). The left number is the lower bound. It may be zero, one, or any other positive integer. When it is zero, the object view may be optional. The right number is the upper bound. It may be one, any other positive integer, or lower case "m." Lower case "m" indicates an unstated arbitrary number of occurrences.

The *Conditionality* of an object view may be

- **Unstated**. The Object View may or may not appear according to the optionality and cardinality rules.

- **A stated predicate**. The object view may not appear in the message unless the predicate is true. If the predicate is true it must, may, or may not appear in the message according to the Optionality and Cardinality stated above. The predicate is of the form "Object View-Data Element" = <list of values>, where the specified data element is an enumerated data type and the values listed are valid for the data type. If the data element that is the named in the predicate is not present in the message the predicate is false.

It is not a valid construct to include a stated predicate with an optionality of Not Permitted.

The *required value* of an object view adapts the message format to specific Interactions. It requires that an optional object view be sent and further specifies the value of an enumerated data element in the object view. A required value is always stated in the row corresponding to a data element. Its usage is described below, under "Applied to Data Elements."

5.1.4.2 Applied to Data Elements

The data elements that are specified as part of an object view may only appear in the message if the corresponding object view appears. When a value appears for a data element it may have a value, it may have a special value called Null, or it may be "not present." When the object view does appear, the optionality, cardinality, and conditionality of each attribute may be further specified.

The *optionality* of a data element within an object view may be:

- **Mandatory**. The data element must be present every time the object view appears (subject to the optionality implied by cardinality and conditionality, below.) It must not have a null value and it must be present.

- **Required**. A position for data element must be present every time the object view appears. It must be present (subject to the optionality implied by cardinality and conditionality, below) but it may have a null value.

- **Site Specific**. A position for data element must be present every time the object view appears. It may be present (consistent with the optionality implied by cardinality and conditionality, below.) Whether is present will be defined on a site-specific basis. If present, it may have a null value. See discussion on HL7 Conformance Claims *(needs to be written.)*

- **Not Permitted**. A position for data element must be present every time the object view appears. A value may never be present (This option exists because the object view represented by the row of the HMD may be used in other message descriptions described by other columns of the same HMD.) If the data element that is the named in the predicate is not present in the message the predicate is false.

The *cardinality* of an data element refers to the number of times it may repeat within an individual occurrence of an object view. It is specified as a pair of numbers specifying the lower bound and the upper bound, as in (1,m). The left number is the lower bound. It may be zero, one, or any other positive integer. When it is zero, the data element may be not present in an occurrence of the object view. The right number is the upper bound. It may be one, any other positive integer, or lower case "m." Lower case "m" indicates an unstated arbitrary number of occurrences.

A valid HMD may not have any intersection of data element row and column in which the optionality of the data element is Mandatory and the lower bound of the cardinality is zero.

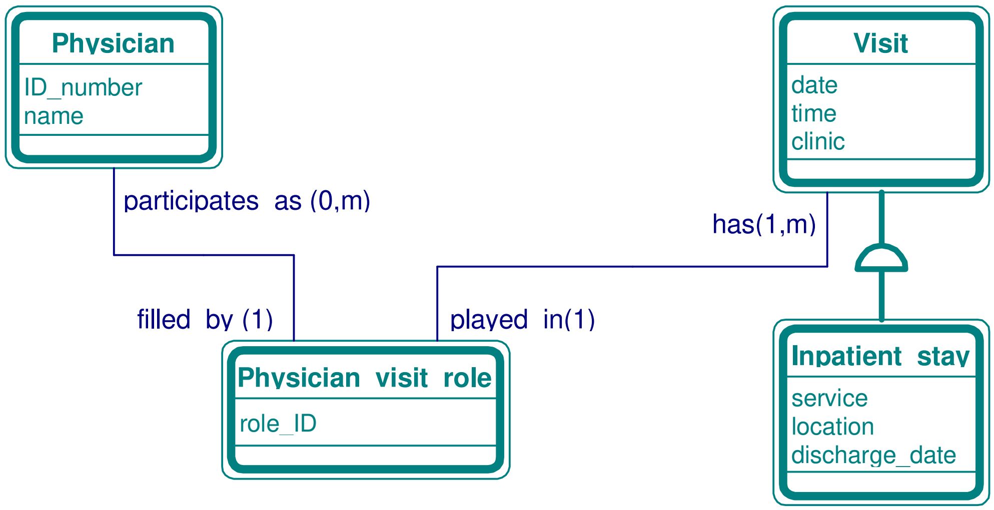
The *Conditionality* of a data element may be

- **Unstated**. The data element may or may not be present in the object view according to the optionality and cardinality rules.

- A **stated predicate.** The data element may not appear in the object view unless the predicate is true. If the predicate is true it must, may, or may not be present in an instance of the object view according to the Optionality and Cardinality stated above. The predicate is of the form "Object View-Data Element" = <list of values>, where the specified data element is an enumerated data type and the values listed are valid for the data type.

It is not a valid construct to include a stated predicate with an optionality of Not Permitted.

The *required value* of an object view adapts the message format to specific Interactions. It requires that an optional object view be sent and further specifies the value of an enumerated data element in the object view.



Consider, for example, the above fragment of an imaginary MIM. Further suppose that role_ID in Physician_visit_role were of an enumerated data type that specifies the role with values for Attending, Admitting, Referring, Senior House Staff, etc. The HMD might include one column for a message that will be associated with the Admit event. That column might include the notation that a value of Attending is required for one occurrence of the Physician_visit_role class. This is equivalent to saying that there must be an attending physician in the Admit message.

Required values for object views are noted at the cell that is the intersection of a column representing a group of Interaction_ids and a row representing a data element within the object view. It contains a value which is one of the enumerated values of the data element.

The inclusion of a required value for an object view implies that the entire object view is required. If the cardinality of the object view connection permits multiple occurrences of the object view, then other occurrences may be in the message, unconstrained as to the value of the specific data element. Multiple required values may be included if multiple occurrences the object view are permitted. Each occurrence of the object view must have one of the stated values.

{It might be better to replace this whole notion with a more general one of business rules--Wes.}

5.2 Creating the Message Information Model

5.2.1 Overview

The Message Information Model (MIM) is created by selecting elements of the Reference Information Model that are sufficient to describe all the information in a logical group of messages. Using a subset of the entire Reference Information Model will clarify the information relationships in the messages. Using a single model for the group of messages will decrease the total effort in creating the standard and help its users understand the information relationships among the messages.

Every class, attribute, and connection that appears in the MIM must also appear in the Reference Information Model. However, the MIM will exclude some classes from the Reference Information Model. Where it includes a class, it may leave out some of its attributes. Where it includes two classes that have a connection, it may exclude the connection.

Where the MIM includes a connection from the Reference Information Model it may adjust the cardinality in such a way to reduce the optionality. Specifically

(1,m) can be reduced to (1,1)
(0,m) can be reduced to (0,1)

(0,m) can be reduced to (1,m)
(0,1) can be reduced to (1,1).

However, what was required in the Reference Information Model cannot be made optional in the MIM.

(1,m) can *not* be reduced to (0,m)
(1,1) can *not* be reduced to (0,1).

Where the MIM includes an attribute from the Reference Information Model it may be required in the MIM even though it is optional in the Reference Information Model. However, if it is required in the Reference Information Model it must be required in the MIM. When an attribute has a cardinality greater than one in the Reference Information Model it may be reduced in the MIM.

5.2.2 Procedures

The domain Information Model that was created in stage II reflects an understanding of the information that must be sent that was derived from prior analysis of use cases and enhanced through contemporaneous creation of the Interaction Model. It is a primary source of candidate Classes, Attributes and Connections to include in the MIM. However, the scope of the domain Information Model and that of the MIM may be different; several MIMs may document specific groups of messages that are related to a single domain Information Model.

5.2.3 Documentation

The MIM is documented according to the same conventions as those used to describe the Reference Information Model.

5.2.4 Discussion and helpful hints

There are no exact rules for selecting the contents of an MIM. One strategy is to look at the "largest" message in a group. This would be the Interaction that, according to the use cases, seems to have references to a large number of classes. Technical Committee members may then review other, related Interactions, adding classes as dictated by their use cases.

There is little benefit to making a careful analysis in order to weed out a few data elements from a class. Work can progress faster by allowing them to remain in the MIM and selecting the relevant attributes later, when building the HMD. If, on the other hand, a class with many attributes is included for an incidental reason, requiring only a few of attributes, the Technical Committee may choose to eliminate the unused attributes from the MIM.

There are two benefits to excluding classes and connections from the MIM: (a) it will make the MIM easier to understand for readers of the standard, and (b) various busywork will be avoided if the graphical expression of the MIM fits easily on a page. The cost of excluding information is only that there may be a requirement to add it later, if the committee discovers a need while building the HMD. Since the MIM and HMD are developed during the same stage, it is convenient to make such iterative adjustments in their contents.

Likewise, there are no exact rules for deciding when to use a single MIM and when to have more. Splitting them will make smaller, more manageable graphical and literary expressions. Combining them will help the user understand the information relationships among message descriptions more clearly, and will receive some parsimony in the total length of the specification.

5.2.5 Criteria

5.3 Creating the Hierarchical Message Description

5.3.1 Overview

The HMD is represented as a table whose rows represent object views and data elements. A row that identifies an object view is labeled with the name of the class that it is drawn from and a suffix to distinguish it from other object views drawn from the same class. A row representing a data element is labeled with the data element name and type.

Each column represents a message description--the complete listing of all the object views and data elements that may appear. A column heading contains one or more Interaction IDs (from the Interaction Model) that are served by messages that have a common format.

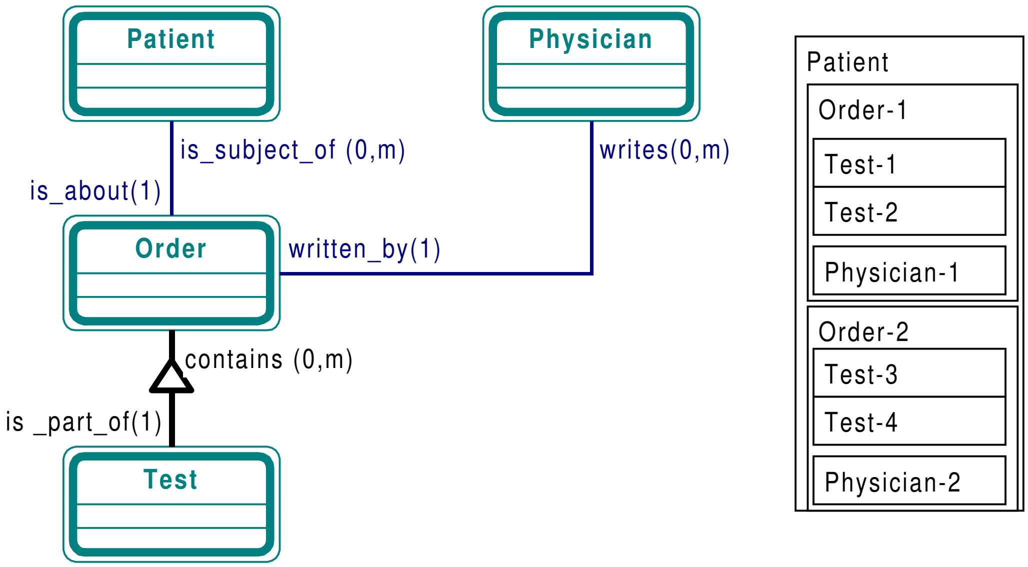
The cell at the intersection of an object view row and a column describes its connection to at most one other object view and its optionality, cardinality, and conditionality. The cell at the intersection of a data element row and a column describes its optionality, cardinality, and conditionality and required use.

In preparing the Hierarchical Message Description we will answer these important questions:

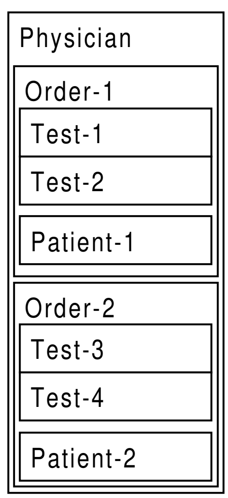
- What object views will be sent in a message?
- How are they related to the other object views in the message (by what connection?)
- When will they be sent (always, or are they optional, do they repeat)?
- What attributes of the classes of the MIM from which object views are derived will be sent as data elements for each class?
- In what order will the object views be sent?

The Hierarchical Message Description is a hierarchy in order to portray both sequential relationships among the object views and subordinate relationships. The sequential aspect portrays the order and the subordinate relationship defines the relationship among the object views (as in this is the patient who is the subject of this visit.) In simple messages the relationships may be self-evident. However in most messages the relationships must be clearly shown (as in this is the Person who is the Patient, this is the Person who is the Guarantor, this is the Person who is the Physician who gives primary care to the Patient while that is the Person who is the Physician who will be the Attending physician for this Visit.)

The relationships among object views are most frequently represented in the message by grouping the subordinate objects “within the scope” of the superior object. The physical representation of ‘within the scope’ will vary in the different Implementable Message Specifications, but the logical concept is the same.



The figure above shows an imaginary MIM fragment and one logical interpretation of the subordinate relationships. (Physician-1 is shown “within” Order-1 simply to denote the association between them in a manner consistent with the implementation techniques of the IMSs.) However, the figure below represents another logical interpretation of the subordinate relationships that may be appropriate for a different Interaction.



At the end of the process the HMD, through its subordinate and sequential relationships will be a specific interpretation of the MIM as appropriate for one or more Interactions.

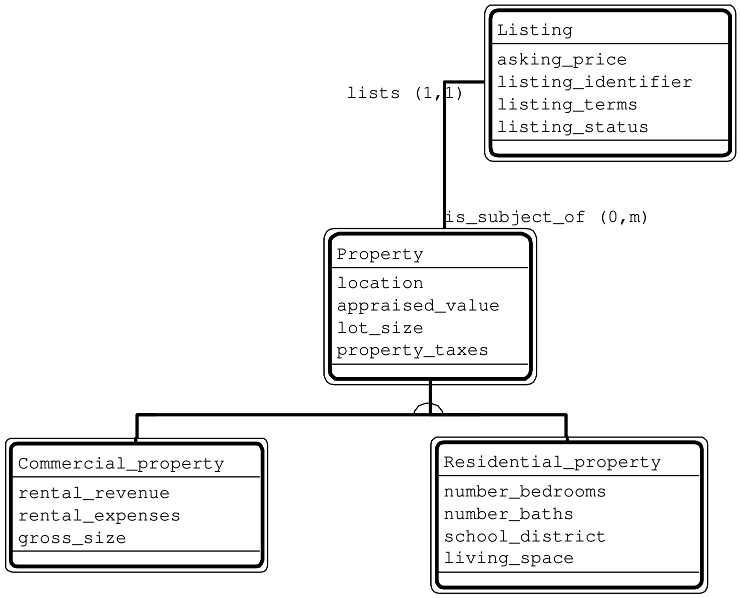
For convenience these “within the scope” relationships will be shown by indentation in Hierarchical Message Description.

So one interpretation of the MIM fragment above might be shown as:

		Patient
is_subject_of	(0,m)	Order
contains	(0,m)	Test
written_by	(1)	Physician

The indentation shows the reader that the written_by relationship is between Physician and Order rather than Test or Patient.

Another issue arises in interpretation of an information model when a connection is shown to a class that is has a gen-spec relationship with another class. Consider the imaginary MIM fragment drawn from the toy model.



Different instance of messages may be describing a Commercial_property, a Residential_property or a Property (not otherwise specified.) These alternatives must be described in the Hierarchical Message Description. This will be done by using the special construct **choice of**.

		Listing
lists	1	**choice of**
		Property
		Commercial_property
		Residential_property

The HMD is produced in three steps:

1. Create a Table of Object Views
2. Create an Adjusted Table of Object Views
3. Create the Hierarchical Message Diagram.

The Table of Object Views shows only the object views (without the data elements.) It is a first draft of the order in which the Object Views will appear in the final message and it has subordinate relationships to describe how the relationships in the MIM should be interpreted for a specific message.

The Table of Object Views can be represented as a table or in a graphical manner.

The Adjusted Table of Object Views is merely a re-ordering of the “first draft” Table of Object Views to optimize certain aspects of the message.

The Hierarchical Message Description is a recapitulation of the Adjusted Table of Object Views with the addition of the data elements.

5.3.2 Procedures

5.3.3 Documentation

5.3.4 Discussion and helpful hints

- Ensure that the attributes from each class in the subset model are represented. Check repetition against the cardinalities in the subset model.

- Pay special attention to situations in which the group of attributes drawn from a class appear multiple times within the table. This can happen when there are multiple instance connections between a class and other classes. Consider whether the multiple connections represent multiple objects (class instances) or repeated uses of the same object. In the second case, you should push these attributes to the top of the table and use a reference number to point to them.

- Data elements need to have the following information recorded for each message profile: xxxx

- You should note that multiple implementable Message Specifications (IMS) can be created for each message. There can be one for each message syntax that HL7 supports. Creation of an IMS is a strictly mechanical process based on the message contents and the message construction rules for the syntax being used.

5.3.5 Criteria

- All attributes from the message model are included as data elements.

- There is a message profile column for each information flow in an Event/Role diagram, unless the project team has clearly labeled two flows as identical.

- Element optionality is recorded for each element and message profile.

- Repeating objects within messages are handled correctly.

5.4 Creating the Implementable Message Specification for Printable Character Stream Messages

5.4.1 Overview

5.4.2 Procedures

5.4.3 Documentation

5.4.4 Discussion and helpful hints

5.4.5 Criteria

5.5 Creating the Implementable Message Specification for OLE and CORBA Message

5.5.1 Overview

5.5.2 Procedures

5.5.3 Documentation

5.5.4 Discussion and helpful hints

5.5.5 Criteria

6. Toy Model

The Toy Model[25] has been constructed to server as a serious extended example of the models defined with this Message Design Framework. It should be used as a tool for understanding the concepts introduced in this document, and as a guide for the actual process of HL7 message development.

6.1 Use Case Model

This section of the Toy Model includes:

- Model description and scope statement
- Use case model diagrams
- Actors in the use case model
- Use case descriptions. Individual use case diagrams are included for two of the use cases.

6.1.1 Model Description: HL7_Real_Estate_Demonstration_Model

6.1.1.1.1.1.1 Identifications:

Organization: Health_Level_7

Version: V 30.05 19960726

ModelID: HL7_TOY_3005

Developed by: QA/DM & Control Committees

Description of: **HL7_Real_Estate_Demonstration_Model**

MODEL SCOPE

The scope for this model is a sub-set of the residential real estate industry. In particular, it deals with those activities involved in: listing properties for sale with a multiple listing service, making offers against listed properties, making counter-offers, and accepting offers or counter-offers. Thus the principle actors are the buyers, sellers and real estate agents who represent them.

The model does not deal with activities surrounding the financing of real estate transactions or with final contract closure. These are the responsibility of a different chapter.

BACKGROUND

This model is being developed to serve as a vehicle for demonstrating and teaching the HL7 Version 3 standards-development processes as embodied in the Message Development Framework.

The domain of the model is outside of healthcare domain. The model is intended to be sensible, but is probably neither correct nor rigorous. Many elements of the model were included solely to allow demonstration of a particular modeling or transformation

[25] This model is Copyright 1996 by Health_Level_7

practice. By selecting non-healthcare subject matter we hope to focus on process rather than the content for the models.

The HL7 members involved in this effort offer our sincere apologies to individuals in the real estate industry for the liberties we have taken in modeling their domain. To us, this model is a toy to be played with. We trust that you have played doctor or nurse somewhere in your past.

Model files: QADC305*.* - The copyright to the data model contained and expressed in these files is held by HL7. Use of the model and expressions is restricted to activities that support the development of HL7 standards.

This model is being developed by members of the QA/DM.

This is the fourth update to the model. It includes changes to meet the needs of presenting the Model Development Framework during the August 1996 HL7 meeting.

The various graphical forms of the model include:

INFOMODL.WMF (.GRA) - The layout of the Classes in the Information model.

6.1.2 Model Diagrams

Two diagrams are included. The first shows the high level use cases, and the second shows all the use cases, and indicates their decomposition from the high level use cases. This second diagram may not be useful for HL7 development, due to the complex use case models that may be developed.

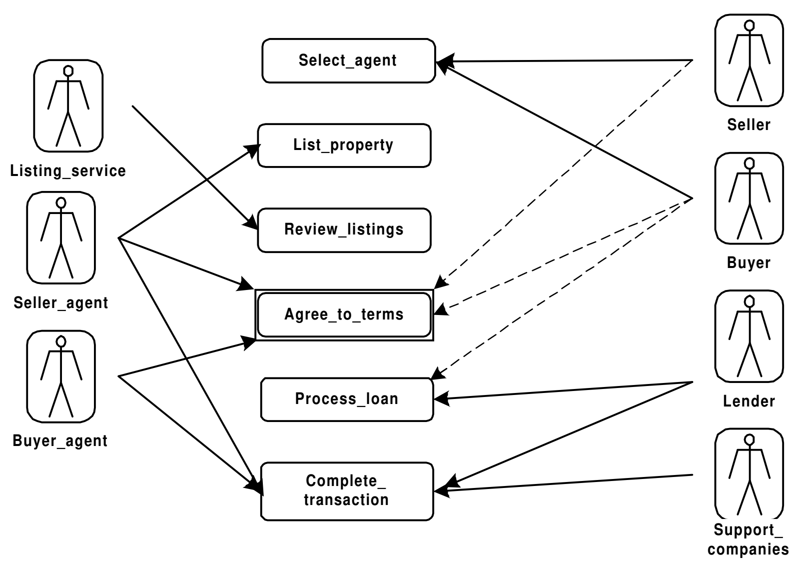


Figure 7High Level Use Cases

The following diagram indicates the relationship between the different use cases.

