Extending the AIM Model

Topics in this guide include:

- Abstract
- Background and Objectives
- AIM Basic Concepts
  - What is an AIM annotation?
- From AIM 3.0 to the AIM Foundation Model
  - Why Use AIM Statements and Other Enhancements?
  - Fundamental Changes from AIM 3.0 to the Foundation Model
- AIM UML Modeling
- How to Create and Use AIM Annotations
- Guidelines to the AIM Foundation Model
- Extending AIM Foundation Model to AIM 4.0
  - AIM Library
  - Using AIM Library
  - ImageAnnotation
  - AnnotationOfAnnotation
  - AIM Library Objects
  - AIM Library Operations
  - Environment Configuration
  - Sample Code
- References

Original Publication Date: November 30, 2013

Abstract

The Annotation and Image Markup (AIM) Foundation information model is an evolution of AIM version 3 revision 11. It has evolved in response to the feedback and changing demands from the imaging community. Feedback was collected as part of the requirements for improving the foundation model. The model is described in a Unified Modeling Language (UML) class diagram and used as a base model for other imaging disciplines to extend information collection not currently captured in the Foundation model. The model captures imaging physical entities and their characteristics, imaging observation entities and their characteristics, markups (two- and three-dimensional), AIM statements, calculations, image source, inferences, annotation role, task context or workflow, audit trail, AIM creator, equipment used to create AIM instances, subject demographics, and adjudication observations.
The AIM 4.0 model is an extension of the AIM Foundation model. It has nine additional classes for storing lesion observation information in Radiology and Oncology.

The objective of this document is to provide information on how the AIM Foundation model can be extended to store the additional information that other imaging disciplines require.

Executive Summary

The objective of this document is to provide information on how the Annotation and Image Markup (AIM) [1-4] Foundation model can be extended to store the additional information that other imaging disciplines require. The AIM Foundation information model is an evolution of AIM version 3 revision 11. It has evolved in response to the feedback and changing demands from the imaging community. Feedback was collected as part of the requirements for improving the foundation model. The AIM 4.0 model is an extension of the AIM Foundation model. It captures lesion observation information for Radiology and Oncology.

The AIM standard has demonstrated its usefulness by integrating the descriptive information of an image or images with user-generated graphical symbols and textual descriptors placed on the image into a single common information source. The AIM information model provides a supporting infrastructure for creating and collecting image annotations. Annotation of findings and objects of interest in large clinical or research data collections are fully supported in the model. The AIM project focuses on annotation and markup of any image type. These annotations and image markups are information objects that are linked to (but separate from) the images. This approach is forward-looking, providing an infrastructure for future extension of the base information model that will enable future clinical and research projects, such as annotation of pathology and microscopy images, as well as documentation and tracking of quantitative changes to image features.

The AIM Foundation and AIM 4.0 model are used to express and capture image annotation and markup information relevant to images. They are described using a Unified Modeling Language (UML) class diagram. An annotation can contain explanatory or descriptive information that is generated by humans or machines, which directly relates to the content of a referenced image or images. It describes information regarding the meaning of pixel information in images. Annotations also become a collection of image semantic content that can be used for data mining purposes. An image markup is a graphical symbol or textual description associated with an image. Markups can be used to depict textual information and regions-of-interest visually alongside of, or more typically when overlaid upon, an image.

Both models capture imaging physical entities and their characteristics, imaging observation entities and their characteristics, markups (two- and three-dimensional), AIM statements, calculations, image source, inferences, annotation role, task context or workflow, audit trail, AIM creator, equipment used to create AIM instances, subject demographics, and adjudication observations. The 4.0 model has nine additional classes for storing lesion observations in Radiology and Oncology.

The AIM 4.0 model is used to create the AIM library and a validation and transformation tool. The AIM library is written in C++, using DCMTK and Xerces for DICOM and XML manipulation and creation. The library has two logical components: implementation of the AIM UML model as an object model and definition of transformations, which can be performed on the AIM object model. The AIM library has a collection of public mutator and accessor methods or APIs to manipulate and retrieve information in the AIM model. It also has a set of APIs to read, write, and transcode AIM artifacts to and from AIM DICOM Structured Reporting (SR) [5] objects and AIM XML documents. The ANIVATR tool is a referenced implementation and used for validating AIM annotations and transcoding between AIM XML documents and DICOM SR objects.

This document provides information about the AIM Foundation model and how the model can be extended to store and support other types of imaging disciplines.

Background and Objectives

The objective of this project is to provide sufficient information about AIM 4.0 and guidelines for extending AIM foundation model for other imaging domains.

Modern medical images contain vast amounts of information captured in standard DICOM format. While this information may include metadata about the image, such as how or when the image was acquired, the majority of image information remains in pixel data. This data contains rich content that is neither explicit nor easily accessible by computer programs. The information about how images are perceived by human or machine observers is not currently captured in a form that is directly tied to the images in a structured manner. A wealth of data pertaining to image content is thus segregated from the images, limiting the value of radiologic images for use with other non-imaging data, such as cancer clinical trials.

There is neither widely-adopted standard terminology nor syntax used to capture annotation, markup, and computational descriptions of image features or non-imaging biomedical data. This results in limited interoperability between imaging and health information system. The majority of the human image features and descriptions in the biomedical domain are captured only as free text. This free text often has no association with the spatial location of the feature, making it difficult to relate image features that are described in that text to the corresponding image locations. Free text is also cumbersome, in both the lay and technical sense, as indexing, querying, and searching it to retrieve images or their features are time-consuming.

Image annotation information is collected in both a clinical setting of commercial information systems and imaging research tools of more flexible and tailored software provenance. This diversity of systems has led to the development of a variety of technical frameworks and standards. However, we still need an information model and imaging tools that can be used by imaging interpreters and machine programs to store image findings in a standard format. Selecting a single standard format to store image annotations will streamline software development, and enable the work to focus on providing rich annotation features and functionalities. Designing the software library and tools for compatibility with other standards will enable a high degree of interoperability and allow the incorporation of the annotation standard into commercial and clinical information systems. This has the potential to open many existing resources and databases of image data and metadata for exploitation by the broader research and clinical radiology community.

AIM was created to deliver an information model and tools that will allow both human and computer programs to create annotations in a standard format that is syntactically and semantically interoperable with the informatics infrastructure of National Cancer Informatics Program (NCIP), using the DICOM [6] and HL7 healthcare standards.

The AIM model has to date undergone several major revisions. After the AIM 3.0 model [2] was released and used by image researchers, new requirements emerged, as follows: 1) use ISO 21090 data type; 2) explicitly declare association relationships between AIM major or entity type classes via AIM statements; 3) store three-dimensional markup; 4) store compact calculation results; 5) add the ability to declare task context or workflow; 6) store more than one image annotation or annotation of annotation in a single AIM instance; 7) store a question as a coded term in AIM entity classes; 8) store a unique reference for other types of DICOM objects; 9) improve storing for references for image reference class; 10) store DICOM general image and image plane information; 11) store a unique reference for DICOM segmentation objects; and 12) store information about adjudication observations.

The AIM Foundation model was created to respond to the above requirements. The model had been designed for extensibility by other image researchers to capture information not available in the Foundation model. The AIM 4.0 model is an extension model from the Foundation model. It stores lesion observation information for radiology and oncology.

AIM Basic Concepts

An **image annotation** can be explanatory or descriptive information, generated by humans or machines, directly related to the content of a referenced image or images. It describes information about the meaning of pixel information in images. Annotations become a collection of image semantic content that can be used for data mining purposes. An **image markup** is a graphical symbol or textual description associated with an image. Markups can be used to visually depict textual information and the region-of-interest next to an image.

AIM provides a standard way to store annotation information in a structured manner with standard terminology such as RadLex, SONMED CT, etc. Searching the majority of AIM data can be done using coded terms stored in an AIM instance. AIM data can be stored as AIM DICOM Structured Reporting (SR) objects, AIM XML documents, and AIM HL7 CDA documents.

What is an AIM annotation?

An **AIM annotation** is a collection of associated image annotations and markups. An AIM annotation may contain one or more image annotation or annotation of annotation instances. ***An AIM annotation can only be of either type image annotation or annotation of annotation.*** These are the two kinds of annotations that can be created. *ImageAnnotation* class annotates images. *AnnotationOfAnnotation* class annotates other AIM annotations (both image annotation and annotation of annotation) for comparison and reference purposes.

In the AIM 3.0 model, an image annotation or annotation of annotation is stored as a single file, either as an AIM DICOM SR or AIM XML document. When AIM was used in Pathology, hundreds of thousands of AIM files were created for a single study. Managing AIM files from different image studies became very complicated. A collection described in the next section was used to store AIM instances from the same imaging study.

The next section describes changes from AIM version 3.0 to the AIM Foundation model.

From AIM 3.0 to the AIM Foundation Model

Based on the new requirements in the background and objective section, the development team modified and extended the AIM 3.0 model [2] to create the AIM Foundation model. The Foundation model extends the model to support other imaging disciplines. This section provides a real-world example how of the Foundation model was created.

The most significant change from the AIM 3.0 to the AIM Foundation model is the use of **ISO 21090 data types** and the introduction of **AIM Statements**. An AIM Statement describes a relationship between the subject and object entities in the AIM Foundation model. As such, this is a generic and very expressive way to capture a broad range of information AIM annotations need to capture. AIM statements are a core aspect of the design of AIM that will permit it to be extended in future to meet the needs of other domains.

Why Use AIM Statements and Other Enhancements?

The AIM 3.0 model reflects relationships between classes using containment of a source class to a target class and inheritance or IS-A relationships. The expressive power of the model is limited by these two types of relationships. No other types of relationships in the AIM 3.0 model are possible and not all necessary relationships are present. For instance, there is no direct relationship between instances of the *AnatomicEntity* class, e.g. right-upper lobe of lung, and the *ImagingObservation* class, e.g. mass. Such classes can be indirectly linked to each other only via the *Annotation* class.