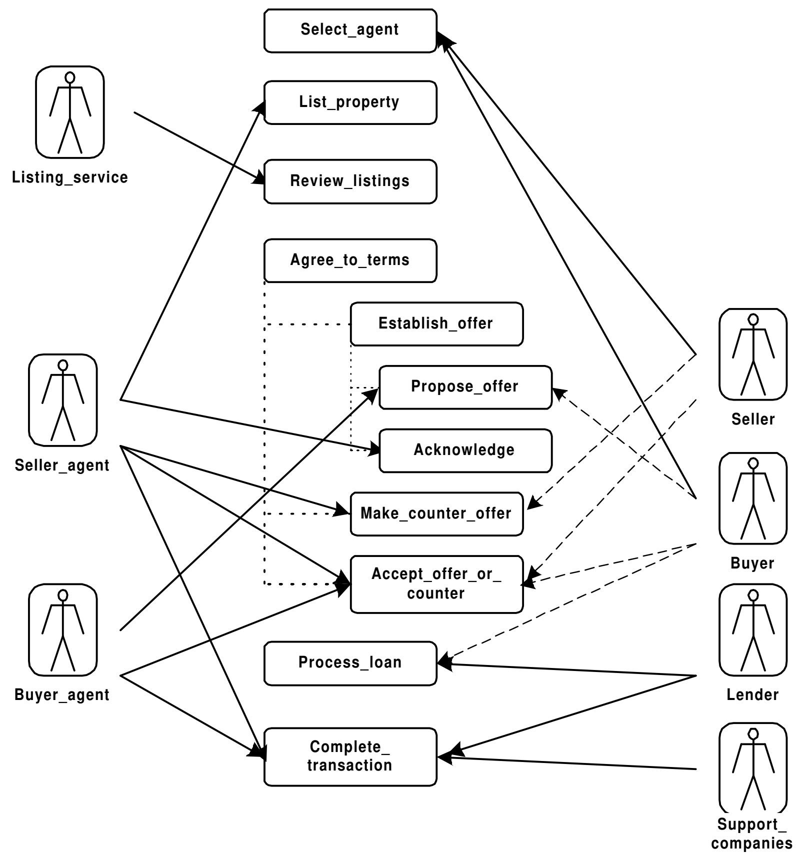


Figure 8 Full Use Case Diagram

6.1.3 Actors in: HL7_Real_Estate_Demonstration_Model

*6.1.3.1.1.1.1 Actor: **Agency***

This is the role of the real estate agencies' systems that support sending listings, offers and counter-offers to other agencies.

Note that Agents are not roles in this model. All of their actions are mediated through the agency for which they work.

*6.1.3.1.1.1.2 Actor: **Buyer***

The individual or corporation that is interested in purchasing a listed property. This actor usually interacts with the system through an agent.

Actions taken by: **Buyer**

20 :: Select_agent
Task of a Buyer to select an agent to represent him.

3 :: Accept_offer_or_counter_offer
Instruct agent to accept.

30 :: Process_loans

5 :: Propose_offer
Establishes the terms of an offer with the Buyer_agent.

6.1.3.1.1.1.3 Actor: ***Buyer_agent***

The individual in a real estate agency who represents the Buyer in transactions for listed properties. This individual makes offers and accepts counter-offers on behalf of the Buyer.

Actions taken by: **Buyer_agent**

3 :: Accept_offer_or_counter_offer
Accept on behalf of principal.

40 :: Complete_transaction

5 :: Propose_offer
Responsibility to compose and transmit the offer on behalf of the buyer.

6.1.3.1.1.1.4 Actor: ***Lender***

The individual or corporation that accepts applications for loans on a property and loans money through a mortgage.

Actions taken by: **Lender**

30 :: Process_loans

40 :: Complete_transaction

6.1.3.1.1.1.5 Actor: ***Listing_service***

This is the role of the organization that oversees the listing service on behalf of the cooperating agencies. It monitors the listings to be certain that all properties are listed in an orderly fashion and that offers and counter-offers are treated according to established rules of priority.

Actions taken by: **Listing_service**

7 :: Review_listings

6.1.3.1.1.1.6 Actor: ***Seller***

The individual or organization that owns a property that is listed for sale. This individual only interacts with the system through the services of an agent.

Actions taken by: **Seller**

3 :: Accept_offer_or_counter_offer
Instruct agent to accept.

4 :: Make_counter_offer
Establish terms for the counter-offer with the Seller_agent.

6.1.3.1.1.1.7 Actor: ***Seller_agent***

The individual in a real estate agency who represents the Seller in transactions for listed properties. This individual creates the listing for the property, makes counter-offers and accepts offers on behalf of the Seller.

Actions taken by: **Seller_agent**

1 :: List_property

3 :: Accept_offer_or_counter_offer
Accept on behalf of principal.

4 :: Make_counter_offer
Transmit a counter-offer as instructed by the seller.

40 :: Complete_transaction

6 :: Acknowledge

6.1.4 Use cases in: HL7_Real_Estate_Demonstration_Model

*6.1.4.1.1.1.1 Use case: **1 :: List_property***

When a real estate Agency wishes to sell a particular property through multiple listing, it assembles the necessary information that describes the property and the asking price, assigns a unique listing identifier, and sends this listing to all other Agency listing systems. This action "creates" a new Listing instance.

When a listing is transmitted to the other Agencies, it must include all of the details of the property, the complete identity of the lender (either individual or organization) if there is an outstanding mortgage, the listing price and terms, the listing agency, the name and priority_phone of the responsible agent, and the name and priority_phone of the agent who manages the agency.

Involves actions by: **Seller_agent**

Derives events: **L01 :: New_listing**

*6.1.4.1.1.1.2 Use case: **10 :: Agree_to_terms***

This high-level use case includes all of the activities involved in making a purchase agreement once a property has been listed. It includes the activities involved in making offers agains a listing, making counter-offers against an offer, and accepting either an offer or a counter-offer.

Is parent of: **2 :: Establish_offer**
3 :: Accept_offer_or_counter_offer
4 :: Make_counter_offer

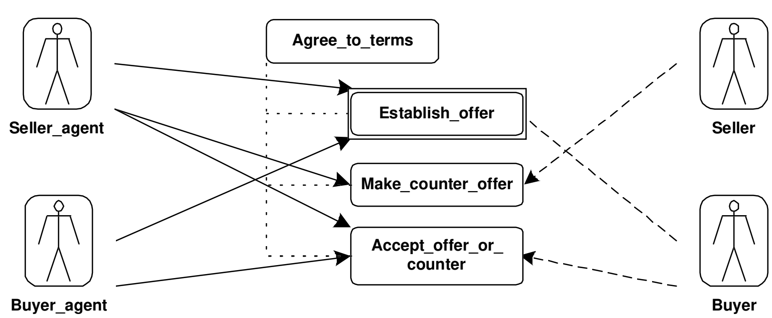


Figure 9 Use Case: Agree to Terms

6.1.4.1.1.1.3 Use case: **2 :: Establish_offer**

When a real estate Agency has a client who wishes to make an offer on a listed property, it creates a new offer and causes this offer to be sent to the Agency that listed the property in question. That agency will acknowledge activation of the offer. If the property is unsold and has no outstanding counter-offers at the time the offer is received, it indicates that the offer is activated. Otherwise it indicates that the offer has not been activated stating which requirement for acceptance was not met.

Is child of: **10 :: Agree_to_terms**

Is parent of: **5 :: Propose_offer**
6 :: Acknowledge

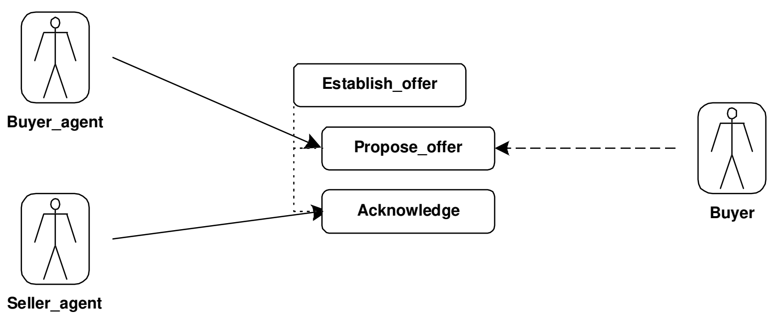


Figure 10 Use Case: Establish Offer

6.1.4.1.1.1.4 Use case: **20 :: Select_agent**

The activity that is undertaken by either a buyer or a seller in engaging an agent to represent them in real estate transactions. It is not represented in the Trigger_events of the toy model.

Involves actions by: **Buyer**

6.1.4.1.1.1.5 Use case: **3 :: Accept_offer_or_counter_offer**

An acceptance is a contract to accept a particular Offer or Counter_offer on exactly the terms specified in that Offer or Counter_offer. Any change in terms would require that a new Offer or Counter_offer be made first.

An Offer or Counter_offer may be accepted up to 36 hours after the Offer or Counter_offer is made, provided that there has been no more recent Offer or Counter_offer on the same Listing and provided that there has been no prior Acceptance issued against the same Offer or Counter_offer.

If an acceptance is received and is in error owing to the expiration of the offer or counter-offer being accepted, or due to an intervening offer, the error condition will be sent to the Agency making the acceptance.

Is child of: **10 :: Agree_to_terms**

Involves actions by: **Buyer**
Buyer_agent
Seller
Seller_agent

Derives events: **A01 :: Acceptance**

6.1.4.1.1.1.6 Use case: ***30 :: Process_loans***

This action is undertaken by the Lender upon receipt of a loan application from a Buyer. It is not represented in the Trigger_events of the toy model.

Involves actions by: **Buyer**
Lender

6.1.4.1.1.1.7 Use case: ***4 :: Make_counter_offer***

When an agency has received an offer on a particular listing, and when the client providing the listing wishes to make a counter-offer to the specific offer, the agency establishes a unique counter-offer identifier and causes this counter-offer to be sent to the agency that made the original offer.

Note that no acknowledgment is necessary since the only way that the Counter_offer would be invalid is if it were made after the expiration of the Offer, and system logic is presumed to prevent this from happening. In the event that an Agency receives a Counter-offer for which no valid Offer exists, this error condition will be returned to the Agency sending the Counter-error.

Is child of: **10 :: Agree_to_terms**

Involves actions by: **Seller**
Seller_agent

Derives events: **O03 :: Counter_offer**

6.1.4.1.1.1.8 Use case: ***40 :: Complete_transaction***

This covers the events and activities surrounding the closing of a sale, including functions related to title insurance, loans, escrow management, etc. It is not represented in the Trigger_events of the toy model.

Involves actions by: **Buyer_agent**
Lender
Seller_agent

6.1.4.1.1.1.9 Use case: ***5 :: Propose_offer***

This is the action of proposing an offer to an Agency that has listed a property, it creates a new offer in the "proposed" state and causes this offer to be sent to the Agency that listed the property in question.

Is child of: **2 :: Establish_offer**

Involves actions by: **Buyer**
Buyer_agent

Derives events: **O01 :: Propose_offer**

6.1.4.1.1.1.10 Use case: ***6 :: Acknowledge***

When an Agency receives an Offer it must reply immediately with an Acknowledgment of that Offer. The response will either indicate that the offer has been "activated" or that it was "not activated." In the latter case the agency will also state which requirement for activation was not met.

If the Offer is for a listing that is still unsold, and for which the listing Agency has not established an unexpired Counter_offer at the time the offer is received, then the listing Agency must acknowledge activation of the Offer against that listing. Otherwise, it indicates that the offer has not been activated and states which requirement for acceptance was not met.

Is child of: **2 :: Establish_offer**

Involves actions by: **Seller_agent**

Derives events: **O02 :: Offer_ack**

*6.1.4.1.1.1.11 Use case: **7 :: Review_listings***

This is an action initiated by the Listing_service to request a complete summary of all of the active Listings for a particular Agency. The Agency must respond with a report covering all of the active (unsold) listings that it has. The report will show the Listing number, Listing Status, and the ID and Name of the Agent responsible for each listing.

Involves actions by: **Listing_service**

Derives events: **S01 :: Summarize_listings**
S02 :: Summary_response

6.2 Information Model

The Information Model for a real estate listing service includes a graphical and a literary expression based on the JWG-CDM framework. Connections and attributes are documented under the class(es) that are relevant.

6.2.1 Information Model Diagram

Here is the graphical representation:

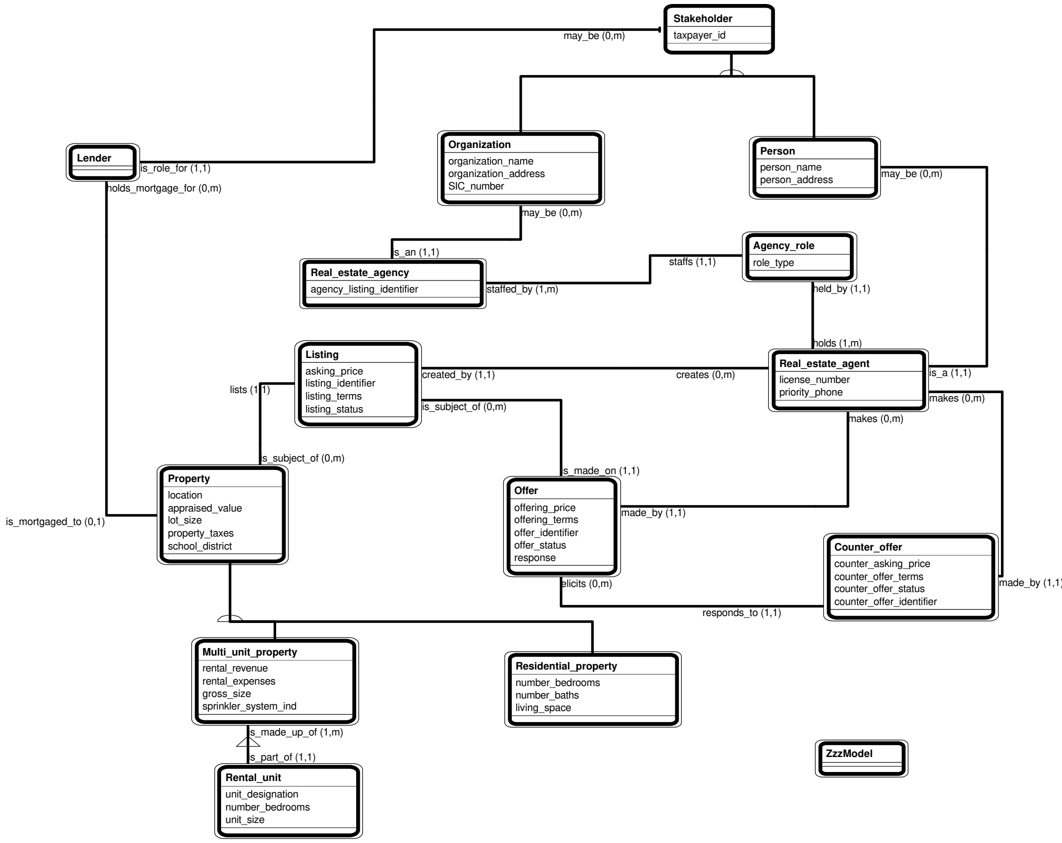


Figure 11 Information Model

6.2.2 Subject Areas

Subject Area: **Participants**

The classes in this subject model the stakeholders in the domain information model.

Contains classes:

Organization
Person
Stakeholder

Subject Area: **Properties**

This subject area models the properties that may be listed or bid upon by the agents represented in this model. It is designed to include undeveloped property, commercial property and single family residences.

Contains classes:

Commercial_property
Property
Rental_unit
Residential_property

Subject Area: **Stakeholder_roles**

These are the roles that stakeholders take on in the domain information model. They include things like agent, agency and lender.

Contains classes:

Agency_role
Lender
Real_estate_agency
Real_estate_agent

Subject Area: **Transactions**

Contains the classes for the limited transactions that have been modeled. These are restricted, for now to listing, offers and counter-offers. The information captured is not about the computer transaction for each, but rather is information for the "contract" that underlies each transaction.

Contains classes:

Counter_offer
Listing
Offer

6.2.3 Classes

This section contains the classes, attributes, and connections for the toy model.

6.2.3.1 Class: Agency_role

Connected to:

Real_estate_agency
Real_estate_agent

Description of: **Agency_role**

Reflects the relationships that may exist between Agencies and Agents. One and only one of the Agents will be the Agency manager.

Connections for: **Agency_role**

staffs (1,1) :: Real_estate_agency :: **staffed_by (1,m)**
held_by (1,1) :: Real_estate_agent :: **holds (1,m)**

Attributes for: **Agency_role**

role_type ::

The type of the role that the agent has in the agency. The only allowed values are "staff" and "manager."

6.2.3.2 Class: Commercial_property

Specialization of: **Property**

Has parts:

Rental_unit

Description of: **Commercial_property**

Defines commercial properties. This specialization of Undeveloped_property includes structures used for commercial (rental) purposes. As such, the property contains parts that are individual rental units.

Whole/part for: **Commercial_property**

is_made_up_of (1,m) :: Rental_unit :: **is_part_of (1,1)**

Attributes for: **Commercial_property**

gross_size ::
Is the gross size of the structure(s) on commercial property measured in square feet.

rental_expenses ::
Gross expense for the most recent calendar year in dollars. Includes all expenses charged against the gross revenue.

rental_revenue ::
Gross revenue of the commercial property expressed as dollars per year, based on the most recent calendar year.

6.2.3.3 Class: Counter_offer

Connected to:
Offer
Real_estate_agent

Description of: **Counter_offer**

Is submitted by the person making the original listing. It is made in response to a particular offer. Based upon the counter offer, the original offer may make a new offer that responds to the counter-offer, but the responds_to relationship is not formally modeled in the domain.

Connections for: **Counter_offer**

responds_to (1,1) :: Offer :: **elicits (0,m)**
Relationship reflects that each counter-offer is made in response to a particular, identified offer.

made_by (1,1) :: Real_estate_agent :: **makes (0,m)**
Relationship between a counter-offer that revises the asking price or terms for a listing, and the identity of the agent responsible for the counter-offer.

Attributes for: **Counter_offer**

counter_asking_price ::
The asking price contained in a specific counter_offer.

counter_offer_identifier ::
A unique identifier assigned to the counter-offer by the real estate agency that is responsible for the counter_offer.

counter_offer_status ::
The status of the counter_offer in its life-cycle. The possible states are - pending, expired, accepted, rejected. With the exception of pending, these are all final states for the class. Each counter_offer responds to a particular offer. If a new offer replaces the offer against which this counter_offer is made, then the counter_offer moves to the "rejected" state.

counter_offer_terms ::

Any special terms and conditions that apply to the counter offer.

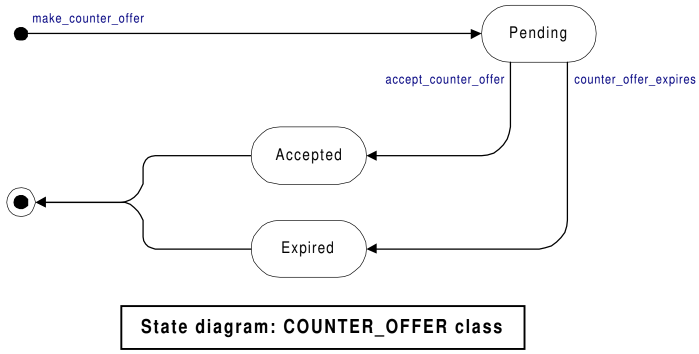


Figure 12 State Transition Diagram: Counter Offer

6.2.3.4 Class: Lender

Connected to:

Property
Stakeholder

<u>Description</u> <u>of:</u> **Lender**

The role involved in holding a mortgage on one or more properties. The role is usually held by an organization, but may also be held by an individual person.

<u>Connections</u> <u>for:</u> **Lender**

holds_mortgage_for (0,m) :: Property :: **is_mortgaged_to (0,1)**

The relationship between the financial institution that holds a mortgage, and the property that is mortgaged.

is_role_for (1,1) :: Stakeholder :: **may_be (0,m)**

6.2.3.5 Class: Listing

Connected to:

Property
Offer
Real_estate_agent

<u>Description</u> <u>of:</u> **Listing**

Is an open offer to sell a given property at a given asking price. Listings are submitted by individual agents on behalf of a seller. Bids (offers to buy) and counter-offers (reductions in asking price for a particular buyer) are made against a specific listing.

<u>Connections</u> <u>for:</u> **Listing**

lists (1,1) :: Property :: **is_subject_of (0,m)**

Relationship between any class of property and the listings that may be made for that property.

is_subject_of (0,m) :: Offer :: **is_made_on (1,1)**

Relationship reflects that each offer to buy a property is made against (or because of) a specific listing of that property. This assures that the appropriate listing agent is credited with the offer transaction.

created_by (1,1) :: Real_estate_agent :: **creates (0,m)**
Relationship between a property listing and the responsible agent.

Attributes for: **Listing**

asking_price ::
The price the seller is asking for the property in the listing.

listing_identifier ::
A unique identifier for the listing that is assigned by the real estate agency that is responsible for the listing.

listing_status ::
The status of a listing provides an indication of where it stands with respect to offers, counter-offers and the like. The initial state is active. From this state, it will move to active_offer if one or more offers are currently valid against the listing. The possible states are: active, active_offer, active_counter, sold, and withdrawn. From active_offer, the listing can move to "sold" as the result of an acceptance; to "active" if the all offers expire; or the "active_counter" if a counter offer is made against one of the offers. From active_counter, the listing will move to 'sold" in the event of an acceptance; to active_offer if the counter_offer expires while one or more offers remain; or to simply active if the counter-offer expires and there are no more offers active. From active, the listing may also be moved to "withdrawn."

listing_terms ::
Particular terms or conditions that qualify a particular listing.

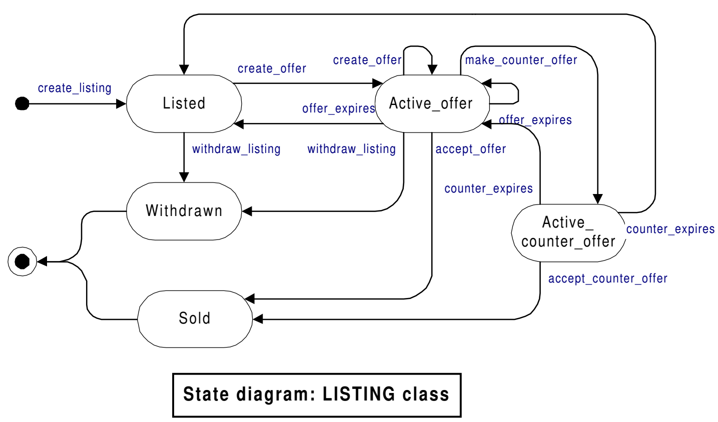


Figure 13 State Transition Diagram: Listing

6.2.3.6 Class: Offer

Connected to:

Counter_offer
Listing
Real_estate_agent

Description of: **Offer**

A bid or offer to buy a listed property at a specific price, and on particular terms. Offers are profferred by individual agents on behalf of a potential buyer.

Connections for: **Offer**

elicits (0,m) :: Counter_offer :: **responds_to (1,1)**

Relationship reflects that each counter-offer is made in response to a particular, identified offer.

is_made_on (1,1) :: Listing :: **is_subject_of (0,m)**

Relationship reflects that each offer to buy a property is made against (or because of) a specific listing of that property. This assures that the appropriate listing agent is credited with the offer transaction.

made_by (1,1) :: Real_estate_agent :: **makes (0,m)**

Relationship between a buy offer and the agent responsible for that offer.

Attributes for: **Offer**

offering_price ::

The price that the buyer is offering to purchase the listed property.

offering_terms ::

Terms and conditions that apply to the offer.

offer_identifier ::

A unique identifier assigned to the offer by the real estate agency that is responsible for the offer.

offer_status ::

The status of the offer as reflected by its life-cycle states. Possible states are: proposed, denied, pending, expired, accepted, rejected. When first created, the Offer is "proposed." As a result of the offer acknowledgement, it will either become "denied" or posted as "pending." All states except proposed and pending are end-states for the class. If a counter_offer is submitted against a particular offer, the offer moves to the rejected state.

response ::

This attribute shows the acknoledgement response to the offer. It is returned by the Agency holding the listing against which the offer is made. The response value will either be "posted" or "denied."

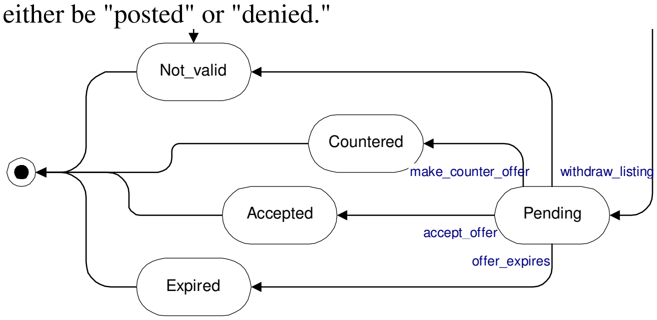


Figure 14 State Transition Diagram: Offer

6.2.3.7 Class: Organization

Specialization of: **Stakeholder**

Connected to:

Real_estate_agency

Description of: **Organization**

Describes any commercial organization. These will be represented by their taxpayer-id, corporate name and corporate address only.

Connections for: **Organization**

may_be (0,m) :: Real_estate_agency :: **is_an (1,1)**

Attributes for: **Organization**

organization_address ::
The principal place of business for this organization.

organization_name ::
The legal name of the organization or corporation.

SIC_number ::
The Securities Industry Code (SIC) number for this organization.

6.2.3.8 Class: Person

Specialization of: **Stakeholder**

Connected to:

Real_estate_agent

Description of: **Person**

Any person about whom information is kept in this domain. Note that these persons may fill several roles.

Connections for: **Person**

may_be (0,m) :: Real_estate_agent :: **is_a (1,1)**

Attributes for: **Person**

person_address ::
The home address for this person.

person_name ::
The person's name.

6.2.3.9 Class: Property

Generalization of:

Commercial_property
Residential_property

Connected to:

Lender
Listing

Description of: **Property**

Root Class for Properties in the Real Estate Model. It identifies the location, asking price, lot size and property taxes for the property in question. Instances of this class represent lots without either residential or rental structures on them. Developed properties are specializations of this class.

Rationale: This class is selected as the site for all of the connections to properties.

Connections for: **Property**

is_mortgaged_to (0,1) :: Lender :: **holds_mortgage_for (0,m)**
The relationship between the financial institution that holds a mortgage, and the property that is mortgaged.

is_subject_of (0,m) :: Listing :: **lists (1,1)**
Relationship between any class of property and the listings that may be made for that property.

Attributes for: **Property**

appraised_value ::
The appraised value of the property.

location ::
Indicates the address, including city, etc. where the property is located.

lot_size ::
Size in acres of the lot upon which the property is situated.

property_taxes ::
The last calendar year's property taxes for the property in dollars.

6.2.3.10 Class: Real_estate_agency

Connected to:

Agency_role
Organization

Description of: **Real_estate_agency**

The role of being a real estate agency that brokers real estate sales. The role is usually held by an organization.

Connections for: **Real_estate_agency**

staffed_by (1,m) :: Agency_role :: **staffs (1,1)**
is_an (1,1) :: Organization :: **may_be (0,m)**

Attributes for: **Real_estate_agency**

agency_listing_identifier ::
The unique identifier assigned to each real-estate agency to identify the agency when it makes listings of properties for sale or submits bids upon listed property.

6.2.3.11 Class: Real_estate_agent

Connected to:
Agency_role
Counter_offer
Offer
Listing
Person

Description of: **Real_estate_agent**

The role, usually held by persons, of being the individual representative for buyers or sellers of real estate. In this model, this role is responsible for submitting all listings and counter-offers on behalf of a seller or bids on behalf of a buyer. These individuals are licensed and can be contacted through their priority telephone.

Connections for: **Real_estate_agent**

holds (1,m) :: Agency_role :: **held_by (1,1)**

makes (0,m) :: Counter_offer :: **made_by (1,1)**

Relationship between a counter-offer that revises the asking price or terms for a listing, and the identity of the agent responsible for the counter-offer.

makes (0,m) :: Offer :: **made_by (1,1)**

Relationship between a buy offer and the agent responsible for that offer.

creates (0,m) :: Listing :: **created_by (1,1)**

Relationship between a property listing and the responsible agent.

is_a (1,1) :: Person :: **may_be (0,m)**

Attributes for: **Real_estate_agent**

license_number ::

The state-issued real-estate license number, stored as a tuplet with the state that issued the license.

priority_phone ::

The phone number through which the individual can be contacted around the clock for high-priority calls.

6.2.3.12 Class: Rental_unit

Is part of: **Commercial_property**

Description of: **Rental_unit**

Summarizes each rental unit in a commercial property.

Whole/part for: **Rental_unit**

is_part_of (1,1) :: Commercial_property :: **is_made_up_of (1,m)**

Attributes for: **Rental_unit**

number_bedrooms ::
Number of bed-rooms in the rental unit.

unit_designation ::
Designation of the rental unit within the property. Example - 3-A.

unit_size ::
The living space in this unit neasured in square feet.

6.2.3.13 Class: Residential_property

Specialization of: **Property**

Description of: **Residential_property**
This class is instantiated for property that includes a single-family residential structure.

Attributes for: **Residential_property**

living_space ::
The total living space in the residential structure measured in square feet.

number_baths ::
Number of bathrooms for the residential property.

number_bedrooms ::
Number of bed-rooms in the residential property.

school_district ::
School district in which the residential property falls.

6.2.3.14 Class: Stakeholder

Is Abstract Class

Generalization of:
Organization
Person

Connected to:
Lender

Description of: **Stakeholder**
The root class for the tree of stakeholders - persons and organizations - about which information is kept in this domain. In most cases, the instances are of interest in the domain because of the roles that they take on. The designation of these roles is represented as an instance connection between Stakeholder (this class) and Stakeholder_Role.

Connections for: **Stakeholder**

may_be (0,m) :: Lender :: **is_role_for (1,1)**

Attributes for: **Stakeholder**

taxpayer_id ::

The taxpayer identification number assigned by the US Internal Revenue Service to the stakeholders, whether they are individuals or corporations.

6.3 Interaction Model

This section of the Toy Model includes:

- Interaction Grid
- Interaction Diagram
- Interaction Profile Table

6.3.1 Interaction Grid

Here are the interactions that are defined for the toy model.

i	Interaction	t	Trigger event	Dependency	s	Sender	r	Receiver	Receiver responsibility
A01	Acccept_offer	A01	Acceptance	Listing for the Acceptance must by in "Active_offer" or "Active_counter" and the Offer or Counter_offer must be in a "Pending" state.	H01	Offer _manager	H01	Offer _manager	Must return an error if the Offer or Counter_offer against which the Acceptance is posted is no longer "pending" or if the Listing is not in the "Active_offer" or
L01	Send_listing	L01	New_listing		L01	Listing _manager	L01	Listing _manager	Activate the listing in the receiving system.
O01	Proposal	O01	Propose_offer		H01	Offer _manager	H01	Offer _manager	Verify that the Offer can be Acknowledged as Active. This requires that the Listing still be Active or Active_offer, but not in Active_counter state. (If it is Active_offer, this becomes a
O02	Acknow-ledge_offer	O02	Offer_ack	Dependent upon a prior occurrence of an O02 event and that the Offer meets all criteria for activation.	H01	Offer _manager	H01	Offer _manager	
O03	Counter	O03	Counter_offer	Listing must be in "active_offer" status and the Offer must be "Pending" If these criteria are not met, the interaction will throw an error.	H01	Offer _manager	H01	Offer _manager	Verify that the Offer is still Pending, else return an error.
S01	Request _listing	S01	Summarize _listings		S01	Service _monitor	L01	Listing _manager	Assemble the necessary report and send it back to the service as an S02.

S02	Summary_report	S02	Summary_response	Prior occurrence of an S01 for which the report has not been filed.	L01	Listing_manager	S01	Service_monitor	

Table 2 Interaction Grid

6.3.2 Interaction Diagram

Figure 15. Interaction Diagram describing the Make Offer Use Case

6.3.3 Interaction Profile Table

This set of interactions implies the following interaction profile:

Sender/Receiver Role	Must Send	Must Receive
Agency_sys_profile	A01, L01, O01, O02, O03, S02	A01, L01, O01, O02, O03, S01
Service_monitor	S01	S02

Table 3 Interaction Profile Table

6.4 Message Design Model

This section of the Toy Model includes:

- Message Object Diagram
- Hierarchical Message Description

6.4.1 Message Object Diagram

The message object diagram will support messages related to listing a property.

Connection	Card	Objects
		Listing
lists	1,1	├─ Property
		│ ├─ *choice of*
		│ │ ├─ *(none)*
		│ │ ├─ Residential_property
		│ │ └─ Multi_unit_property
is_made_up_of	1,m	│ │ └─ Rental_unit
is_mortgaged_to	0,1	│ └─ Lender
is_role_for	1,1	│ └─ Stakeholder_L
		│ └─ *choice of*
		│ ├─ Person_L
		│ └─ Organization_L
created_by	1,1	└─ Real_estate_agent_S
is_a	1,1	├─ Person_S
generalize		│ └─ Stakeholder_S
holds	1,1	└─ Agency_role_S
staffs	1,1	└─ Real_estate_agency
is_an	1,1	├─ Organization_A
generalize		│ └─ Stakeholder_A
staffed_by	1,1	└─ Agency_role_M
held_by	1,1	└─ Real_estate_agent_M
is_a	1,1	└─ Person_M
generalize		└─ Stakeholder_M

Figure 16 Message Object Diagram: List Property

6.4.2 Hierarchical Message Description

This HMD will support messages related to listing a property.

Condition	Requirement	Connection	Card	Objects and Attributes						
				Listing						
			nn		NM	asking_price				
			nn		EI	listing_identifier				
			nn		ID	listing_status				
					FT	listing_terms				
		lists	**1,1**		**Property**					
						NM	appraised_value			
			nn			AD	location			
						NM	lot_size			
						NM	property_taxes			
						ST	school_district			
						choice of				
							(none)			
							Residential_property			
								NM	living_space	
								NM	number_baths	
								NM	number_bedrooms	
							Multi_unit_property			
								NM	gross_size	
								NM	rental_expenses	
								NM	rental_revenue	
								ID	sprinkler_system_ind	
		is_made_up_of	**1,m**					**Rental_unit**		
									NM	number_bedrooms
									ST	unit_designation
									NM	unit_size

Figure 17 Hierarchical Message Description: List Property

7. Extracts from the JWG-CDM Data Model Framework

This section includes selections from the Data Model Framework developed by the Joint Working Group for a Common Data Model. It should be used to supplement the material in the body of the document.

7.1 HL7 model components

This clause describes each component of a data model. The components are presented under three main headings. The following table lists the title for each subclause as a guide to finding a particular description.

- 5.1 Model identification and scope
  - 5.1.1 Model
  - 5.1.2 Scenario
  - 5.1.3 Subject area
- 5.2 Use case model
  - 5.2.1 Actor
  - 5.2.2 Use Case
  - 5.2.3 Action
- 5.3 Interaction model
  - 5.3.1 Trigger event
  - 5.3.2 Interaction
  - 5.3.3 Application profile
- 5.4 Data Model
  - 5.4.1 Class
  - 5.4.2 Generalization-specialization connection
  - 5.4.3 Whole-part connection
  - 5.4.4 Instance connection
  - 5.4.5 Attribute
  - 5.4.6 Unique attribute set
  - 5.4.7 State
  - 5.4.8 State attribute condition
  - 5.4.9 Service
- 5.5 Datatype specifications
  - 5.5.1 Defined datatype
  - 5.5.2 Primitive datatype
  - 5.5.3 Enumerated datatype
  - 5.5.4 Enumerated datatype element
  - 5.5.5 Compound datatype
  - 5.5.6 Compound datatype component
  - 5.5.7 Collection datatype
  - 5.5.8 Constrained datatype

Each component is presented with a short description and a list of information kept about that component. This clause describes the items needed to express the data content of health care informatics standards in a data model.

7.1.1 Model identification and scope

The following set of data model components provides information needed to identify the model and to determine the scope of the model content. The model object describes the model, the subject area identifies the major information categories in the model, and the scenarios provide text to explain the intent of the data modeler and the business case justification for model content.

7.1.1.1 Model

A model is a collection of subject areas, scenarios, classes, attributes, and services depicting the data needed in some aspect of a health care informatics standard. A model has several representations (graphical, textural, etc.). Information kept about a model includes:

7.1.1.1.1 Name

A descriptive title for the data model. Name may include qualifiers like version numbers, preliminary, draft, final, standard, and unauthorized. The name shall be unique within the set of models developed by any particular model developing organization.

7.1.1.1.2 Description

A short narrative describing the scope and intent of the data model.

7.1.1.1.3 Developing organization name

A short form name for the organization responsible for the publication and maintenance of the data model. This is usually formed by using the initials, abbreviation, or acronym associated with the organization.

7.1.1.1.4 Developing committee name

The name of the committee within the developing organization that actually undertakes the development of the data model (e.g. ADT Committee).

7.1.1.1.5 Version number

A number showing the release level of the data model. The version number shall be unique for all public releases of the model. Some organizations may wish to have this number correspond with the version designation of their standard.

7.1.1.1.6 Last modified date

The date the data model was last modified by the model developing organization. Last modified date is primarily a development organization internal control mechanism to distinguish different iterations of the model between public releases.

Model is the highest level qualifier used to decide uniqueness for model components. The combination of Model Name, Model Developing Organization Identifier, Version Number, and Last Modified Date must be unique.

7.1.1.1.7 Model Identifier

A unique identifier assigned to the model by the developing organization.

7.1.1.2 Scenario

A scenario is a statement of health care relevant events supported by the data model, related business models, and information interchange models. Scenarios provide a mechanism for explaining and justifying data modeling decisions that might not be otherwise obvious. Scenarios can be developed at varying levels of detail but should be sufficient to help in identifying required subject areas, classes, attributes, and services. One method for documenting scenarios is the Use Case. Use cases are described in annex B. The purpose of scenarios is to add clarity to the model and assist in understanding the model content. Scenarios may be used to provide explanation for any portion of the data model. Information kept about a scenario includes:

7.1.1.2.1 Identification number

A unique number assigned to the scenario to simplify references to the scenario.

7.1.1.2.2 Title

A short phrase that provides a descriptive title for the scenario.

7.1.1.2.3 Content

The text, diagrams, and tables that constitute the scenario and provide the details necessary to assist in construction and interpretation of the data model. Use Cases can be used as documentation of scenarios. Refer to annex B for a recommendation on use cases.

7.1.1.2.4 Reference to parent model

A reference to the model this scenario is a part of.

7.1.1.2.5 List of class references

An optional reference to one or more model classes cited in the scenario. All classes used in the scenario should be referenced.

7.1.1.3 Subject area

A major category of information represented in the data 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 contain classes and may be the primary subject area for a subset of the classes it contains. Subject area can overlap or be nested in other subject areas. Information kept about subject areas includes:

7.1.1.3.1 Name

The name given to the subject area. Generally the plural form of a noun. A subject area name is often the plural form of the name of the central or dominant class within the subject area.

7.1.1.3.2 Description

Short informative text describing the subject area to clarify what type of classes it includes.

7.1.1.3.3 Reference to parent subject area

A reference to a subject area that is the parent subject area for this nested subject area.

7.1.1.3.4 Reference to parent model

A reference to the model this subject area is a part of.

7.1.2 Use case model

7.1.2.1 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. Actors include both humans and computer systems. The actors interact at the system boundary and thereby delineate the boundary of the system under consideration. A use case represents a complete course of events initiated by an actor and it specifies the interaction that takes place between an actor and the system. The actors represent everything that interacts with the system including people and automated processes. Each kind of actor interacts in a different way, otherwise the actors would not be different. Each way of interacting with the health care system is called a use case. Any particular user may fulfill several roles, and thus perform as several different actors. Information kept about actors includes:

7.1.2.1.1 Name

The actors in the data model are each given a unique name. The actor name is a singular noun or noun phrase.

7.1.2.1.2 Description

A short informative statement that allows people to determine, with certainty, whether a particular real world role is an instance of the actor as conceptualized in the data model.

7.1.2.1.3 Reference to parent model

A reference to the model in which this actor is formally defined and maintained.

7.1.2.2 Use Case

A use case is a generic description of system usage, where system refers in the broadest sense of the healthcare system. A use case describes the possible sequences of interactions among the components of the system and one or more actors in response to some initial stimulus by one of the actors or by a change in the state of an element of the system.
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 data model, related business models, and information interchange 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 one or more trigger events that initiate interactions. If the linkage is to multiple such events, the events should be tightly coupled so that they usually occur together.

Information kept about a use case includes:

7.1.2.2.1 Identifier

An identifier assigned to the use case. The identifier is unique within the scope of the model in which the use case is defined.

7.1.2.2.2 Title

A short phrase that provides a descriptive title for the use case. The title shall be unique within the scope of the model in which the use case is defined.

7.1.2.2.3 Description

The text that describes the use case and provides the details necessary to understand the interactions, transactions and events that are involved in the use case.

7.1.2.2.4 Reference to parent model

A reference to the model in which this use case is formally defined and maintained.

7.1.2.2.5 List of child use case references

An optional reference to one or more other use cases that are children of the use case. The interactions ascribed to all of the children constitute the complete interaction of the parent.

7.1.2.3 Action

An association between use cases and actors that links the actors who take an action in a particular use case to that use case. Information kept about an action includes:

7.1.2.3.1 Description

The text that describes the action and provides the details necessary to understand the association.

7.1.2.3.2 Reference to use case

A reference to the use case in which the action occurs.

7.1.2.3.3 Reference to actor

A reference to the actor that takes the action.

7.1.3 Interaction model

7.1.3.1 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. This is also known as the initiator of an interaction or a trigger event. Each trigger event is linked to one or more leaf level use cases that define the context from which the trigger event is derived.

The occurrence of a trigger event may also be dependent upon the state of one or more objects in the domain or upon the prior occurrence of a different trigger event. Such dependencies allow the definition of a sequence of events and interactions that constitute a transaction.

Information kept about a trigger event includes:

7.1.3.1.1 Identifier

An identifier assigned to the trigger event. The identifier is unique within the scope of the model in which the trigger event is defined.

7.1.3.1.2 Name

A name assigned to the trigger event. The name is unique within the scope of the model in which the trigger event is defined.

7.1.3.1.3 Description

The text that describes the trigger event. When viewed along with the description of the use case from which the event derives, this description must have sufficient detail that the event can be reliably recognized when it occurs.

7.1.3.1.4 Dependency

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.

7.1.3.1.5 Type

The type of occurrence that characterize the trigger event. These include the expiration of some set time duration, the occurrence of a particular date time, a change of state in a particular class in the information model, or a deliberate, external action by a user.

7.1.3.1.6 Reference to parent model

A reference to the model in which the trigger event is formally defined and maintained.

7.1.3.1.7 Reference to defining use cases

Reference to the leaf level use cases that provide the context from which the trigger event is derived.

7.1.3.2 Interaction

An association between a specific message (information transfer), a particular trigger event that initiates or triggers the interaction, and the profiles 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.

Note - The requirement stated above that "an interaction may not specify a return message" does not preclude the return of a "message received" acknowledgment that serves the purpose of providing for guaranteed delivery of the interaction message.

A message is the specific information transfer that occurs in an interaction to meet the requirements of use cases. This is an atomic unit in that the entire message or information content will be sent in an interaction, or no part of it will be sent. That is, no further decomposition is possible.

7.1.3.2.1 Identifier

An identifier assigned to the interaction. The identifier should be unique within the scope of the model in which the interaction is defined.

7.1.3.2.2 Name

A name assigned to the interaction and its message. The name is unique within the scope of the model in which the interaction is defined.

7.1.3.2.3 Description

The text that describes the message and the context for the interaction. The message description should include: references to the classes in the data model that are the subject of the message; and references to the use cases and actors whose requirements the message fulfills. The description is intended to guide the efforts of individuals responsible for designing the message structure.

7.1.3.2.4 Receiver responsibility

The text that describes the responsibility of the receiver of the message once receipt is acknowledged. This is an optional element in that there may no follow-on responsibility. If a responsibility exists it might include explicit manipulations of objects in the domain information space, and/or the initiation of another trigger event that would trigger a subsequent interaction to occur. Transactions can be established through a chain of receiver responsibilities for individual interactions.

7.1.3.2.5 Reference to initiating trigger event

A reference to the trigger event that triggers or initiates this interaction.

7.1.3.2.6 Reference to sending profile

A reference to the application profile that is responsible for sending the message involved in this interaction.

7.1.3.2.7 Reference to receiving profile

A reference to the application profile that is responsible for receiving the message involved in this interaction.

7.1.3.3 Application profile

A defined set of responsibilities that a system or an individual holds with respect to interactions. The profile may have responsibility to send or receive one or more interactions. The application profile and its responsibilities may form the basis for establishing conformance specifications for a standard.
If an application profile is identified as the sender of an interaction, it has responsibilities to recognize the trigger event that initiates the interaction, and to cause the appropriate message or information content to be sent to the receiver.
If an application profile is identified as a receiver of an interaction, it must be prepared to accept the message and to fulfill the further responsibilities defined as the receiver responsibility in the definition of the interaction.
Application profiles may be aggregated into higher level, composite application profiles. In the case of such aggregation, the composite application profile must undertake all of the responsibilities of the lower level profiles that compose it. Note that an application profile need not be part of any composite, and may be part of more than one composite.
Information kept about an application profile includes:

7.1.3.3.1 Identifier

An identifier assigned to the application profile. The identifier is unique within the scope of the model in which the application profile is defined.

7.1.3.3.2 Name

A name assigned to the application profile. The name is unique within the scope of the model in which the application profile is defined.

7.1.3.3.3 Description

The text that describes the application profile and summarizes the interaction responsibilities that are part of that profile.

7.1.3.3.4 Reference to parent model

A reference to the model in which this application profile is formally defined and maintained.

7.1.3.3.5 List of component application profile references

An optional reference to one or more other application profiles that compose a composite application profile. The interaction responsibilities of a composite profile are the aggregate of the individual responsibilities undertaken by the component profiles that make up the composite.

7.1.4 Classes, connections, attributes, and services

Classes, connections, attributes, and services are the primary building blocks of the data 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 services describe the actions which affect the attributes of the class.

7.1.4.1 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. The term *Class* is used in this framework to represent both classes and objects. Classes which cannot be instantiated as objects are referred to as abstract classes. Information documented about a class includes:

7.1.4.1.1 Name

The classes in the data model are each given a unique name. The class name is a singular noun or noun phrase.

7.1.4.1.2 Description

A short informative statement that allows people to determine, with certainty, whether a particular real world thing is an instance of the class as conceptualized in the data model.

7.1.4.1.3 Reference to parent model

A reference to the model in which this class is formally defined and maintained.

7.1.4.1.4 Reference to primary subject area

An optional reference to the subject area that is the primary subject area for the class. A class can appear in many subject areas, this allows one subject area must be designated as primary.

7.1.4.1.5 Abstract class indicator

A indicator that specifies if the 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.

7.1.4.2 Generalization-specialization connection

Generalization-specialization is an association between a class and a specialization of the class. A generalization class can be associated with more than one specialization class. Each specialization class is mutually exclusive. A specialization class may be associated with more than one generalization. The hierarchy or lattice of generalizations and specializations is called a generalization-specialization structure. The specialization inherits the attributes, connections, and services of all of the generalization classes with which it is associated. Information kept about the generalization-specialization connection includes:

7.1.4.2.1 Description

A short informative description of the generalization-specification connection.

7.1.4.2.2 Reference to generalization class

A reference to the generalization class of this connection.

7.1.4.2.3 Reference to specialization class

A reference to the specialization class of this connection.

7.1.4.3 Whole-part connection

An association between classes that depicts the relationship between a whole class and its component parts. Information kept about the whole-part structure includes:

7.1.4.3.1 Description

A short informative description of the whole-part connection.

7.1.4.3.2 Whole to part reference phrase

A short phrase representing the nature of the whole-part connection from the perspective of the whole class to the part class. Example phrases are: "include", "contains", "consist of", "has as parts". Other phrases may also be used. If no phrase is specified, the phrase "contains" will be assumed.

7.1.4.3.3 Part to whole reference phrase

A short phrase representing the nature of the whole-part connection from the perspective of the part class to the whole 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.

7.1.4.3.4 Reference to whole class

A reference to the class that forms the whole of the whole-part connection.

7.1.4.3.5 Reference to part class

A reference to the class that is the part class in the whole-part association.

7.1.4.3.6 Part class cardinality

A set of values and value ranges indicating the number of part class instances involved in the whole class. 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.

7.1.4.3.7 Whole class cardinality

The whole class cardinality is always a minimum and a maximum of one and does not have to be specified.

7.1.4.4 Instance connection

An association between classes that depicts the occurrence of a reference attribute used to connect class instances (objects). The class containing the reference is known as the "source class", the class referred to is the "destination class". Objects can be of the same or different classes. Information kept about an instance connection includes:

7.1.4.4.1 Name

A short action phrase that specifies the nature of the connection from the source class perspective. Each instance connection between the same pair of classes must have a unique name.

7.1.4.4.2 Inverse name

A short action phrase that specifies the nature of the connection from the destination class perspective. Each instance connection between the same pair of classes must have a unique name.

7.1.4.4.3 Description

A short informative statement that describes the relationship between the classes connected by the instance connection.

7.1.4.4.4 Source class cardinality

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.

7.1.4.4.5 Destination class cardinality

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.

7.1.4.4.6 Reference to source class

A reference to the class from which the instance connection perspective is captured.

7.1.4.4.7 Reference to destination class

A reference to the class that is the target of the instance connection.

7.1.4.5 Attribute

Attributes in the data model are the major source of the data content used in health care information standards. 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. Information documented about an attribute includes:

7.1.4.5.1 Name

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.

7.1.4.5.2 Description

A short informative description of the class characteristic captured by the attribute.

7.1.4.5.3 Reference to parent class

A reference to the class the attribute is a part of.

7.1.4.5.4 Reference to datatype

A reference to the datatype that describes the structure, format, and permitted values of the attribute. This reference includes a specification of the attribute's maximum length and a declaration that the attribute as optional or required.

7.1.4.6 Unique attribute set

A collection of attributes whose combinations of values must be unique among the class instances. A unique attribute set is also known as a candidate key. Unique attribute sets are not required for each or any of the classes in the model, but may be used where the modeler feels they add value. Information kept about a unique attribute set includes:

7.1.4.6.1 Attribute set identification number

An identifier for the unique attribute set to simplify references to the set.

7.1.4.6.2 List of attribute references

A reference to one or more attributes that comprise the unique attribute set. All attributes included in this list must be attributes of the parent class of the unique attribute set, or an attribute that has been inherited by the parent class.

7.1.4.6.3 Reference to parent class

A reference to the parent class the unique attribute set is a part of.

7.1.4.7 State

The identification of a unique combination of attribute value(s) and connections which are of interest about a class. The state is determined by the values of a specified subset of the attributes and connections of the class. State is not required for each or any of the classes in the model, but should be defined whenever the state affects permitted values of attributes or connections of the class. It may also be useful to document valid state transitions. Information kept about states includes:

7.1.4.7.1 Name

A word or short verbal phrase that depicts a stable state of a class.

7.1.4.7.2 Description

A short informative description of the state.

7.1.4.7.3 State predicate

The condition or set of conditions that when true about the attribute(s) and connections of the parent class places the class into the declared state.

7.1.4.7.4 Reference to parent class

A reference to the class this state is a part of.

7.1.4.8 State attribute condition

A specification of the allowable attribute values and connections for a specific state. Although all attributes can at sometime have the value of "none" or "unknown," there are certain states in which the concept of "none" or "unknown" would not be acceptable. Indeed, even among the set of permitted values for an attribute there are some states for which certain otherwise valid attribute values are not allowed. The state attribute condition is used to capture these types of state-related attribute value restrictions. Information kept about a state attribute condition includes:

7.1.4.8.1 Condition statement

A statement that specifies constraints on the attribute values. The condition may specify a smaller set of permitted values for the attribute than is suggested by the datatype. A condition statement would also specify if values of "none" or "unknown" are allowed for the given attribute for the given state.

7.1.4.8.2 Reference to parent state

A reference to the state this state attribute condition is a part of.

7.1.4.8.3 Reference to constrained attribute

A reference to the attribute within the class referenced by the state to which this condition applies.

7.1.4.9 Service

A service is a specific behavior that a class is responsible for exhibiting. Services are the means by which state transitions occur, although not all services result in state transitions. The specification of services is beyond the scope of the common data model effort. However, it is important that each SDO define the relevant services for classes in the model. These should be defined if the model developing organization has a requirement for them. It is anticipated that services will be a key link between the data model and the other components of an SDO's standards framework. Information kept about a service includes:

7.1.4.9.1 Name

The name of the service, generally a descriptive verb or verb phrase that communicates the activity the service performs. A service name must be unique within the parent class. The name should make sense when combined with the class Name as the object of the verb or verb phrase.

7.1.4.9.2 Description

Descriptive text describing the service in terms of preconditions (events, triggers, predicate statements, states, etc.) which must be present to enable the service and the effect the service has on attributes of the class. The service description should describe what the service does (not how it does it, who does it, or what technology is used.) The how, who, and why of a service should be part of a scenario.

7.1.4.9.3 Reference to parent class

A reference to the class this service is a part of.

NOTE - Subclauses 5.4.9.4, 5.4.9.5, and 5.2.9.6 need only be specified if they are of use to the modeling organization. They are included here only as a guide to the SDOs to consider in their standards development framework.

7.1.4.9.4 List of affected attribute references

An optional reference to one or more attributes whose values are affected by the service. All attributes referenced must be part of the same class as the service. Affected attributes are those attributes whose values are set or removed by the service.

7.1.4.9.5 List of affected instance connection references

An optional reference to one or more instance connections that are established, transferred, or disconnected by the service. The parent class of the service must be a class participating in the instance connection.

7.1.4.9.6 List of accessed attribute references

An optional reference to one or more attributes whose values are assessed by the service. These include identifiers or unique attribute sets of classes that are the target of affected instance connections and attributes that influence process branches in the service. The values of attributes in this list are unaffected by the service. The attributes in this list may be part of any class in the model; they are not restricted to the parent class of the service the way that affected attributes are.

7.1.5 Datatype specifications

Datatype specifications are used to define the structure, format, and permitted values of attributes. The JWG-CDM has identified five classes of datatypes: Primitive, Enumerated, Compound, Collection, and Constrained. These five types are a beginning set that will be added to as need for additional datatypes are made known. Each of the five datatypes is described in this clause. Datatype specifications constitute a separate model within the data model. They are referenced by the attributes contained in the data model, but the specifications span data models and data modeling organizations. Datatypes are expected to be reused and should be developed with care to allow for maximum reuse capability without sacrificing utility.

7.1.5.1 Defined datatype

Defined datatype is an abstract class. It is specified here to allow those elements that are common to all defined datatypes to be specified once, instead of repeated in each. A defined datatype is always one of the three types: Enumerated, Compound, or Constrained. Information kept in common to all defined datatypes are:

7.1.5.1.1 Name

Each defined datatype is assigned a unique name. The name must be unique across all models developed by the modeling organization.

7.1.5.1.2 Description

A short narrative description of the datatype.

7.1.5.1.3 Reference to parent model

A reference to the model in which this datatype is formally defined and maintained.

7.1.5.1.4 Maintaining organization name

The name of the organization that maintains the datatype specification.

7.1.5.1.5 Last modified date

The date of the last change made to the datatype specification.

7.1.5.2 Primitive datatype

A primitive datatype is the simplest form of datatype. It declares the base form of an attribute value such as string, integer, real, or Boolean. Information kept about primitive datatypes includes:

7.1.5.2.1 Basic datatype

A specification of the basic datatype of a primitive datatype. A basic datatype value can be: string, integer, real, or Boolean.

7.1.5.2.2 Maximum length

The maximum length of the attribute, indicating the maximum number of characters needed in a textual representation of the attribute value. The maximum length can be expressed as unlimited. Maximum length must be greater than zero.

7.1.5.2.3 Precision

Specifies the number of decimal positions to the right of the decimal point of a real datatype component.

7.1.5.3 Enumerated datatype

An enumerated datatype is a specialization of a defined datatype. It specifies a set of values and their meaning. The permitted values of an attribute associated with this datatype are limited to this set of values. The individual values and their meanings are specified in the enumerated datatype elements that are a part of this enumerated datatype. Information kept about the enumerated datatypes includes all the information from defined datatype above and:

7.1.5.3.1 Responsible Organization

The identification of the organization or entity that has maintenance responsibility for the value table.

7.1.5.3.2 Reference to primitive datatype

A reference to the primitive datatype that describes the basic datatype of the value component of the enumerated datatype.

7.1.5.4 Enumerated datatype element

The specification of a value and its meaning. Information kept includes:

7.1.5.4.1 Value

The value being translated.

7.1.5.4.2 Name

The meaning of the value. A single word or phrase that reflects the meaning associated with the value.

7.1.5.4.3 Description

A short informative description of the value. This may include explanatory remarks, exceptions, and other comments to clarify the value meaning.

7.1.5.4.4 Reference to parent enumerated datatype

A reference to the enumerated datatype the enumerated datatype element is a part of.

7.1.5.5 Compound datatype

A compound datatype is a specialization of a defined datatype. It is a datatype that permits an attribute to be composed of multiple parts each of which has its own set of datatype constraints. Information kept about compound datatypes includes the information specified for defined datatypes and:

7.1.5.5.1 Maximum Length

The maximum aggregate length for the compound datatype, indicating the maximum number of characters needed in a textual representation of the attribute value. The maximum length may be less than the sum of the components.

7.1.5.5.2 Reference to datatype

A reference to the datatype that specifies the structure, format, and permitted values of the compound datatype component.

7.1.5.6 Compound datatype component

A part of a compound datatype. Information kept about the compound datatype component includes:

7.1.5.6.1 Name

A singular noun used to identify the compound datatype component.

7.1.5.6.2 Preferred order

A positive integer that indicates the preferred order of presentation for this component.

7.1.5.6.3 Description

A short informative description of the compound datatype component.

7.1.5.7 Collection datatype

A datatype specification that defines the cardinality and form of repeating attributes. Information kept about a collection datatype includes:

7.1.5.7.1 Number of occurrences

The maximum number of elements in the collection. The maximum occurrence may be specified as unlimited.

7.1.5.7.2 Collection Type

A specification of the type of collection. At this time the list type is the only type defined in this standard. Other candidate types that may be considered for future inclusion in the standard are array, ordered list, set, and bag.

7.1.5.7.3 Reference to datatype

A reference to the datatype specification for the elements in the collection. This may be a reference to a Defined datatype or to a Primitive datatype, but shall not be a reference to another Collection datatype.

7.1.5.8 Constrained datatype

A constrained datatype is a specialization of a defined datatype. It is a datatype specification which details specific criteria that values of the attribute must meet in order to be considered valid.

7.1.5.8.1 Predicate statement

An acceptance rule that states the conditions that an asserted value should have in order to be accepted as a legal value of the attribute.

7.1.5.8.2 Reference to datatype

A reference to the datatype that specifies the domain the attributes associated with the constrained datatype.

7.2 Expression of model content

This clause provides the rationale, formats, and specifications for expressing the content of data models developed using this framework. It is divided into four subclauses. It starts with the rationale for these expressions and a description of them. This is followed by the specification of formal representations and formats that are used in all of the expression forms. Finally, there is a detailed specification for the Literary expression, and the Graphical expression. Consideration was been given to developing an expression in a Formal Description Language (FDL). This was not undertaken for this version of the standard.

7.2.1 Types of expression in this standard

As data modelers develop and subsequently make use of their data models, they will need a standard way of expressing the content of those models to serve their own purposes and the purposes of others. Each of these purposes presents a different set of requirements. For this reason, this standard specifies a set of different, mutually compatible expressions to be used.

7.2.1.1 Literary expression

The literary expression of the data model should be included by data modelers when a model is submitted to other SDOs for discussion and when it is submitted to the JWG-CDM for harmonization with the CDM.

The literary expression is a structured document that contains the full semantics of the data model, which expresses all of the normative content of the model and includes all of the informative, descriptive information necessary for readers to understand the intent of the modelers as the latter developed the content.

The literary expression is designed to be readily maintained with simple word-processing tools. Even though it requires a standard structure, it is designed to be read and interpreted by individuals who understand the general concepts underlying object-oriented models, but who may not wish to submit to the rigor that an FDL would require.

7.2.1.2 Graphical expression

The graphical expression is a picture of the data model. It provides an alternate way to depict the core content of the model. It provides the viewer with the essential features of each class, the names of the attributes in the class, the connections between classes, the cardinality of those connections, and the inheritance structure within which each class is placed. Modelers submitting models for harmonization by the JWG-CDM and/or use by other SDOs need not provide a graphical expression, but if a graphical expression is used the expression defined in clause 6.4 of this standard shall be used.

7.2.2 Formal representations and formats

The elements that define the data model will ultimately be parsed and stored in data systems. Also, it is likely that the classes in the data model will need to undergo formal registration in order to be used with some standards. Therefore, it is necessary to restrict certain elements defined in the data model, or used in data model expression, to formal representations. These formal representations and the requirements for their usage follow.

7.2.2.1 Representation of unique names

7.2.2.1.1 NameString for unique names

Various elements of the data model are required to be unique throughout the model or within a limited scope within the model, as specified in subsequent clauses. These elements are listed in the following table, which shows the name of the restricted attribute, the meta-model component to which that attribute belongs, the range over which the attribute is required to be unique, and the preferred case for the initial character of the restricted attribute.

Restricted attribute	Meta-model component	Range for uniqueness	Initial character
name	Data_model	Models from any given SDO	Upper-case
name	Subject_area	Data_model	Upper-case
name	Class	Data_model	Upper-case
name	Attribute	Class	lower-case
name	Instance_connection	pairs of connected Classes	lower-case
inverse_name	Instance_connection	pairs of connected Classes	lower-case
name	Service	Class	Upper-case
name	State	Class	Upper-case
name	Datatype	Model	Upper-case

In a data model defined under this standard, the unique value used for the above meta-attributes shall be of datatype 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); 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 constrained as follows. The first character of all names in the above table, with the exception of attribute name, should begin with an upper case (capital) letter, as indicated in the right-hand column of the table. Conversely, attribute name and the two phrases for Instance connection should begin with a lower case character. 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.

The naming and listing of the meta-model elements in this clause of the standard follow these rules and use the preferred capitalization convention presented above.

7.2.2.1.2 Specification of fully qualified names

Although this standard does not require such references, there may be times in documenting a particular data model when the modeler will need to refer to a particular attribute of a particular class and need to distinguish this from an attribute of a different class that has the same attribute name. In that instance, the qualified name of the attribute shall be expressed by concatenating the NameString names of the class name and attribute name, in that sequence, using the dot character (.) as a separator. In this terminology, the qualified name of the date and time for a lab procedure in the Sample Data Model (Annex C) would be "Lab_Procedure.date_time".

If a modeler needs to distinguish between elements drawn from more than one data model, then the model name must also be included in the qualified name with the element concatenated to it. If a unique model identifier has been assigned to the model, this identifier may be used as the fully qualified name of the model. If a unique identifier has not been assigned, then the fully qualified model name shall be the concatenation in the following sequence of: the Model.developing_organization_identifier; the Model.name; the Model.version_number expressed as "Vn-m", where 'n' is the version number and 'm' is the release number; and the Model.effective_date expressed in the standard Date datatype (CCYYMMDD). The dot character (.) shall be used as a separator in this concatenation.

For example, the version of the Sample Data Model provided in annex C has a fully qualified name of: "IEEE.Sample_Data_Model.V1-0.19940425" and the fully qualified name of the date and time for a lab procedure in that model would be "IEEE.Sample_Data_Model.V1-0.19940425.Lab_Procedure.date_time". If the same model had been assigned the identifier "IEEE_SMP_0001" then the same element could be referred to as "IEEE_SMP_0001.Lab_procedure.date_time".

7.2.2.2 Representation of unique identifiers

Various data model elements in the Use model and Interaction model are required to have identifiers that are unique throughout the model, as specified in subsequent clauses. These elements will be specified as being represented by an IdentifierString.

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, as discussed in clause 6.2.1.2. However, each of the elements that uses an IdentifierString also has another unique identifier that can be used in building fully qualified names.

7.2.2.3 Representation of descriptive text

In most instances, the information to be kept about model components, as specified in Clause 5, includes provision for a textual description of the component. Initial efforts to document data models using this draft standard have revealed the need to structure these descriptions in order to provide for cross reference to particular standards, identification of open issues, explanation of modeling rationale, etc.

Descriptions of model components shall be of type DescriptiveText, as specified in this sub-clause.

7.2.2.3.1 Descriptive text components

In order to accommodate these requirements without extending the number of components to be specified, this standard provides for the representation of up to five different components within each description. The basic description component is contained in the first paragraph(s) of the descriptive text. The remaining components, if they are included, are identified by the appearance of a reserved keyword at the beginning of the paragraphs. The following table lists the key words that are used to introduce each component, and the rules governing the content and frequency of occurrence of each component.

Component	Key word	Purpose
Basic description		The basic description component is made up one or more paragraphs which serve the primary purpose of describing the data model component in question. The basic description has no keyword; shall be the first paragraphs in the DescriptiveText; and includes all paragraphs up to the first occurrence of a paragraph that starts with one of the key words specified in this table. Paragraphs that are part of the basic description shall not begin with any of the four reserved keywords specified in this table. The basic description component may not be repeated within a DescriptiveText.
Standards reference	StdRef	The standards reference component provides a vehicle by which individual SDOs may reference data model components to particular elements of their standards. Each standards reference component shall be a single paragraph that begins with the keyword "StdRef." There is no restriction on what may follow this key-word. Thus the SDOs can extend this component with their own reserved words in order to define sub-components. A DescriptiveText may include multiple standards reference components.
Modeling rationale	Rationale:	This component is designed to allow the data modeler to document the rationale or justification for the specification of a particular data model component. This rationale may occupy one or more paragraphs in the DescriptiveText. The first paragraph of the modeling rationale component shall begin with the keyword "Rationale:". Subsequent paragraphs of the modeling rationale component, if any, shall not begin with any of the four reserved keywords specified in this table. Only one modeling rationale component should appear in each DescriptiveText.
Open issue	OpenIssue:	This component of the DescriptiveText is intended to allow the data modelers to identify and discuss any open issues that remain to be resolved with respect to the model component being described. This issue statement may occupy one or more paragraphs in the DescriptiveText. The first paragraph of an open issue component shall begin with the keyword "OpenIssue:". Subsequent paragraphs of each open issue component, if any, shall not begin with any of the four reserved keywords specified in this table. There may be multiple open issue components in a DescriptiveText.
Scenario reference	Scenario:	This component permits the data modeler to link a data model component to a specific Scenario in the same data model. This component shall be a single paragraph that opens with the keyword "Scenario:". Following the keyword, the component shall be made up of a comma-delimited list of the scenario identification numbers for the scenarios in which the element is discussed. There shall be no more than one scenario reference component in each DescriptiveText.

NOTE - The key-word "StdRef" does not include a colon as do the other key-words. This is done because it is expected that each SDO will extend this keyword with its own sets of keywords, and that these extensions will include a colon. An example from HL7 are the reference key phrases "StdRef HL7 Table:" and "StdRef HL7 Segment:"

7.2.2.3.2 HL7 standards reference types

HL7 has extended the standards reference component of the DescriptiveText discussed in the preceding sub-clause. The HL7 reference components, key words, and descriptions are in the following table.

Component	Key word	Purpose
Reference	StdRef HL7 Ref:	Identifies the Version of the HL7 standard in which this component first appeared or was first modeled.
Segment	StdRef HL7 Seg:	Identifies the V2.x Segment from which the component was derived.

Element	StdRef HL7 El:	Identifies the V2.x Data Element from which the component was derived.
Table	StdRef HL7 Table:	Identifies the V2.x Table from which the component was derived.
Technical committee	StdRef HL7 TC:	Identifies the HL7 Technical Committees and SIGS whose PIMs include the identified component.

7.2.2.3.3 Deprecated usage

It is intended that there be few restrictions on what a data modeler may enter in descriptive fields. In order that the expression of these models can be automatically parsed, however, there are a few restrictions that should be observed in writing descriptions. These restrictions are:

a) The following special characters should not be used in descriptions: left curly bracket ({), right curly bracket (}), and vertical bar (|) .
b) The character combination of a colon surrounded by spaces (' : ') should not be used in descriptions.
c) The reserved words CONDITIONAL, MANDATORY, OPT, OPTIONAL, and REQUIRED expressed in all capital letters should not be used in descriptions.
d) The special 'standard phrases' specified as part of the line patterns in the literary expression should not be used in descriptions.

7.2.2.4 Representation of cardinality and datatype elements

7.2.2.4.1 Representation of connection cardinality

A set of values and value ranges including the minimum and maximum occurrence are required for instance connections and whole/part relations in the meta-model. A CardinalityString shall be used to represent this set in both the literary and graphical expressions of the model. The CardinalityString is a constrained string. It is built according to the following rules:

6.2.3.1.1 A CardinalityString shall have at least a minimum and a maximum value.

6.2.3.1.2 A CardinalityString may also include ranges or individual values to define a set.

6.2.3.1.3 A CardinalityString may also include an open ended range at the upper end.

6.2.3.1.4 A CardinalityString shall be expressed as a series of one or more elements enclosed in parentheses.

6.2.3.1.5 The elements in the series making up a CardinalityString shall be separated by a comma (,).

6.2.3.1.6 The elements making up a CardinalityString shall be either a range, zero, a positive integer, or the letters "m" or "n".

6.2.3.1.7 A range element in a CardinalityString shall be represented by a pair of periods (..) separating a lower value that must be an integer and an upper value that may either be an integer or the letter "m" or the letter "n". The range includes both its upper and lower values.

6.2.3.1.8 If the letter "m" (preferred) or "n" appears in a CardinalityString, 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 CardinalityString.

6.2.3.1.9 The minimum value for the cardinality shall be the smallest integer in the CardinalityString.

6.2.3.1.10 The maximum value for the cardinality shall be the largest integer in the CardinalityString, and must be greater than zero.

6.2.3.1.11 The elements making up a CardinalityString should be ordered in ascending order, but are not required to be.

7.2.2.4.2 Representation of data element length

A representation of the maximum length of the basic datatypes String and Integer, and a representation of the maximum number of elements in a List are required as part of the datatype specification for attributes. The LengthString shall be used in these cases. The LengthString is a constrained String. It is built according to the pattern "(x)". The element, "x", shall be either a positive integer or the character 'm.' This value expresses the maximum length of the component, or the maximum number of elements in a List. If an integer is used, it is the maximum. If the character 'm' is used it represents an indeterminate maximum. If the LengthString is omitted at any place, the default value shall be "(m)".

7.2.2.4.3 Representation of precision

A representation of precision expressed as the number of digits required to the right of a decimal point is part of the specification of the Real datatype. The RealString shall be used in these cases. The RealString is a constrained String. It is built according to the pattern "(x.s)". The first element ('x') shall be either a positive integer or the character 'm.' This value expresses the maximum length of the number. If an integer is used, it is the maximum. If the character 'm' is used, it represents an indeterminate maximum. The second element in the pattern ('s') shall be a positive integer specifying the minimum number of digits to the right of the decimal point in the real number.

7.2.2.4.4 DatatypeString entries for attributes

In the literary and graphical expression for data models, it is necessary to specify the datatype for each attribute in the data model. This specification is represented by a constrained String known as a DatatypeString. The rules for constructing this string follow.

7.2.2.4.4.1 Optionality indicator

The first component in a DatatypeString is a field that indicates whether the DatatypeString is optional, required, or conditional. This component itself is optional. The values that may be assigned to this component are: 'OPTIONAL', 'OPT', and '?' (meaning the component is optional); 'REQUIRED', 'MANDATORY', and 'M' (meaning the component is required); and 'CONDITIONAL' (meaning the optionality is determined by a state of the class to which the attribute belongs).

The presence of this first component in a DatatypeString is optional. If the component is omitted, the default assumption shall be that the attribute or attribute part being specified is required. If this component of the DatatypeString is present, it shall be separated from the remainder of the DatatypeString by a space.

7.2.2.4.4.2 Datatype name and modifiers

The second component of the DatatypeString shall be the name of a particular datatype including, where appropriate, the LengthString or RealString for that datatype. Allowable datatype names are the names of any datatypes that are defined within the model; the names of datatypes defined in other models; the names of the Basic types; and the datatype name List. Thus, the datatype name and modifier must come from the following list:

a) **Datatype.name ::** The name of a datatype defined within the model being expressed, or the fully qualified name of a datatype defined in some external model. In the latter case, the name is the concatenation of the fully qualified model name of the source model and the datatype name, as specified above.

b) A name of a Basic datatype which shall be one of the following:

  1) **Boolean ::** The name of the Boolean Basic datatype.

  2) **Real (x.s) ::** The name of the real Basic datatype combined with the RealString that represents the length and precision of that real number.

  3) **Integer (x) ::** The name of the integer Basic datatype combined with the LengthString that represents the length of the integer. The LengthString may be omitted in which case the default value "(m)" will be assumed.

  4) **String (x) ::** The name of the string Basic datatype combined with the LengthString that represents the length of the string. The LengthString may be omitted in which case the default value "(m)" will be assumed.

c) **List (x) of: ::** The name of the list datatype including the LengthString that represents the maximum number of occurrences within that list. The LengthString may be omitted in which case the default value "(m)" will be assumed.

If the datatype name is List ('c' in the list), then there shall be a third component of the DatatypeString as specified in the following subclause.

7.2.2.4.4.3 DatatypeString for Lists

If the datatype name in the second component of the DatatypeString being defined is List, then the DatatypeString shall have a third component. The third component shall consist of a properly defined DatatypeString that specifies the datatype of the entries in the List except that the DatatypeString following a List(x) of: shall have a required optionality. This is a recursive definition. The definition may recur as often as necessary but the specification "List of List of" should not be used.

7.2.2.5 Literary expression format

The literary expression of a data model is a structured document that presents all of the normative and informative information that is contained in the model. The document is built in outline form and includes four major sections. Each of the sections contains one or more subsections that may, in turn, be subdivided. The maximum depth of this structure is four levels, where level one is the section, level two is the subsection, etc.

This hierarchical structure, the indented format in which it should be documented, and the rules applied to each section are designed to provide a document that can be easily read and within which specific information can be found rapidly. Secondarily, the specifications permit the document to be automatically generated from the output of graphical modeling tools and to be parsed in order to provide source data for such graphical tools and for tools that can validate and document the model.

7.2.2.5.1 Indentation

The literary document is intended to be created with simple word processing or text editing programs. The document structure, which allows a reader to find specific information rapidly, is made evident through indentation where each succeeding level is indented to the right of the preceding level. The amount by which each succeeding level is indented should be constant, and the indentation increment should be between 0.3 in and 0.7 in on a printed page. The typeface(s) used in this document shall be 10 (ten) point size or larger.

7.2.2.5.2 String separators and line spacing

At several places in the definition of both the literary and graphical expression for data models, this standard calls for the concatenation of two or more NameStrings to form a compound expression. When this is called for, any combination of one or more spaces and/or one or more colons (:) may be used as the separator between each pair of NameStrings. The use of two colons surrounded by spaces (::) is preferred as a separator. The Sample Data Model in annex C demonstrates a selection of separators that are designed to enhance readability.

A blank line should be used to separate individual lines and paragraphs in the literary expression.

7.2.2.5.3 Text attributes

This standard specifies no requirement for typeface, or the use of bold-face and underlined text styles. However, authors are encouraged to use these attributes to enhance the legibility of the literary expression when possible. One example of this usage can be seen in the literary expression for the Sample Data Model contained in annex C.

7.2.2.6 Literary expression pattern notation

As each section of the literary expression is specified, this standard will specify the level of indentation for that section or subsection and a format by which the lines in that subsection are to be constructed. A pattern is used to convey this information. The remainder of this clause specifies the notation used in the patterns.

The pattern for a line or a set of lines in the expression contains: the indentation level, a standard phrase, and the content to be placed in the line. The pattern for the beginning of a class definition section will be used as an example for the discussion in this clause. The pattern is:

L1   <u>Class:</u> <Class name>

7.2.2.6.1 Indentation level

Each pattern starts with the letter "L" followed by a single digit. The digit specifies the indentation level, where L0 is placed at the left margin with no indentation, L1 is indented one step, L2 is indented two steps, etc.

7.2.2.6.2 Standard phrase

A standard phrase follows the indentation level in most, but not all of the patterns. Wherever a pattern is used in the document, the specified standard phrase shall begin the line, except in the circumstance where a pattern is used to specify a sequence of single lines. In the latter case, the standard phrase in the pattern should not be used in the second and subsequent lines of the sequence.

These phrases are shown as underlined text in the pattern. When the phrases are used in expressing a data model they may be underlined, but they are not required to be. The standard phrase for the example pattern above is "Class:".

7.2.2.6.3 Line content

The third element of a pattern represents the line content that shall appear on that line. Most, but not all patterns call for such values. These values may be single elements as in the class name called for in the example pattern, or they may be compound values built by combining individual values according to simple rules. A descriptive phrase for the line content is placed in the pattern enclosed in a pair of angled brackets (< >). A precise definition for the descriptive phrase is provided in the paragraph that follows the pattern.

7.2.3 Literary expression specification

The structured elements that make up the literary expression of a data model are specified in this clause. The sequence and listing are very similar to that used in Clause 5. Annex C provides a proper literary expression of a sample data model. Readers will find it helpful to refer to annex C for illustrations of each of the specifications in the subclauses that follow. Further, the following table lists the title for each subclause as a guide to finding a particular specification.

- 6.3.1 Model identification and scope section
  - 6.3.1.1 Model identification
  - 6.3.1.2 Scenario definitions
  - 6.3.1.3 Subject area definitions
- 6.3.2 Use case model
  - 6.3.2.1 Actors
  - 6.3.2.2 Use cases
- 6.3.3 Interaction model
  - 6.3.3.1 Trigger events
  - 6.3.3.2 Interactions
  - 6.3.3.3 Application profiles
- 6.3.4 Data model
  - 6.3.4.1 Class summary
  - 6.3.4.2 Generalization-specialization
  - 6.3.4.3 Whole
  - 6.3.4.4 Parts
  - 6.3.4.5 Instance connections
  - 6.3.4.6 Attributes
  - 6.3.4.7 Unique attribute sets
  - 6.3.4.8 States
  - 6.3.4.9 Services
- 6.3.5 Datatype specifications
  - 6.3.5.1 Enumerated datatype
  - 6.3.5.2 Compound datatype
  - 6.3.5.3 Constrained datatype
- 6.3.6 External references
  - 6.3.6.1 Referenced classes
  - 6.3.6.2 Referenced datatypes

7.2.3.1 Model identification and scope section

The model identification and scope section shall be the first section of a literary expression document. This section provides information about the model including supporting scenarios and subject areas, a description of the model's scope, and the organization responsible for developing the model. The pattern for the first line is:

L0 <u>Data model:</u> <Model name>

The line content shall be name of the model expressed as a NameString.

7.2.3.1.1 Model identification

The model identification subsection shall be the first subsection under model identification and scope. It provides the name of the model, the organizations responsible for developing the model, the model version, and a description of the model's scope. The pattern for the first line is:

L1 Identifications: <null>

This line has no content. It shall be followed by:

L2 Organization: <Developing organization name>

The line content shall be the name of the standards developing organization responsible for developing the model.

L2 Version: <Version number and last modified date>

The line content is the version and release number of the model followed by the last modified date. The version shall be expressed as "Vn-m", where 'n' is the version number and 'm' is the release number. The last modified date which follows shall be expressed in the standard Date datatype (CCYYMMDD). A space shall separate these values in the content.

L2 ModelID: <Assigned model identifier>

If a specific model identifier has been assigned by the developing organization, or some other group seeking to manage multiple models, this line shall appear. The line content is the assigned model identifier.

L2 Developed by: <Developing Committee name>

The line content shall be a string that contains the name and other identifiers of the committee within the developing organization that developed the model.

L2 Description of: <Model name>

The name of the model shall be expressed as a NameString.

L3 <Model description>

The textual description of the intent and scope of the model shall follow and be expressed as DescriptiveText. This concludes the model identification subsection.

7.2.3.1.2 Scenario definitions

The scenarios definitions subsection shall be the second subsection in the model identification and scope section. The first line of this subsection shall be:

L1 Scenarios for: <Model name>

The name of the model shall be expressed as a NameString.
Following the opening line there shall be one sub-subsection for each scenario defined in the data model. The scenarios shall be listed in the order of their Identification numbers. Each sub-subsection shall open with:

L2 Scenario: <Scenario Identification_number and Title>

The line content in this pattern shall be the identification number of the scenario followed by its Title. A period and a space should separate the number from the title string.

L3 <Scenario Content>

The scenario content that describes the scenario shall follow in one or more paragraphs. Note that scenarios may be more complex than simple text, and may include diagrams, tables, etc. The scenario content concludes the sub-subsection for each scenario.

NOTE - In order to retain the ability to parse the literary expression, a standard separator will need to be defined to surround scenario content that contains other than simple text. This specification will be added in later versions of the standard.

L3 Class references: <Class name>

This line shall be repeated for each of the classes that is referenced in the scenario. The line content shall be the name of the contained class as a NameString. The lines shall be listed in alphabetic order by the names of the classes in the list.

7.2.3.1.3 Subject area definitions

The subject area definition subsection shall be the third subsection in the model identification and scope section. The first line of this subsection shall be:

L1 Subject areas for: <Model name>

The name of the model shall be expressed as a NameString.
Following the opening line there shall be one sub-subsection for each subject area defined in the data model. The subject areas shall be listed in alphabetic order by their names. Each sub-subsection shall open with:

L2 Subject area: <Subject_area name>

The name of the subject area shall be expressed as a NameString.

L3 <Subject_area description>

The text that describes the subject area shall follow expressed as DescriptiveText.

L3 Parent area: <Parent_subject_area name>

If a parent subject area is identified for this subject area, this line shall be used with the NameString name of the parent as the line content.

L3 Contains classes: <Class name>

This line shall be repeated for each of the classes that is contained in the class list for this subject area. The line content shall be the name of the contained class as a NameString. The lines shall be listed in alphabetic order by the names of the classes in the list.

7.2.3.2 Use case model

If the model contains definitions for use cases, and actors, then the use case model definitions section shall be the second section in the literary expression. It opens with:

L0 Use case model for: <Model name>

The name of the model shall be expressed as a NameString.

7.2.3.2.1 Actors

If the model includes one or more actors, this subsection shall be used. The first line of this subsection shall be:

L1 Actors in: <Model name>

The name of the model shall be expressed as a NameString. Following the opening line there shall be one sub-subsection for each actor in the model, listed in alphabetic order by the actor name. The sub-subsection shall open with:

L2 Actor: <Actor name>

The name of the actor shall be expressed as a NameString.
Following the actor identification line, there will be a description sub-subsection and a use case action sub-subsection for each actor. The specification for these sub-subsections are in the remainder of this subclause. In this subclause the phrase "subject actor" will be used to refer to the actor being defined.

L3 Description of: <Actor name>

If a description is provided for the subject actor, this line shall be used. The line content is the name of the subject actor expressed as a NameString. It shall be followed by:

L4 <Actor description>

The line content is the text that describes the subject actor expressed as DescriptiveText. If the subject actor has an action in one or more use cases in an action relationship, the following two lines shall be listed for each of the use cases. The order of listing shall be the identifier sequence of the use cases.

L3 Takes action in: <Use case identifier: Use case name>

The line content of the first of the pair of lines shall be a compound built of two components. The first component shall be the use case identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the use case name. If there is a description for the action relationship identified on the first line, then it shall be followed by the second line:

L4 <Action description>

The line content shall be the text of the description for the Action association expressed as DescriptiveText.

7.2.3.2.2 Use cases

If the model includes one or more use cases, this subsection shall be used. The first line of this subsection shall be:

L1 Use cases in: <Model name>

The name of the model shall be expressed as a NameString. Following the opening line there shall be one sub-subsection for each use case in the model, listed in order of the use case identifier. The sub-subsection shall open with:

L2 Use case: <Use case identifier: Use case name>

The line content shall be a compound built of two components. The first component shall be the use case identifier. Following a separator, the second component shall be the NameString that expresses the use case name. Following the use case identification line, there will be a description subsection, a heritage subsection and an action subsection for each use case. The specification for these sub-subsections are in the remainder of this subclause. In this subclause the phrase "subject use case" will be used to refer to the use case being defined.

L3 Description of: <Use case name>

If a description is provided for the subject use case, this line shall be used. The line content is the name of the subject use case expressed as a NameString. It shall be followed by:

L4 <Use case description>

The line content is the text that describes the subject use case expressed as DescriptiveText. If the subject use case is a child of one or more other (parent) use cases, the following line shall be used:

L3 Is child of: <Parent use case identifier: Parent use case name>

This line shall be repeated once for each defined parent of the subject use case. The line content shall be a compound built of two components. The first component shall be the use case identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the use case name. The order of listing shall be the numeric order of the parent use case identifiers.

If the subject use case is a parent to one or more other (child) use cases, the following line shall be used:

L3 Is parent to: <Child use case identifier: Child use case name>

This line shall be repeated once for each defined child of the subject use case. The line content shall be a compound built of two components. The first component shall be the use case identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the use case name. The order of listing shall be the numeric order of the child use case identifiers.

If the subject use cases has one or more actions, the following line shall be used:

L3 Involves actions by: <Actor name>

This line shall be repeated once for each defined actor that takes an action in the subject use case. The line content shall be the NameString that is the name of the actor. The order of listing shall be the alphabetic order of the Actor names.

If the subject use cases has derived trigger events, the following line may be used to list the events that are derived:

L3 Derives events: <Trigger event identifier: Trigger event name>

The order of listing shall be the sequence of the trigger event identifiers. The line content shall be a compound built of two components. The first component shall be the trigger event identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the trigger event name.

7.2.3.3 Interaction model

If the model contains definitions for trigger events, application profiles and or interactions, then the interaction model definitions section shall be the third section in the literary expression. It opens with:

L0 Interaction model for: <Model name>

The name of the model shall be expressed as a NameString.

7.2.3.3.1 Trigger events

If the model includes one or more trigger events, this subsection shall be used. The first line of this subsection shall be:

L1 Trigger events in: <Model name>

The name of the model shall be expressed as a NameString. Following the opening line there shall be one sub-subsection for each trigger event in the model, listed in alphabetic order by the trigger event name. The sub-subsection shall open with:

L2 Trigger event: <Trigger event identifier: Trigger event name>

The line content shall be a compound built of two components. The first component shall be the trigger event identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the trigger event name.
In the remainder of this subclause the phrase "subject trigger event" will be used to refer to the trigger event being defined. The identification line shall be followed the following line:

L3 Derives from: <Use case identifier: Use case name>

The line identifies the use cases from which the subject trigger event derives. The line content shall be a compound built of two components. The first component shall be the use case identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the use case name. This line shall be repeated once for every use case from which the subject trigger event derives, listed in order of the use case identifiers.

If a description is provided for the subject trigger event, the following two lines shall be used:

L3 Description of: <Trigger event name>

The line content is the name of the subject trigger event expressed as a NameString. It shall be followed by:

L4 <Trigger event description>

The line content is the text that describes the subject trigger event expressed as DescriptiveText. If a type is defined for the subject trigger event, the following two lines shall be used:

L3 Event type: <Trigger event type>

The line content shall be the text string that defines the type of the subject trigger event. If a dependency is defined for the subject trigger event, the following two lines shall be used:

L3 Dependency of: <Trigger event name>

The line content is the name of the subject trigger event expressed as a NameString. It shall be followed by:

L4 <Trigger event dependency>

The line content is the text that defines the dependency of the subject trigger event expressed as DescriptiveText.

7.2.3.3.2 Interactions

If the model contains one or more interactions, this subsection shall be used. The first line of this subsection shall be:

L1 Interactions in: <Model name>

The name of the model shall be expressed as a NameString. Following the opening line there shall be one sub-subsection for each interaction in the model. The interaction sub-subsection may include up to seven entries for each interaction. These entries provide the identification of the interaction, its initiating trigger event, the message to be sent, the interaction description, the sending and receiving application profiles, and the receiver responsibilities. The order of listing interactions shall be sequence of the interaction identifiers. The seven entries for each interaction are specified by the next seven line patterns. The first entry shall be:

L2 Interaction: <Interaction identifier>

The line content is the interaction identifier expressed as an IdentifierString. This line shall be followed by the second of the seven interaction entries:

L3 Initiated by: <Trigger event identifier: Trigger event name>

The line identifies the trigger event that initiates or triggers the interaction. The line content shall be a compound built of two components. The first component shall be the trigger event identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the trigger event name. This line shall be followed by the fourth of the seven interaction entries:

L3 Sender: <Application profile identifier: Application profile name>

The line identifies the application profile that is sends or initiates the interaction. The line content shall be a compound built of two components. The first component shall be the application profile identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the application profile name. This line shall be followed by the fifth of the seven interaction entries:

L3 Receiver: <Application profile identifier: Application profile name>

The line identifies the application profile that is the receiver of the interaction. The line content shall be a compound built of two components. The first component shall be the application profile identifier,

expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the application profile name.
If a description is provided for the interaction, the sixth of the seven interaction entries shall occur:

L3 Description: <Interaction description>

The line content is the description of the interaction expressed as DescriptiveText. If a receiver responsibility is provided for the interaction, the seventh and final interaction entry shall occur:

L3 Receiver responsibility: <Receiver responsibility description>

The line content is the receiver responsibility for the interaction expressed as DescriptiveText. This is the last of the entries that make up an interaction sub-subsection of a literary expression.

7.2.3.3.3 Application profiles

If the model includes one or more application profiles, this subsection shall be used. The first line of this subsection shall be:

L1 Application profiles in: <Model name>

The name of the model shall be expressed as a NameString. Following the opening line there shall be one sub-subsection for each application profile in the model, listed in the order of the application profile identifier. The sub-subsection shall open with:

L2 Application profile: <Application profile identifier: Application profile name>

The line content shall be a compound built of two components. The first component shall be the application profile identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the application profile name. In the remainder of this subclause the phrase "subject application profile" will be used to refer to the application profile being defined. If a description is provided for the subject application profile, the following two lines shall be used:

L3 Description of: <Application profile name>

The line content is the name of the subject application profile expressed as a NameString. It shall be followed by:

L4 <Application profile description>

The line content is the text that describes the subject application profile expressed as DescriptiveText.
If the subject application profile is a component of one or more composite application profiles, the following line shall be used:

L3 Is component of: <Composite application profile identifier: Composite application profile name>

The line shall be repeated once for each composite application profile of which the subject application profile is a component. The line content shall be a compound built of two elements. The first element shall be the composite application profile identifier, expressed as an IdentifierString. Following a separator, the second element shall be the NameString that expresses the composite application profile name. The order of listing shall be the order of the composite application profile identifiers.
If the subject application profile is a composed of one or more component application profiles, the following line shall be used:

L3 Is composed of: <Component application profile identifier: Component application profile name>

The line shall be repeated once for each component of which the subject application profile is composed. The line content shall be a compound built of two elements. The first element shall be the component application profile identifier, expressed as an IdentifierString. Following a separator, the second element shall be the NameString that expresses the component application profile name. The order of listing shall be the order of the component application profile identifiers.

The following lines may be used to show the interactions for which the subject application profile is a sender and receiver. This listing should be reserved for leaf level profiles, application profiles that have no components. If either of the following lines is used, both shall be used in order that the full responsibilities of the profile are expressed. The first of the optional profile responsibility lines shall be:

L3 <u>Sends interactions: </u><Interaction identifier>

The line shall be repeated once for each interaction that the subject application profile sends. The line content shall be the identifier of the sent interaction, expressed as an IdentifierString. The order shall be the sequence of the interaction identifiers. These lines shall be followed by the second of the optional profile responsibility lines:

L3 <u>Receives interactions: </u><Interaction identifier>

The line shall be repeated once for each interaction that the subject application profile sends. The line content shall be the identifier of the sent interaction, expressed as an IdentifierString. The order shall be the sequence of the interaction identifiers.

7.2.3.4 Data model

The data model class definitions section shall be the fourth section in the literary expression. It opens with:

L0 <u>Classes in:</u> <Model name>

The name of the model shall be expressed as a NameString.
Following the opening line there shall be one subsection for each class defined in this data model. The subsection for each class shall occur in alphabetic order by the class name. Each subsection shall open with:

L1 <u>Class:</u> <Class name>

The name of the class shall be expressed as a NameString.
Following the class identification line, the subsection for each class is further broken down into sub-subsections. Not all of these sub-subsections will be present for each class in the model. The specification for each of these sub-subsections along with the rules that determine when they are required are given in the next eight subclauses. Throughout these subclauses the phrase "subject class" will be used to refer to the class being defined.

7.2.3.4.1 Class summary

The sub-subsection for class summary shall be part of each class definition. The first pattern in this sub-subsection is:

L2 <u>Primary model:</u> <Parent model and class name>

This line shall be used only if the subject class has its primary definition in a model other than the one being documented. The line content shall be the fully qualified name of the class as it is known in the parent model. The class name in the model being documented should be the same as that used in the parent model, but the two names are not required to be the same. Authors may choose to copy the class definition from the parent model in order to present a complete expression of the model, but are not required to do so. All references to classes in other models are also expressed in a separate section.

L2 <u>In Subject area:</u> <Primary_subject_area name>

If the subject class is in one or more subject areas, then one of those subject areas shall be the primary subject area for the subject class, and this line shall be used. It contains the NameString that is the name of the primary subject area for the subject class.

L2 <u>Is Abstract class</u> <null>

If the subject class is an abstract class, this line shall be used. There is no other line content.

L2 Specialization of: <Generalized Class name>

If the subject class is a specialization of another class, this line shall be used. The line content is the name of the parent class expressed as a NameString.

L2 Generalization of: <Specialized class name>

If the subject class is a generalization of one or more specialized classes in a generalization-specialization relationship, this line shall be listed once for each of the specialized classes. The order of listing should be the alphabetic sequence of the specialized class names. The line content shall be the NameString that expresses the specialized class name.

Note - The following three elements of the class summary subsection are indicated as optional. In aggregate, they complete the thumb-nail view of the class being described, and thus their inclusion is encouraged. They are, however, redundant in that the same information content is included in subsequent required elements of the expression. Thus, their omission would not lead to incomplete specification of the model being expressed.

L2 Is part of: <Whole class name>

If the subject class is a part of another class in the model, this line may be used and the line content shall be the NameString that is the name of the other class of which the subject class is a part.

L2 Contains: <Part class name>

If the subject class has one or more part classes in a whole/part relationship, this line may be listed once for each of the part classes. The order of listing shall be the alphabetic sequence of the part class names. The line content shall be the NameString that expresses the part class name.

L2 Connected to: <Target class name>

If the subject class has instance connection(s) to one or more other classes in the model, this line may be listed once for each instance connection. The order of listing should be the alphabetic sequence of the names of the classes at the other (target) end of the instance connection. The line content shall be the NameString that is the name of the class at the other (target) end of the instance connection.

L2 Description of: <Class name>

If a description is provided for the subject class, this line shall be used. The line content is the name of the subject class expressed as a NameString.

L3 <Class description>

If a description is provided for the subject class, the text that describes the subject class shall follow expressed as DescriptiveText.

There are no further elements in the class Summary sub-subsection.

7.2.3.4.2 Generalization-specialization

If the subject class participates in a generalization-specialization connection with another class, and if there is a description provided for that connection, this sub-subsection shall be placed in the document starting with the following line:

L2 Gen-Spec description for: <Class name>

The line content is the name of the subject class expressed as a NameString. This line shall be followed by:

L3 <Gen-spec description>

The text of the description for the generalization specialization connection shall follow expressed as DescriptiveText.

7.2.3.4.3 Whole

If the subject class is a part of another class, the whole sub-subsection shall be placed in the document starting with the following line:

L2 Whole/part for: <Class name>

The line content is the name of the subject class expressed as a NameString. This line shall be followed by two more lines to describe the relationship:

L3 <Part to whole reference phrase (1,1) :: Whole class name ::
Whole to part reference phrase and cardinality>

The line content shall be a compound built of five components. The first component shall be the reference phrase from the perspective of the part class, if such a phrase has been defined. Otherwise, the default reference phase "is_part_of" shall be used. Following a space as the separator, the second component shall be the default cardinality for the whole "(1,1)". Following a separator, the third component shall be the NameString that is the name of the class of which the subject class is a part. Following a separator, the fourth component shall be the inverse phrase from the perspective of the whole class, if such has been defined. Otherwise, the default phrase "contains" shall be used. Following a space as the separator, the fifth component shall be the cardinality of the part relationship expressed as a CardinalityString.

L4 <Whole/part description>

If a description is provided for the whole/part relationship, the text of the whole/part description shall follow expressed as DescriptiveText.

7.2.3.4.4 Parts

If the subject class is a whole (composite) element in a whole/part relationship, the parts sub-subsection shall be placed in the document starting with the following line:

L2 Whole/part for: <Class name>

The line content is the name of the subject class expressed as a NameString. Following this opening line, the parts sub-subsection shall include a pair of entries for each of the classes that are parts of the subject class. The order of listing shall be the alphabetic sequence of the part class names. The pair of entries are specified by the next two line patterns.

L3 <Whole to part reference phrase and cardinality :: Part class name ::
Part to whole reference phrase (1,1)>

The line content shall be a compound built of five components. The first component shall be the reference phrase from the perspective of the whole class, if such a phrase has been defined. Otherwise, the default reference phase "contains" shall be used. Following a space as the separator, the second component shall be the cardinality of the part relationship expressed as a CardinalityString. Following a separator, the third component shall be the NameString that is the name of the class for which the subject class is a whole. Following a separator, the fourth component shall be the inverse phrase from the perspective of the part class, if such has been defined. Otherwise, the default phrase "is_part_of" shall be used. Following a space as the separator, the fifth component shall be the default cardinality for the whole "(1,1)".

L4 <Whole/part description>

If a description is provided for the whole/part relationship specified in the preceding line, the text of the description shall follow expressed as DescriptiveText. The description is the second of the two line patterns that are used for each part in the subject class.

7.2.3.4.5 Instance connections

If the subject class partakes in one or more instance connections, this sub-section shall appear and shall begin with:

L2 Connections for: <Class name>

The line content is the name of the subject class expressed as a NameString. Following this opening line, the connections sub-subsection shall include a pair of entries for each of the instance connections in which the subject class participates. The order of listing shall be the alphabetic sequence of the target class names. The pair of entries for each connection are specified by the next two line patterns.

L3 <Reference phrase and cardinality :: Target class name :: Inverse phrase and cardinality>

The line content shall be a compound built of five components. The "target class" is the class to which the subject class is connected. The first component shall be the reference phase for the instance connection when read from the subject class to the target class. Following a space as the separator, the second component shall be the cardinality for the target class in this connection expressed as a CardinalityString. Following a separator, the third component shall be the NameString that is the name of the target class. Following a separator, the fourth component shall be the inverse phrase used to describe the returning connection from the target class to the subject class. Following a space as the separator, the fifth component shall be the cardinality of the subject class in this connection expressed as a CardinalityString. Note that this phrase is designed to be read in a circular fashion starting and ending with the name of the subject class which is contained on the preceding line. This will result in a crude sentence of the form "Subject_class connects_to (1,m) :: Target_class :: connects_back_to (1,1) Subject_class". A similar phrase construct is used in the graphical expression.

L4 <Instance connection description>

If a description is provided for the instance connection, the text of that description shall follow expressed as DescriptiveText. The description is the second of the two line patterns that are used for each instance connection in the subject class.

7.2.3.4.6 Attributes

If the subject class contains one or more attributes, this sub-subsection shall be used, and it shall start with:

L2 Attributes of: <Class name>

The line content is the name of the subject class expressed as a NameString. Following this opening line, the attributes sub-subsection shall include a pair of entries for each attribute. The order of listing shall be alphabetic sequence of the attribute names. The pair of entries are specified by the next two line patterns.

L3 <name/DatatypeString>

The line content is a compound string that combines the name of the attribute and the DatatypeString that is the domain specification. The first element in the compound string shall be the attribute name expressed as a NameString. This shall be followed by a separator. The second element shall be the DatatypeString that is the domain specification.

L4 <Attribute description>

If a description is provided for the attribute, the text of the description shall follow expressed as DescriptiveText. The description is the second of the two line patterns that are used for each attribute in the subject class.

7.2.3.4.7 Unique attribute sets

If the subject class contains the specification for one or more unique attribute sets, this sub-subsection shall appear, and it shall start with:

L2 Unique attribute sets for: <Class name>

The line content is the name of the subject class expressed as a NameString. Following this opening line, each attribute set in the subject class will be expressed with its identification number and the list of attributes within the set. The order of listing shall be the numeric sequence of the attribute set identification numbers. The expression of the entries is specified by the following two patterns.

L3 Unique set: <Attribute_set ID>

The line content is the integer that is the identification number for the set.

L4 <Attribute name>

One line containing the NameString for an attribute name shall be listed for each attribute in the set. The order of listing shall be the alphabetic order of the attribute names.

7.2.3.4.8 States

If one or more states are defined for the subject class, the states sub-subsection shall be placed in the document starting with the following line:

L2 States for: <Class name>

The line content is the name of the subject class expressed as a NameString. Following the opening line, the states sub-subsection shall consist of an entry for each of the states listed in any order. Each state entry will be built from the following four patterns:

L3 State: <State name>

The line content shall be the NameString that is the state name.

L4 <State description>

If a description is provided for this state, the text of that description shall follow expressed as DescriptiveText.

L4 Predicate: <Predicate text>

The text that is the predicate for the state shall be line content.

L4 Condition on: <Attribute name : ConditionString>

If conditions (limitations) on attributes of the subject class are defined for the state, this line shall be repeated once for each attribute condition, listed in alphabetic sequence by the names of the conditioned attributes. The line content shall be the NameString that expresses the attribute name followed by the ConditionString that represents the attribute condition.

NOTES

1 - The JWG-CDM acknowledges that semantics by which the attribute condition should be expressed are needed and have not been defined. This issue will be deferred for resolution in later versions of the standard.

2 - In subsequent versions, the JWG-CDM will introduce condition statements for instance connections. These will be similar to the conditions on attributes above.

3 - The JWG-CDM considered the specification of state transition tables as part of this standard. This specification was not sufficiently developed and tested to include in the normative material of this version of the standard. A preliminary recommendation for this representation is included in annex B.

7.2.3.4.9 Services

If one or more services are defined for the subject class, the services sub-subsection shall be placed in the document starting with the following line:

L2 Services for: <Class name>

The line content is the name of the subject class expressed as a NameString. Following the opening line, the services sub-subsection shall consist of an entry for each of the services listed in alphabetic sequence by the service name. Each service entry will be built from the following three patterns:

L3 Service: <Service name>

The line content shall be the name of the service expressed as a NameString.

L4 <Service description>

If a description is provided for the service, the text of that description shall follow expressed as DescriptiveText.

L4 Affected attribute: <Attribute name>

This line shall be repeated once for each affected attribute, and the attributes shall be listed in alphabetic order by their names. The line content is the NameString that is the name of the affected attribute.

L4 Affected connection: <Subject Class :: reference phrase :: Target class>

This line shall be repeated once for each affected instance connection. The listing order is alphabetic by target class. The line content shall be built of three components. The "target class" is the class to which the subject class is connected. The first component shall be the NameString name of the subject class. Following a separator, the second component shall be the reference phase for the affected connection when read from the subject class to the target class. Following a separator, the third component shall be the NameString that is the name of the target class.

L4 Accessed attribute: <Class and Attribute name>

This line provides the references to any attributes whose values are accessed by the service in order to complete the service's activity. This line shall be repeated once for each accessed attribute. Since these attributes will not usually be in the subject class, the line content shall be the qualified name of the attribute that combines the name of the attribute with the name of its class. The attributes shall be listed in alphabetic order by their qualified names.

7.2.3.5 Datatype specifications

The datatype definitions section shall be the fifth section in the document. It opens with:

L0 Defined datatypes in: <Model name>

The name of the model shall be expressed as a NameString.

Following the opening line there shall be one subsection for each datatype defined in the data model. Although Clause 5 describes several kinds of datatypes that may be used in specifying attributes, not all of these kinds of datatypes need to be formally defined in this section. Specifically, the primitive datatypes (Boolean, Real, Integer, and String) and the List datatype are specified directly with the attributes in question and need no further definition here. Thus the kinds of datatypes that must be defined in this section of the literary expression are Enumerated, Compound, and Constrained datatypes.

The subsection for each datatype shall occur in alphabetic order by the datatype name. Each subsection shall open with:

L1 Datatype: <Datatype name>

The name of the datatype shall be expressed as a NameString. Throughout this subclause and its own subclauses that define the patterns for the definition of a particular datatype, the phrase "subject datatype" will be used to refer to the datatype being defined.

The first pattern in this subsection is:

L2 Primary model: <Parent model name>

This line shall be used only if the subject datatype has its primary definition in a model other than the one being documented. The line content shall be the fully qualified name of the datatype as it is known in the parent model. The datatype name in the model being documented should be the same as that used in the parent model, but the two names are not required to be the same. Authors may choose to copy the datatype definition from the parent model in order to present a complete expression of the model, but are not required to do so. All references to datatypes in other models are also expressed in a separate section.

The next line of the subsection shall be:

L2 Maintained by: <Maintaining organization / last modified date>

The line content shall be the maintaining organization name for the subject datatype followed by the last modified date of the definition expressed as a Date.

L3 <Datatype description>

If a description is provided for the subject datatype, the text that describes the subject datatype shall follow expressed as DescriptiveText.

The remaining elements in the datatype definition subsection are dependent upon the kind of datatype being defined. Each of the following three subclauses provides the specification for one of these kinds.

7.2.3.5.1 Enumerated datatype

Enumerated datatypes may either be defined within the expression of the model, or defined by reference to an external publishing authority.

7.2.3.5.1.1 Defined within the model

If the subject datatype is an enumerated datatype that is defined within the model, the next line in the definition of the subject datatype shall be:

L2 Enumerated :: <DatatypeString>

The line content shall be the DatatypeString that specifies the domain for the enumerated datatype value that is used to encode the subject datatype. This DatatypeString shall be further constrained in that it may only contain one of the four basic types (Boolean, real, integer, or string).

L2 Responsible: <Responsible organization>

The line content is the text that identifies the organization or entity that has maintenance responsibility for the value table. This may or may not be the same as the maintaining organization for this datatype definition.

The next line in the subsection should be a table header line defined by the following pattern:

L2 { Value { Name } Description } <null>

If it is used, this line has no content. If preferred, the vertical bar character (|) may be used in place of the curly brackets ({}) in the pattern above.

Following the opening lines, the subsection shall include an entry for each defined value specified for the subject datatype being enumerated. These entries are specified by the line pattern:

L2 <Value/Name/Description>

The line content is a compound string that fully defines each enumerated datatype element. For each element, it combines the value of the element, the name of the element, and the description of the element. The compound string shall have three segments. A left curly bracket ({) or a vertical bar (|) shall precede the first segment. The first segment shall contain the value of the element. A left curly bracket ({) or a vertical bar (|) shall follow the first segment and precede the second segment. The second segment of the compound string shall be either the element name, if one is provided, or one or more spaces if there is no name. A right curly bracket (}) or a vertical bar (|) shall follow the second segment and precede the third segment. The third segment of the compound string shall be either the element description, if one is provided, or one or more spaces if there is no description. The element description may contain more than one paragraph. A right curly bracket (}) or a vertical bar (|) shall follow the third segment of the compound string.

The set of lines constitutes a table of definitions for the enumerated datatype. Authors should not mix the use of vertical bars and curly brackets as separators within a single table. As can be seen in annex C, the segments of these strings can be spaced so as to appear as an aligned table in the literary expression. The table entry for the last element of the enumerated datatype ends the datatype definition subsection for a particular enumerated datatype.

7.2.3.5.1.2 Referenced to external authority

If the subject datatype is an enumerated datatype that is referenced rather than being defined within the model, the next line in the definition of the subject datatype shall be:

L2 Enumerated :: <DatatypeString>

The line content shall be the DatatypeString that specifies the domain for the enumerated datatype value that is used to encode the subject datatype. This DatatypeString shall be further constrained in that it may only contain one of the four basic types (Boolean, real, integer, or string).

L2 Reference source: <Maintaining organization>

The line content is the text that identifies the organization or entity that has maintenance responsibility for the value table. This may or may not be the same as the maintaining organization for this datatype definition. This line shall be followed by:

L3 <Reference description>

The line content shall be an explicit reference to the enumerated table that is published or maintained elsewhere. The description shall include explicit information as to which portions of the referenced table correspond to the value, name and description fields that would be part of an internally defined enumerated datatype.

7.2.3.5.2 Compound datatype

If the subject datatype is a compound datatype, the next line in the definition of the subject datatype shall be:

L2 <u>Compound datatype has parts:</u> <null>

There is no line content for this line. Following this opening line, the subsection shall include a pair of entries for each compound datatype component for the subject datatype. The components shall be listed according to their preferred order of presentation. The pair of entries are specified by the next two line patterns.

L3 <name/DatatypeString>

The line content is a compound string that combines the name of the compound datatype part and the DatatypeString that is the domain specification of the part. The first element in the compound string shall be the compound datatype part name expressed as a NameString. This shall be followed by a separator. The second element shall be the DatatypeString that is the domain specification of the part.

L4 <Compound_datatype_part description>

If a description is provided for the compound datatype part, the text of the description shall follow expressed as DescriptiveText. The description is the second of the two line patterns that are used for each part in the subject datatype.

The line pair for the last part of the compound datatype ends the datatype definition subsection for a particular compound datatype.

7.2.3.5.3 Constrained datatype

If the subject datatype is a constrained datatype, the next line in the definition of the subject datatype shall be:

L2 <u>Constrained ::</u> <DatatypeString>

The line content shall be the DatatypeString that is the domain specification of the nominal (unconstrained) attribute datatype of the subject datatype. This line shall be followed by:

L2 <u>Predicate statement for:</u> <Datatype name>

The line content shall be the NameString name of the subject datatype. This line shall be followed by:

L3 <Predicate statement>

The line content shall be one or more paragraphs that express the rules of constraint to be applied to the subject datatype. These paragraphs end the datatype definition subsection for a particular constrained datatype.

7.2.3.6 External references

The external references section shall be the sixth and final section in the document. This section collects all references to classes or datatypes that have their primary definition in a model other than the one being documented. All external references must be listed in this section. The classes and datatypes listed here may also be defined in earlier sections of the document, if desired. If they are listed there, the fully qualified name of the element must also be listed with that definition.

The External references section opens with:

L0 <u>External references for:</u> <Model name>

The name of the model shall be expressed as a NameString.

Following the opening line there shall be one subsection for referenced classes and one for referenced datatypes.

7.2.3.6.1 Referenced classes

L1 Referenced classes: <null>

If any external classes are referenced in the model, this line shall be used to open the referenced classes subsection. There is no line content.

L2 <External class name :: Class name>

This line shall be used for each class in the model that has its primary definition in a model other than the one being documented. The line content is a compound string combining the external reference with the internal name of the class, if it is different. The first segment of the line content shall be the fully qualified name of the class as it is known in the parent model. If a different name has been assigned to the class within the model being documented, the NameString internal name shall be the second segment of the line content with a separator in between. The order of listing shall be the alphabetic sequence of the fully qualified external names of the classes.

7.2.3.6.2 Referenced datatypes

L1 Referenced datatypes: <null>

If any external datatypes are referenced in the model, this line shall be used to open the referenced datatypes subsection. There is no line content.

L2 <External datatype name :: Datatype name>

This line shall be used for each datatype in the model that has its primary definition in a model other than the one being documented. The line content is a compound string combining the external reference with the internal name of the datatype, if it is different. The first segment of the line content shall be the fully qualified name of the datatype as it is known in the parent model. If a different name has been assigned to the datatype within the model being documented, the NameString internal name shall be the second segment of the line content with a separator in between. The order of listing shall be the alphabetic sequence of the fully qualified external names of the datatypes.

7.2.4 Graphical expression specification

The graphical expression of a data model is the pictorial representation of the most critical elements of the data model. It can express the classes in the model, the various connections between those classes, the attributes of the classes, the name of the attribute datatype for each attribute, and the services for the classes. It cannot, however, express the model scenarios, the unique attribute sets and states for the classes, nor the specification for the named datatypes defined in the model. Moreover, the graphical expression contains none of the descriptive text that may be associated with each element of the data model.

The principle role of the graphical expression is to provide an overview of the model, to make clear the important connections that exist between classes, and to provide a map by which readers can navigate through the complete specification contained in the literary expression.

The graphical expression is made up of graphic symbols for the classes and connections in the model, and of textual labels that are placed within or adjacent to those symbols. The subclauses that follow will specify the form of these symbols and the format and location of the labels. This standard cannot, however, specify the layout of these elements, and good layout is an essential feature of a useful graphical expression. Thus layout is left to the artistic skill of the modeler.

The following graphics are provided to exemplify the symbols being described in the following subclauses. The graphics do not represent a complete or even a logical model. Their sole purpose is to demonstrate each element of the graphical expression.

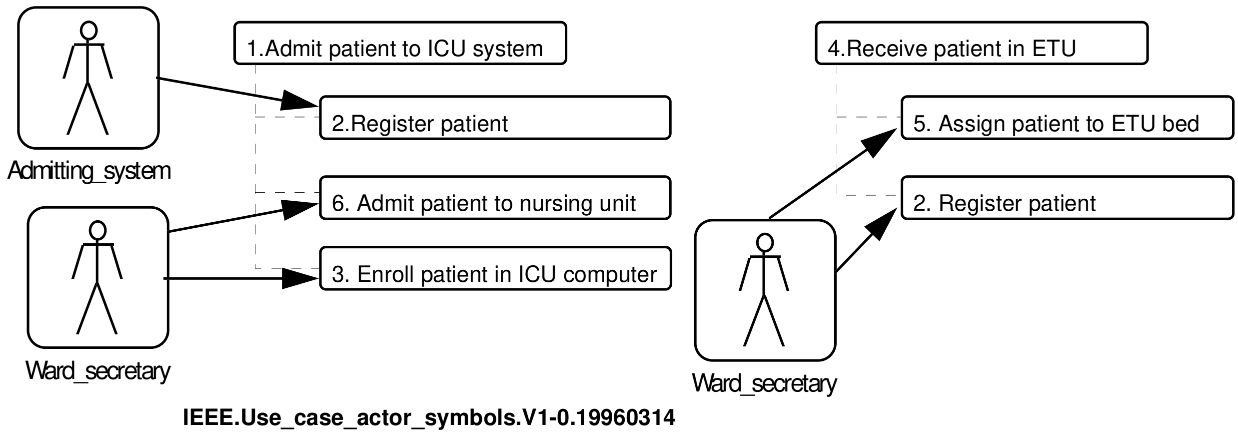


Figure 6.1 - Diagramming conventions for use cases and actors.

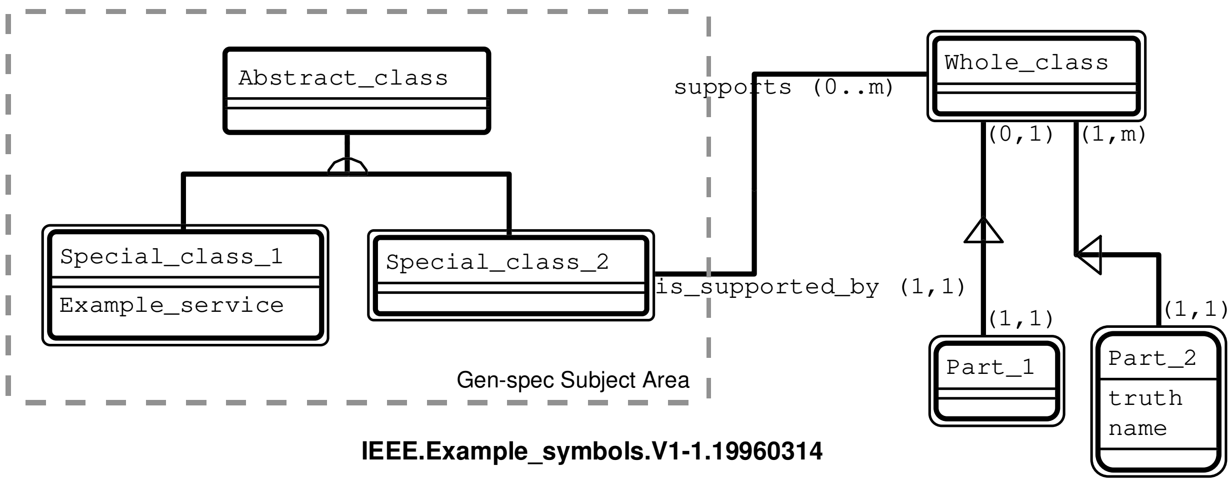


Figure 6.2 - Example of graphical expression conventions for data model

The conventions for expressing the components of a data model in the graphical symbols are specified in the following subclauses. These conventions are those documented by Coad and Yourdon [B1]. The order of presentation is the same order that was used in the literary expression.

7.2.4.1 Model identification

Each page of the graphical expression shall contain the identification of the model being represented. This identification shall be a textual label containing the fully qualified name of the model, which includes the developing organization, and the name, version, and last modified date of the model. In figure 6.2, this identification is at the bottom of the diagram.

7.2.4.2 Subject areas

Subject areas may be depicted in the graphic expression if the authors desire. Subject areas should be indicated by a gray or dashed line surrounding the symbols of the classes that are part of the subject area. An example subject area is shown on the right side of the diagram above. At times, the subject area lines confuse a drawing because they may obscure important connecting lines.

7.2.4.2.1 Actors

Actors should be diagrammed as a rectangle with a cartoon of a person in it. The NameString of the actor should be displayed immediately adjacent to the actor. Actions by an actor in one or more use cases should be indicated by an arrow drawn from the actor to the use case. If actions are shown, at least one action must be shown for each bottom level use case in the diagrammed hierarchies.

7.2.4.2.2 Use cases

Use cases should be diagrammed as rounded rectangular boxes that surround the use case label. The label should be the use case identifier followed by the use case title. Action by an actor in one or more use cases should be indicated by an arrow drawn from the actor to the use case.

The parent child relationships between use cases should be indicated by an indented list structure in which the children are shown below and indented beneath the parent. (See example in figure 6.1.) Linkage lines between the parent and children may also be used as shown above.

A use case may be placed in the diagram more than once. This is useful if one use case is the child of several parents.

7.2.4.2.3 Interactions

In this version of the manual, HL7 has not yet established a recommended diagramming convention for representing interactions. However, a tabular format is recommended for summarizing the interactions that exist in a model. This table will assist readers in recognizing the relationships between trigger events, application profiles and interactions, and will assist in navigating the literary expression in order to find the detailed specification of a particular interaction. The recommended tabular expression follows.

The table should be comprised of a repeating set of elements drawn from the model. Whether these elements are the columns of a table with one row for each interaction, or the rows of a table with one column for each interaction is up to the modeler. The elements that should be used are listed below along with a recommended sequence. Where table elements have both a unique identifier and a unique name, the name is an optional element.

The recommended elements are:

Table element	Comment
Interaction.identifier	Identifies interaction
Message.identifier	Message sent
Message.name (optional)	
Trigger_event.identifier	Event that initiates
Trigger_event.name (optional)	
Trigger_event.dependency	Dependency of initiating event (summarized)
Application_profile.identifier	Sending profile
Application_profile.name	Sender
Application_profile.identifier	Receiving profile
Application_profile.name	Receiver
Interaction.receiver_responsibility	

The sorting sequence for the table should be either of the following:

Trigger_event.identifier : Interaction.identifier

Interaction.identifier : Trigger_event.identifier

7.2.4.3 Classes

The classes in the model may be represented in the graphical expression by its own symbol. The class symbol is a rounded rectangle nested inside a similar rectangle. The inner rectangle should be drawn with a bolder line than the outer rectangle. The inner rectangle represents the class and terminates all connections that apply to every instance of the class, while the outer rectangle represents the instances of the class and terminates the connections that apply only to specific instances of the class.

The class symbol shall be divided into three sections by horizontal lines. The NameString that is the name of the class shall be placed in the upper-most section of the symbol.

An abstract class is one that will not be instantiated. Therefore it is drawn without the outer rectangle which represents the instances of a class. Figure 4 contains an abstract class in the upper left, and five classes that can be instantiated.

7.2.4.4 Generalization-specialization connections

The generalization-specialization structure is a connection between a more general class and one or more specializations of that class. This connection shall be drawn as a line between the bottom of the general class rectangle (inner rectangle) and the top of the specialized class rectangle (inner rectangle). A semicircle shall be drawn somewhere along the line such that the line to the general class connects to the upper, circular portion of the semicircle, while the line(s) to the specialized class(es) connect to the flat bottom of the semicircle. The generalization-specialization connection is not labeled.

The three classes on the left of figure 4 are connected by a generalization-specialization structure in which the abstract class is the general class.

7.2.4.5 Whole-part connections

Whole-part connections are connections between the instances of a whole class and the instances of its parts. Each part class shall be connected to its whole class by a connection from the top of the instance (outer) rectangle of the part to the bottom of the instance (outer) rectangle of the whole. Somewhere along the line shall be placed a triangle with the line to the whole connecting to the apex of the triangle and the line to the part connecting to the base of the triangle. Each part shall have its own connecting line to the whole.

The whole-part connection shall be labeled with the cardinality of the relationship. The CardinalityString that represents the cardinality of the part shall be placed near where the connection line meets the whole. The cardinality of the whole is always expressed by the CardinalityString "(1,1)". This string should be placed near where the connection meets the part.

The three classes on the right of figure 4 participate in a whole-part structure.

7.2.4.6 Instance connections

Instance connections express a relationship between the instances of two classes. Instance connections shall be drawn from one side of the instance (outer) rectangle of one of the classes to a side of the instance (outer) rectangle of the other.

The instance connection line shall be labeled with the cardinalities of the connection and should also be labeled with the reference phrases. The labels should be placed near the connection points at either end of the line. The labels are built of two parts. Each label should start with the reference phrase that expresses the nature of the relationship from the perspective of the class that is adjacent to the label. Following the phrase should be the cardinality of the class <u>at the other end</u> of the connection.

The location of the cardinality is that recommended by [B1]. Although it is the reverse of what is normally done in entity-relationship diagrams, it provides an easily read diagram. An instance connection is drawn between the Special_Class_2 and Whole_Class in the example. Starting from Special_Class_2 this connection can be read as "Special_Class_2 is_supported_by one and only one Whole_Class." Starting from the other direction, this connection can be read as "Whole_Class may support many Special_Class_2."

7.2.4.7 Attributes

The attributes of each class may be listed by including their NameStrings in the center section of the class rectangle.

The Part_2 class in figure 4 shows two attributes.

7.2.4.8 Services

The services of each class that has defined services may be listed in the bottom-most section of the class rectangle. These shall be listed as the NameString name of the service.

Special_Class_1 has an example of a service.

7.2.4.9 Compound Datatypes

Compound datatypes are a structured datatype made up of multiple parts each of which has its own set of datatype constraints. These may be represented graphically along with the elements of the data models they apply to. If some of the compound datatypes for a model are represented graphically, all of them should be.

Compound datatypes shall be contained within a subject area named "Compound Datatypes." Each compound datatype shall be represented using the same symbol as that specified for an abstract class. (See 6.4.3.) This symbol is a rectangle divided into three sections by horizontal lines. The NameString that is the name of the compound datatype shall be placed in the upper-most section of the symbol.

The names of the compound datatype components that are the parts of the compound datatype may be represented by their NameStrings in the center section of the compound datatype rectangle.

Although they are diagrammed using class symbols, the compound datatypes shall not diagrammed as: being structured in generalization-specialization structures; being connected by whole-part or instance connection relationships; or possessing services, unique attribute sets or states.

7.3 HL7 Use case and interaction models

The Joint Working Group for a Common Data Model (JWG-CDM) has extended its draft standard for data models by including an annex of recommended practices. These extensions are primarily of use in modeling system interactions, trigger events and transactions. These elements are critical to the expression of the models needed to support the generation of version 3 standards in HL7.

This document has been adapted from annex B of the JWG-CDM draft standard. It is fully compliant with the intent of that annex, but certain model components have been renamed to be more consistent with traditional HL7 usage.

This annex provides: a brief rationale for including these components in a set of models; definition of the recommended model components; specification for the representation of these components; and extensions to represent the recommended components as part of the formal representation of the meta-model.

7.3.1 Rationale for recommended practices

The following model components are included in the recommended practices -- use cases, actors, trigger events, messages, and application roles. Each of these elements has been found useful in modeling needed to support the development of healthcare information interchange standards.

Note: The JWG-CDM recommended practices use the term "business event" instead of "trigger event" and "interaction roles" instead of "application roles." Further, the JWG-CDM recommended practices identify the notion of an "interaction" as distinct from a "message." An interaction is an association between one event and one message. This allows for a message to be used in multiple interactions. The HL7 Version 3 Task Force has elected to say that each message is used in only one interaction, and thus has collapsed the interaction and message concepts into the interaction concept described here, where the message name is an attribute of the interaction. Finally, the JWG-CDM also includes the option of aggregating interactions into interaction sets. This option has not been incorporated for use by HL7.

7.3.1.1 Use cases and actors

As HL7 has approached the revision of its development process for Version 3, it has found it necessary to represent the users who interact with the system, and the instances of information interchange that these users mediate. It has found that these requirements are readily represented by the notions of use cases and actors that have been introduced into object oriented analysis by Ivar Jacobson, *et al.*[B3]. For this reason, the use cases, actors, and the relationships between them are part of the recommended practices.

The use case models provide the statement of scope and functional requirements that are necessary to both develop the classes, attributes, connections and structures in the data model, and to specify the events that cause information to be exchanged and the messages that flow as a result of such events, Thus the use case and actor model feeds the more detailed modeling of the classes and interactions.

7.3.1.2 Trigger events and interactions

The developers of information interchange standards inevitably must identify the interactions between systems in which the interchange occurs. Specification of an interchange includes the identification of: the events that cause information to move, the roles - either individual or system - that are the senders and receivers of the interchange, and the information that is interchanged. In the X12N business narratives, the initiator of an interaction is defined, along with the nature of that interaction. Similarly, in HL7, a trigger event is defined along with the messages to be sent when that event occurs. In ASTM, the trigger event defines the situation when information is captured for storage. .Both HL7 and X12N have a concept of a sender and a receiver as well as of the message that moves.

These recommended practices include models of these concepts. The events are termed "trigger events" and the interaction content is expressed as messages. The former is carried as a textual description of the event. Similarly, there is not a detailed definition of the messages, but rather a description of their general content, their relationship to the events and to other messages.

7.3.2 Definition of recommended practices components

The following subclauses provide definitions for the recommended components. The following table lists the title for each subclause as a guide to finding a particular definition.

7.3.2.1 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. Actors include both humans and computer systems. Each kind of actor interacts in a different way, otherwise the actors would not be different. Each way of interacting with the health care system is called a use case. Any particular user may fulfill several roles, and thus perform as several different actors. Information kept about actors includes:

7.3.2.1.1 Name

The actors in the data model are each given a unique name. The actor name is a singular noun or noun phrase.

7.3.2.1.2 Description

A short informative statement that allows people to determine, with certainty, whether a particular real world role is an instance of the actor as conceptualized in the data model.

7.3.2.1.3 Reference to parent model

A reference to the model in which this actor is formally defined and maintained.

7.3.2.2 Use Case

A use case is a generic description of system usage, where system refers in the broadest sense of the healthcare system. A use case describes the possible sequences of interactions among the components of the system and one or more actors in response to some initial stimulus by one of the actors or by a change in the state of an element of the system.

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 data model, related business models, and information interchange 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 one or more trigger events that initiate interactions. If the linkage is to multiple such events, the events should be tightly coupled so that they usually occur together.

Information kept about a use case includes:

7.3.2.2.1 Identifier

An identifier assigned to the use case. The identifier is unique within the scope of the model in which the use case is defined.

7.3.2.2.2 Title

A short phrase that provides a descriptive title for the use case. The title shall be unique within the scope of the model in which the use case is defined.

7.3.2.2.3 Description

The text that describes the use case and provides the details necessary to understand the interactions, transactions and events that are involved in the use case.

7.3.2.2.4 Reference to parent model

A reference to the model in which this use case is formally defined and maintained.

7.3.2.2.5 List of child use case references

An optional reference to one or more other use cases that are children of the use case. The interactions ascribed to all of the children constitute the complete interaction of the parent.

7.3.2.3 Use case participant

An association between use cases and actors that links the actors who participate in a particular use case to that use case. Information kept about a use case participant includes:

7.3.2.3.1 Description

The text that describes the use case participant and provides the details necessary to understand the association.

7.3.2.3.2 Reference to use case

A reference to the use case in which the participant participates.

7.3.2.3.3 Reference to actor

A reference to the actor that is the participant.

7.3.2.4 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. This is also known as the initiator of an interaction or a business event. Each trigger event is linked to one leaf level use case that defines the context from which the trigger event is derived.

The occurrence of a trigger event may also be dependent upon the state of one or more objects in the domain or upon the prior occurrence of a different trigger event. Such dependencies allow the definition of a sequence of events and interactions that constitute a transaction.

Information kept about a trigger event includes:

7.3.2.4.1 Identifier

An identifier assigned to the trigger event. The identifier is unique within the scope of the model in which the trigger event is defined.

7.3.2.4.2 Name

A name assigned to the trigger event. The name is unique within the scope of the model in which the trigger event is defined.

7.3.2.4.3 Description

The text that describes the trigger event. When viewed along with the description of the use case from which the event derives, this description must have sufficient detail that the event can be reliably recognized when it occurs.

7.3.2.4.4 Dependency

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.

7.3.2.4.5 Reference to parent model

A reference to the model in which the trigger event is formally defined and maintained.

7.3.2.4.6 Reference to defining use case

A reference to the leaf level use case that provides the context from which the trigger event is derived..

7.3.2.5 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.

Note - The requirement stated above that "an interaction may not specify a return message" does not preclude the return of a "message received" acknowledgment that serves the purpose of providing for guaranteed delivery of the interaction message.

A message is the specific information transfer that occurs in an interaction to meet the requirements of use cases. This is an atomic unit in that the entire message or information content will be sent in an interaction, or no part of it will be sent. That is, no further decomposition is possible.

7.3.2.5.1 Identifier

An identifier assigned to the interaction. The identifier should be unique within the scope of the model in which the interaction is defined.

7.3.2.5.2 Name

A name assigned to the interaction and its message. The name is unique within the scope of the model in which the interaction is defined.

7.3.2.5.3 Description

The text that describes the message and the context for the interaction. The message description should include: references to the classes in the data model that are the subject of the message; and references to the use cases and actors whose requirements the message fulfills. The description is intended to guide the efforts of individuals responsible for designing the message structure.

7.3.2.5.4 Receiver responsibility

The text that describes the responsibility of the receiver of the message once receipt is acknowledged. This is an optional element in that there may no follow-on responsibility. If a responsibility exists it might include explicit manipulations of objects in the domain information space, and/or the initiation of another trigger event that would trigger a subsequent interaction to occur. Transactions can be established through a chain of receiver responsibilities for individual interactions.

7.3.2.5.5 Reference to parent model

A reference to the model in which the interaction is formally defined and maintained.

7.3.2.5.6 Reference to initiating trigger event

A reference to the trigger event that triggers or initiates this interaction.

7.3.2.5.7 Reference to sending role

A reference to the application role that is responsible for sending the message involved in this interaction.

7.3.2.5.8 Reference to receiving role

A reference to the application role that is responsible for receiving the message involved in this interaction.

7.3.2.6 Application role

A defined set of responsibilities that a system or an individual holds with respect to interactions. The role may have responsibility to send or receive one or more interaction messages. The application role and its responsibilities may form the basis for establishing conformance specifications for a standard.

If an application role is identified as the sender of an interaction message, it has responsibilities to recognize the trigger event that initiates the interaction, and to cause the appropriate message or information content to be sent to the receiver.

If an application role is identified as a receiver of an interaction message, it must be prepared to accept the message and to fulfill the further responsibilities defined as the receiver responsibility in the definition of the interaction.

Application roles may be aggregated into higher level, composite application roles. In the case of such aggregation, the composite application role must undertake all of the responsibilities of the lower level roles that compose it. Note that an application role need not be part of any composite, and may be part of more than one composite.

Information kept about an application role includes:

7.3.2.6.1 Identifier

An identifier assigned to the application role. The identifier is unique within the scope of the model in which the application role is defined.

7.3.2.6.2 Name

A name assigned to the application role. The name is unique within the scope of the model in which the application role is defined.

7.3.2.6.3 Description

The text that describes the application role and summarizes the interaction responsibilities that are part of that role.

7.3.2.6.4 Reference to parent model

A reference to the model in which this application role is formally defined and maintained.

7.3.2.6.5 List of component application role references

An optional reference to one or more other application roles that compose a composite application role. The interaction responsibilities of a composite role are the aggregate of the individual responsibilities undertaken by the component roles that make up the composite.

7.3.3 Expression of recommended model components

The following paragraphs provide specifications for expressing the recommended model components in the literary and graphical expressions of a model. The format used here is compatible with that in clause 6 of the JWG-CDM draft standard.

7.3.3.1 Formal representation of unique identifiers

Various data model elements recommended in this annex are required to have identifiers that are unique throughout the model, as specified in subsequent clauses. These elements will be specified as being represented by an IdentifierString.

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, as discussed in clause 6.2.1.2. However, each of the elements that uses an IdentifierString also has another unique identifier that can be used in building fully qualified names.

7.3.3.2 Literary expression of interaction components

The subclauses that follow this one contain the specification for representing each of the interaction components in the literary expression of a data model. The following table lists the title for each subclause as a guide to finding a particular specification.

B.3.2.1 Actors
B.3.2.2 Use cases
B.3.2.3 Trigger events
B.3.2.4 Interactions
B.3.2.5 Application roles

If the model contains definitions for use cases, actors, trigger events and or interactions, then the interaction components definitions section shall be the fourth section in the literary expression. It opens with:

L0 Interaction components in: <Model name>

The name of the model shall be expressed as a NameString.

7.3.3.2.1 Actors

If the model includes one or more actors, this subsection shall be used. The first line of this subsection shall be:

L1 Actors in: <Model name>

The name of the model shall be expressed as a NameString. Following the opening line there shall be one sub-subsection for each actor in the model, listed in alphabetic order by the actor name. The sub-subsection shall open with:

L2 Actor: <Actor name>

The name of the actor shall be expressed as a NameString.
Following the actor identification line, there will be a description sub-subsection and a use case participation sub-subsection for each actor. The specification for these sub-subsections are in the remainder of this subclause. In this subclause the phrase "subject actor" will be used to refer to the actor being defined.

L3 Description of: <Actor name>

If a description is provided for the subject actor, this line shall be used. The line content is the name of the subject actor expressed as a NameString. It shall be followed by:

L4 <Actor description>

The line content is the text that describes the subject actor expressed as DescriptiveText.
If the subject actor participates in one or more use cases in a participant relationship, the following two lines shall be listed for each of the use cases. The order of listing shall be the identifier sequence of the use cases.

L3 Participates in: <Use case identifier: Use case name>

The line content of the first of the pair of lines shall be a compound built of two components. The first component shall be the use case identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the use case name. If there is a description for the participant relationship identified on the first line, then it shall be followed by the second line:

L4 <Participant description>

The line content shall be the text of the description for the participant association expressed as DescriptiveText.

7.3.3.2.2 Use cases

If the model includes one or more use cases, this subsection shall be used. The first line of this subsection shall be:

L1 Use cases in: <Model name>

The name of the model shall be expressed as a NameString. Following the opening line there shall be one sub-subsection for each use case in the model, listed in order of the use case identifier. The sub-subsection shall open with:

L2 Use case: <Use case identifier: Use case name>

The line content shall be a compound built of two components. The first component shall be the use case identifier. Following a separator, the second component shall be the NameString that expresses the use case name.
Following the use case identification line, there will be a description subsection, a heritage subsection and an actor participation subsection for each use case. The specification for these sub-subsections are in the remainder of this subclause. In this subclause the phrase "subject use case" will be used to refer to the use case being defined.

L3 Description of: <Use case name>

If a description is provided for the subject use case, this line shall be used. The line content is the name of the subject use case expressed as a NameString. It shall be followed by:

L4 <Use case description>

The line content is the text that describes the subject use case expressed as DescriptiveText.
If the subject use case is a child of one or more other (parent) use cases, the following line shall be used:

L3 Is child of: <Parent use case identifier: Parent use case name>

This line shall be repeated once for each defined parent of the subject use case. The line content shall be a compound built of two components. The first component shall be the use case identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the use case name. The order of listing shall be the numeric order of the parent use case identifiers.

If the subject use case is a parent to one or more other (child) use cases, the following line shall be used:

L3 Is parent to: <Child use case identifier: Child use case name>

This line shall be repeated once for each defined child of the subject use case. The line content shall be a compound built of two components. The first component shall be the use case identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the use case name. The order of listing shall be the numeric order of the child use case identifiers.

If the subject use cases has one or more actor participants, the following line shall be used:

L3 Has participants: <Actor name>

This line shall be repeated once for each defined actor that participates in the subject use case. The line content shall be the NameString that is the name of the actor participant. The order of listing shall be the alphabetic order of the Actor names.

If the subject use cases has derived trigger events, the following line may be used to list the events that are derived:

L3 Derives events: <Trigger event identifier: Trigger event name>

The order of listing shall be the sequence of the trigger event identifiers. The line content shall be a compound built of two components. The first component shall be the trigger event identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the trigger event name.

7.3.3.2.3 Trigger events

If the model includes one or more trigger events, this subsection shall be used. The first line of this subsection shall be:

L1 Trigger events in: <Model name>

The name of the model shall be expressed as a NameString. Following the opening line there shall be one sub-subsection for each trigger event in the model, listed in alphabetic order by the trigger event name. The sub-subsection shall open with:

L2 Trigger event: <Trigger event identifier: Trigger event name>

The line content shall be a compound built of two components. The first component shall be the trigger event identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the trigger event name.
In the remainder of this subclause the phrase "subject trigger event" will be used to refer to the trigger event being defined. The identification line shall be followed the following line:

L3 Derives from: <Use case identifier: Use case name>

The line identifies the use case from which the subject trigger event derives. The line content shall be a compound built of two components. The first component shall be the use case identifier, expressed as an

IdentifierString. Following a separator, the second component shall be the NameString that expresses the use case name.
If a description is provided for the subject trigger event, the following two lines shall be used:

L3 Description of: <Trigger event name>

The line content is the name of the subject trigger event expressed as a NameString. It shall be followed by:

L4 <Trigger event description>

The line content is the text that describes the subject trigger event expressed as DescriptiveText.
If a dependency is defined for the subject trigger event, the following two lines shall be used:

L3 Dependency of: <Trigger event name>

The line content is the name of the subject trigger event expressed as a NameString. It shall be followed by:

L4 <Trigger event dependency>

The line content is the text that defines the dependency of the subject trigger event expressed as DescriptiveText.

7.3.3.2.4 Interactions

If the model contains one or more interactions, this subsection shall be used. The first line of this subsection shall be:

L1 Interactions in: <Model name>

The name of the model shall be expressed as a NameString. Following the opening line there shall be one sub-subsection for each interaction in the model. The interaction sub-subsection may include up to six entries for each interaction. These entries provide the identification of the interaction, its initiating trigger event, the interaction description, the sending and receiving application roles, and the receiver responsibilities. The order of listing interactions shall be sequence of the interaction identifiers. The six entries for each interaction are specified by the next six line patterns. The first entry shall be:

L2 Interaction: <Interaction identifier: Interaction name>

The line content shall be a compound built of two components. The first component is the interaction identifier expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the interaction name. This line shall be followed by the second of the six interaction entries:

L3 Initiated by: <Trigger event identifier: Trigger event name>

The line identifies the trigger event that initiates or triggers the interaction. The line content shall be a compound built of two components. The first component shall be the trigger event identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the trigger event name. This line shall be followed by the third of the six interaction entries:

L3 Sender: <Application role identifier: Application role name>

The line identifies the application role that is sends or initiates the interaction. The line content shall be a compound built of two components. The first component shall be the application role identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the application role name. This line shall be followed by the fourth of the six interaction entries:

L3 Receiver: <Application role identifier: Application role name>

The line identifies the application role that is the receiver of the interaction. The line content shall be a compound built of two components. The first component shall be the application role identifier, expressed

as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the application role name.
If a description is provided for the interaction and the message involved in it, the fifth of the six interaction entries shall occur:

L3 Description: <Interaction description>

The line content is the description of the interaction expressed as DescriptiveText. If a receiver responsibility is provided for the interaction, the sixth and final interaction entry shall occur:

L3 Receiver responsibility: <Receiver responsibility description>

The line content is the receiver responsibility for the interaction expressed as DescriptiveText. This is the last of the entries that make up an interaction sub-subsection of a literary expression.

7.3.3.2.5 Application roles

If the model includes one or more application roles, this subsection shall be used. The first line of this subsection shall be:

L1 Application roles in: <Model name>

The name of the model shall be expressed as a NameString. Following the opening line there shall be one sub-subsection for each application role in the model, listed in the order of the application role identifier. The sub-subsection shall open with:

L2 Application role: <Application role identifier: Application role name>

The line content shall be a compound built of two components. The first component shall be the application role identifier, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the application role name. In the remainder of this subclause the phrase "subject application role" will be used to refer to the application role being defined. If a description is provided for the subject application role, the following two lines shall be used:

L3 Description of: <Application role name>

The line content is the name of the subject application role expressed as a NameString. It shall be followed by:

L4 <Application role description>

The line content is the text that describes the subject application role expressed as DescriptiveText.
If the subject application role is a component of one or more composite application roles, the following line shall be used:

L3 Is component of: <Composite application role identifier: Composite application role name>

The line shall be repeated once for each composite application role of which the subject application role is a component. The line content shall be a compound built of two elements. The first element shall be the composite application role identifier, expressed as an IdentifierString. Following a separator, the second element shall be the NameString that expresses the composite application role name. The order of listing shall be the order of the composite application role identifiers.
If the subject application role is a composed of one or more component application roles, the following line shall be used:

L3 Is composed of: <Component application role identifier: Component application role name>

The line shall be repeated once for each component of which the subject application role is composed. The line content shall be a compound built of two elements. The first element shall be the component application role identifier, expressed as an IdentifierString. Following a separator, the second element shall be the NameString that expresses the component application role name. The order of listing shall be the order of the component application role identifiers.

The following lines may be used to show the interactions for which the subject application role is a sender and receiver. This listing should be reserved for leaf level roles, application roles that have no components. If either of the following lines is used, both shall be used in order that the full responsibilities of the role are expressed. The first of the optional role responsibility lines shall be:

L3 <u>Sends interactions: </u><Interaction identifier: Interaction name>

The line shall be repeated once for each interaction that the subject application role sends. The line content shall be a compound built of two components. The first component is the identifier of the sent interaction, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the interaction name. The order shall be the sequence of the interaction identifiers. These lines shall be followed by the second of the optional role responsibility lines:

L3 <u>Receives interactions: </u><Interaction identifier>

The line shall be repeated once for each interaction that the subject application role sends. The line content shall be a compound built of two components. The first component is the identifier of the sent interaction, expressed as an IdentifierString. Following a separator, the second component shall be the NameString that expresses the interaction name. The order shall be the sequence of the interaction identifiers.

7.3.3.3 Graphical expression specification

If the modeler desires to provide a graphical expression of the use cases in a model, their relationship to the actors, and the parent child relationships between use cases, the following diagramming conventions should be used. These conventions are shown in the sample graphical expression in the following figure:

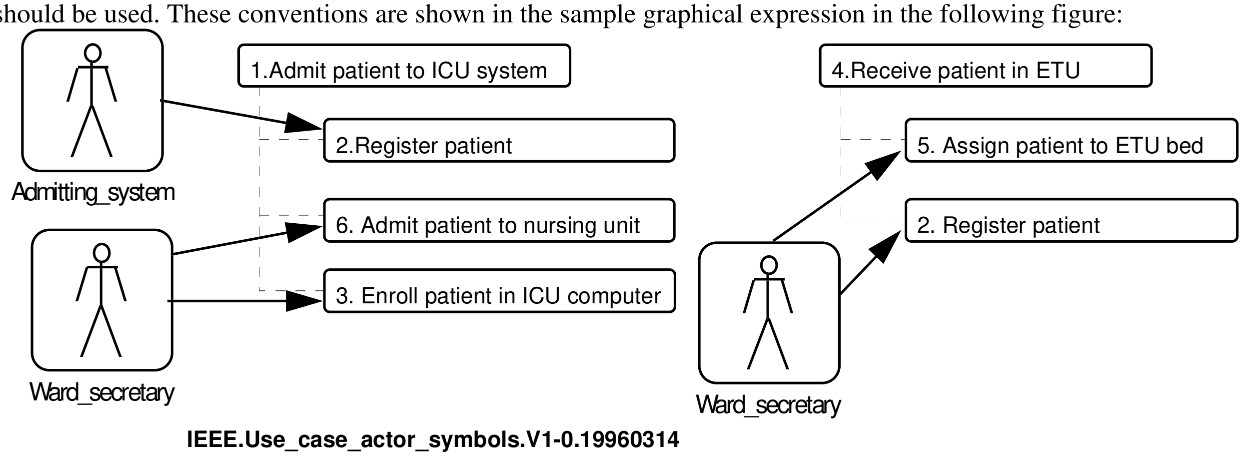


Figure B.1 - Diagramming conventions for use cases and actors.

7.3.3.3.1 Actors

Actors should be diagrammed as a rectangle with a cartoon of a person in it. The NameString of the actor should be displayed immediately adjacent to the actor. Participation by an actor in one or more use cases should be indicated by an arrow drawn from the actor to the use case. If participation is shown, at least one participant must be shown for each bottom level use case in the diagrammed hierarchies.

7.3.3.3.2 Use cases

Use cases should be diagrammed as rounded rectangular boxes that surround the use case label. The label should be the use case identifier followed by the use case title. Participation by an actor in one or more use cases should be indicated by an arrow drawn from the actor to the use case.

The parent child relationships between use cases should be indicated by an indented list structure in which the children are shown below and indented beneath the parent. (See example in figure B.1.) Linkage lines between the parent and children may also be used as shown above.

A use case may be placed in the diagram more than once. This is useful if one use case is the child of several parents.

7.3.3.3.3 Interactions

In this version of the standard, the JWG-CDM is not establishing a recommended diagramming convention for representing interactions. However, a tabular format is recommended for summarizing the interactions that exist in a model. This table will assist readers in recognizing the relationships between trigger events, application roles and interactions, and will assist in navigating the literary expression in order to find the detailed specification of a particular interaction. The recommended tabular expression follows.

The table should be comprised of a repeating set of elements drawn from the model. Whether these elements are the columns of a table with one row for each interaction, or the rows of a table with one column for each interaction is up to the modeler. The elements that should be used are listed below along with a recommended sequence. Where table elements have both a unique identifier and a unique name, the name is an optional element.

The recommended elements are:

Table element	Comment
Interaction.identifier	Identifies interaction
Interaction.name (optional)	Message sent
Business_event.identifier	Event that initiates
Business_event.name (optional)	
Business_event.dependency	Dependency of initiating event (summarized)
Interaction_role.identifier	Sending role
Interaction_role.name	Sender
Interaction_role.identifier	Receiving role
Interaction_role.name	Receiver
Interaction.receiver_responsibility	

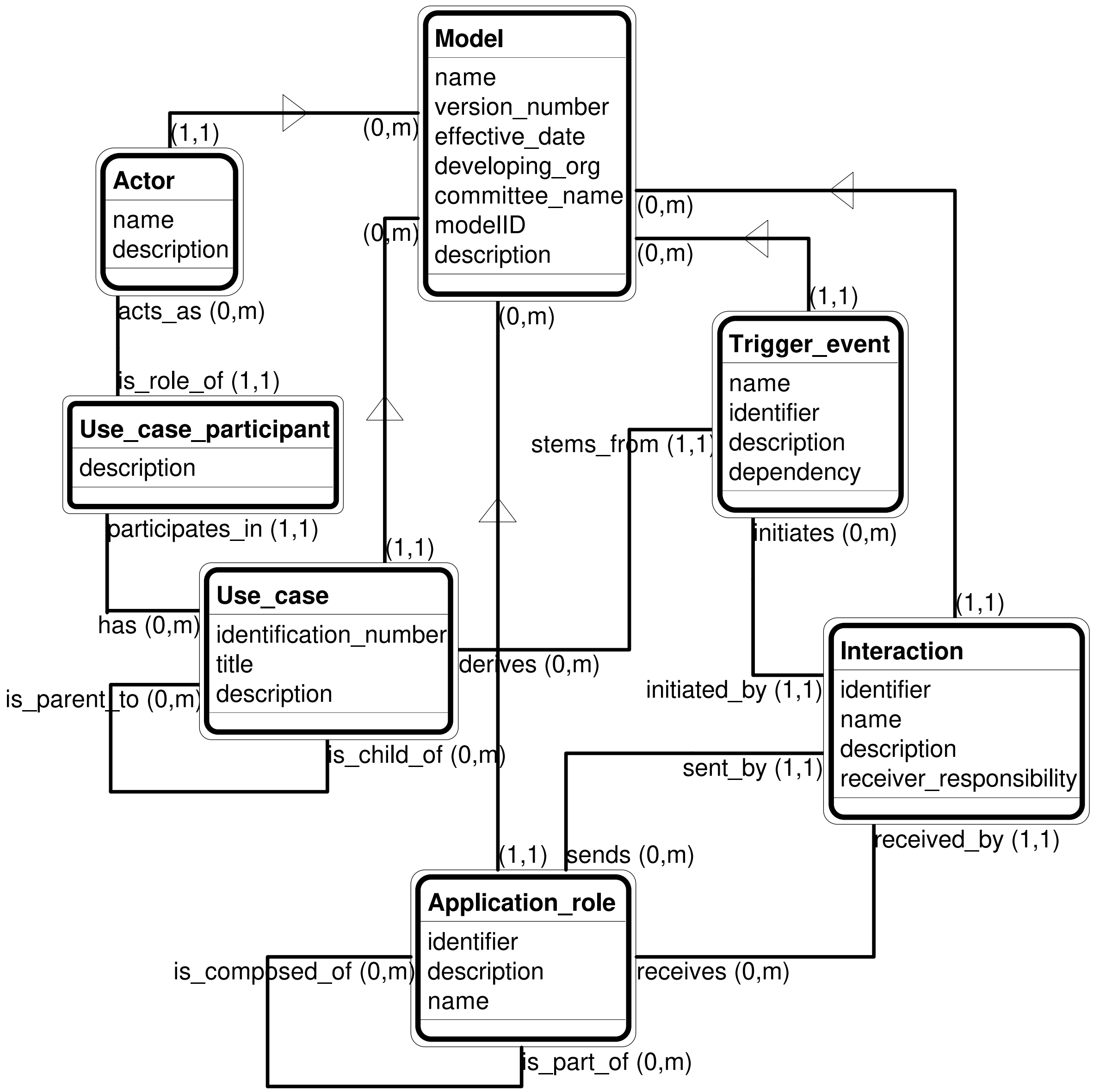
The sorting sequence for the table should be either of the following:

Business_event.identifier :Interaction.identifier

Interaction.identifier : Business_event.identifier

7.3.4 Meta model diagram

Additions to the meta-model in annex A of the JWG-CDM draft standard that represent the inclusion of the interaction components (use cases, actors, trigger events, messages, interactions, and application roles) are indicated in the figure below.



HL7.Interaction_component_meta-model.V1-0.19960331

1

I. Figure B.2 - Meta model components to represent the HL7 interaction components of a model.

1

8. Glossary

Abstraction

The logical form which analogous things have in common. Also, the process of identifying characteristics that distinguish a collection of similar objects.

Actor

An actor is a communication role played by such an entity. Actors include both humans and other quasi-autonomous things, such as machines, computer tasks, and systems. Objects that are indirectly connected to the system are not actors, since their needs must be communicated to the system through an actor.

Analysis

The specification process in which the logical form of a particular problem or subject is considered. Analysis defines only the structure of a subject and does not consider its implementation.

Application

A software program or set of related programs that provide some useful capability or functionality.

Association

Represents a semantic connection between two classes. An association is comprised of relations that specify between classes. For example, the types ‘Person’ and ‘Company’ with the relations ‘Employer’ and ‘Staff’ defined for the types respectively, defines the ‘Employment’ association. Associations are bi-directional. Also called ***Association Relationship***.

Attribute

A data element which holds information about an object. Attributes implement the properties of a type and are used to maintain the state of the instances of a class.

Behavior

The observable effects of a request.

Cardinality

The count of objects in a set of instances of a type.

Class

An abstraction that captures the common structure and common behavior of a set of objects. It is a category that defines an object or group of objects, and how the objects are to behave. Objects can exist within the class as member objects. In other words, the class is a template for defining the methods and variables for a particular type of object. All objects in a given class are identical in form and behavior but contain different data in their variables.

A class usually sits in a “class hierarchy”, where the classes connected above a particular class are the parent (base) classes, while the classes connected below are the child classes.

Class Hierarchy

A characterization of the generalization/specialization relationships between classes. A class hierarchy is typically shown as a tree (directed acyclic graph) for single inheritance or a lattice for multiple inheritance. The nodes represent classes and are connected by arcs to indicate inheritance relations.

Class Diagram
A logical view of a system that shows the existence of classes and class categories, and their relationships.

Class Member
An attribute or method specified for a class.

Cohesion
A description of the strength of association of the contained data and function. A class has strong relationships among its operations.

Component
An object that is part of another object. I.e. there is a whole/relationship between two classes.

Composite
An object that is formed by two or more objects. The composite object is defined by its parts.

Consistency
The assurance that a set of values satisfy prescribed conditions.

Constraint
A condition or proposition that must be maintained as true. Constraints are typically applied to the permissible states or operations of an object.

Data Structure
One or more data elements in a particular relationship, usually used to describe some entity.

Decomposition
The process by which a thing, such as a use case, is broken down into its constituent parts.

Design
Design is the process that extends and modifies an analysis specification; it accommodates certain qualities in an application including extensibility, reusability, testability, and maintainability. Design also includes the specification of implementation requirements such as data persistence.

Domain
The problem or subject to be addressed by a computational or other system. Also, a set of objects mapped into a set of values by a relation.

Domain Expert
An individual who uses the concepts in a particular problem, process, or activity, e.g., users, programmers, or managers.

Entity
Something about which information is stored in a database.

Encapsulation
The characteristic that an object is a discrete unit of data with a set of permissible operations that control access or modification of the values. Encapsulation is the mechanism by which external aspects are separated from the implementation of an object, thus making methods and data private to an object.

Event
A stimulus that causes a noteworthy state change of an object or signal that invokes the behavior of an object.

First Normal Form

A property of a relational schema such that an attribute cannot contain a repeating group (e.g., flat file)

Framework

A specification or implementation (i.e., a collection of classes) that provide a general solution to some problem or aspect of applications. A framework is usually refined to address the specific problem through specialization or additional types or classes. MacAPP, Motif, and the Common Object Request Broker Architecture (CORBA) are examples of frameworks.

Harmonization

The process of reconciling two or more models together. Harmonization involves identifying cases in which the same structure appears in both models, and ensuring that the representation is consistent across both models.

Identity

The unique nature of an object that distinguishes it from all other objects.

Information hiding

The process of hiding all the secretes of an object that do not contribute to its essential characteristics; typically, the structure of an object is hidden as well as the implementation of its methods.

Inheritance

The capability by which properties (data and methods) are inherited by a child class from the parent (base) class. The inheritance feature of object-oriented programming allows a programmer to "re-use" functions and facilities of other objects, without having to copy the code.

Interoperability

The ability to exchange requests between entities. Objects interoperate if the methods that apply to one object can request services of another object.

Layer

The collection of class categories or subsystems at the same level of abstraction.

Life Cycle

The process of creating a hardware/software system from its conception, through analysis, design, implementation, testing, and maintenance.

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'.

Method

A procedure contained within an object that is made available to other objects for the purpose of requesting services of that object.

Model

A representation of a problem or subject area that uses abstraction to express concept. A model is often a collection of schema and other documentation.

Modularity

The property of a system that has been decomposed into a set of cohesive and loosely coupled modules.

Module

A function or a set of functions referred to by name. A collection of objects, methods, and classes that collaborate to provide a subset of the functionality of an application.

Object

A software packet 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-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.

Object Identity

The feature that the existence of an object is independent of any values associated with the object.

Object-Oriented

A principle of analysis, design, and programming that treats system components as objects, i.e., any method, language, or system that supports object identity, classification, encapsulation, and specialization. C++, Smalltalk, Objective-C, and Eiffel are examples of object-oriented implementation languages.

Object Request Broker (ORB)

A software mechanism by which objects make and receive requests and responses.

Overload

To define more than one method with the same name, but with a different signature and implementation.

Paradigm

An acquired way of thinking about something that shapes thought and action in ways that are both conscious and unconscious. Paradigms are essential because they provide a culturally shared model for how to think and act, but they can present major obstacles to adopting newer, better approaches.

Private

A visibility designation that prevents the clients of a class (except friends) from accessing the associated class members.

Process

A set of operations (functions), i.e., a sequence of behavior.

Reference Model

The model which serves as the authoritative repository of the data, use cases and other items that have been developed by HL7.

Referential Integrity

The assurance that an object reference identifies a single object.

Responsibility

A set of related behaviors specified for a type. Responsibilities are a means to organize the various behaviors of a type by a common purpose or functionality/

Scenario

A sequence of events that occur during one particular execution of a system.

Schema	A formal representation with a defined set of symbols and rules that govern the formation of a representation using the symbols.
Second Normal Form	Property of a relational schema such that it is in first normal form and each row has a primary key.
Semantics	The semantics of a type is the meaning or definition of the type.
State	Represents all of the object's static properties and the current value of each property.
State Diagram	Shows the event-ordered behavior of the instances of the class.
State Transition	A change of state caused by an event.
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.
System	An organized collection of computer or other components that interact.
Ternary Association	An association between three classes.
Third Normal Form	Property of a relational schema such that it is in second normal form and each non-primary key fields depends on the primary key.
Tuple	An ordered set of values (a record).
Variable	A storage place within an object for a data element.
Visibility	The accessibility of the attributes and methods of a class.

1