The desire to improve the expressiveness of the AIM model and specific use cases prompted us to create a flexible model of AIM statements. Use cases that led us to change the AIM model from the containment association approach to explicitly stated relationships between two classes follow.

1. Justin Kirby at the NCI's Cancer Imaging Program has a use case for storing information from a mammography case report form (CRF). The CRF has "Associated Findings" or imaging observation characteristics that are associated with the entire breast and are not specific to a mass. Dr. David Channin at Guthrie Clinic also has a similar use case.
2. Dr. Lior Weizman, a research fellow working with Dr. Daniel Rubin, wants to associate calculation results with a DICOM segmentation object.
3. Several AIM users want to capture imaging observations and calculations related directly to image markup.
4. A user wants to capture a measurement of a liver volume from a CT scan to facilitate clinical assessment of liver disorders, to improve decision making in liver transplant surgery and to avoid donor-recipient graft mismatch.

Adding an association relationship between the *AnatomicEntity* and *ImagingObservationCharacteristic* resolved the CRF issue. Adding an association between segmentation and calculation satisfied Dr. Weizman's comment. Applying the same approach could satisfy use cases 3 and 4.

As the AIM model is used by an increasing number of users, additional requests will arise that add relationships between classes or create new classes to store other important information related to AIM. Managing and rearranging associations with classes will be too complex to manage without the AIM *State ment* class and its subclasses.

Fundamental Changes from AIM 3.0 to the Foundation Model

The foundation has seventy new classes. We renamed twelve classes and deleted four classes. The detailed AIM model change logs are at https://wiki.nci.nih.gov/x/N4FSBg. This section describes changes from the AIM 3.0 to AIM Foundation model.

The AIM Foundation model uses ISO 21090 data types.

AIM 3.0 Data Type	ISO 21090
Boolean	BL
CalculationResultIdentifier	Not Applicable
ComparisonOperators	Not Applicable
Date	TS
Double	REAL
Integer	INT or II
String	ST or Uid

Four attributes in an AIM class are now mapped to a single ISO 21090 CD data type.

AIM 3.0 Data Type	ISO 21090
codeValue	CD
codeMeaning	CD
codingSchemeDesignator	CD
codingSchemeVersion	CD

All entity type classes in AIM Foundation derive from the Entity class. The Entity class is an abstract class. It represents the existence of a thing, concept, observation, calculation, measurement, or graphical drawing in AIM. It is a parent class of all entity classes in the foundation model. Entity classes in both models are as follows.

- AnnotationEntity (abstract class)
- AnnotationRoleEntity
- CalculationEntity
- DicomImageReferenceEntity
- DicomSegmentationEntity
- Entity (abstract class)
- GeometricShapeEntity (abstract class)
- ImagingObservationEntity
- ImagingPhysicalEntity
- ImageReferenceEntity (abstract class)
- InferenceEntity
- MarkupEntity (abstract class)
- SegmentationEntity (abstract class)
- TaskContextEntity
- TextAnnotationEntity
- ThreeDimensionGeometricShapeEntity (abstract class)
- TwoDimensionGeometricShapeEntity (abstract class)
- UriImageReferenceEntity

An AIM Statement describes a relationship between two entity type classes in the AIM Foundation model. **An AIM statement has a subject-predicate-object construct to precisely define a relationship between two classes: a subject and an object class.** The introduction of AIM statements requires structural and class name changes from the AIM 3.0 model. **The naming convention used to create an AIM statement is a concatenation between subject, predicate, and object. A predicate defines a relationship between a subject and an object.** Subject and object information come from entity classes in AIM models. The current predicates in AIM are as follows.

- Excludes
- Has
- IsComparedWith
- IsCompriseOf
- IsFoundIn
- IsIdentifiedBy
- References
- Uses

An AIM statement expresses the most granular amount of information an AIM annotation can have. An AIM annotation presents its content in a collection of semantic statements. Statements describe a thing found, measured, and graphically annotated on an image or images from the same series in an imaging study. This approach provides AIM with the ability to describe images at varying granularity and thus is highly extensible, capable of meeting the needs of future use cases not yet even anticipated.

Each entity class in the AIM model must have a unique identifier (UID) that must conform to the DICOM UID format. An AIM statement contains a UID from a subject and an object. A name of an AIM statement class expresses an explicit meaning and relationship between two entity type classes (subject and object) in the AIM model.

Three abstract classes, namely *AnnotationStatement*, *ImageAnnotationStatement* and *AnnotationOfAnnotationStatement*, are used to create AIM statements.

The *AnnotationStatement* class represents a general concept about a statement used to describe something found that can be compared or addressed on an image or images in a series. It can be used for image annotation or annotation of annotation. The *AnnotationStatement* class has eight different subtypes of annotation statements that can be applied to both annotation of annotation and image annotation. The statements available in this group are depicted in figure 1. Classes that can be used for annotation statements are as follows (descriptions of each of these classes and those mentioned below are given in Section 5).

- *AnnotationEntity* (abstract class)
- *AnnotationRoleEntity*
- CalculationEntity
- *ImagingObservationEntity*
- *ImagingPhysicalEntity*

- *TaskContextEntity*

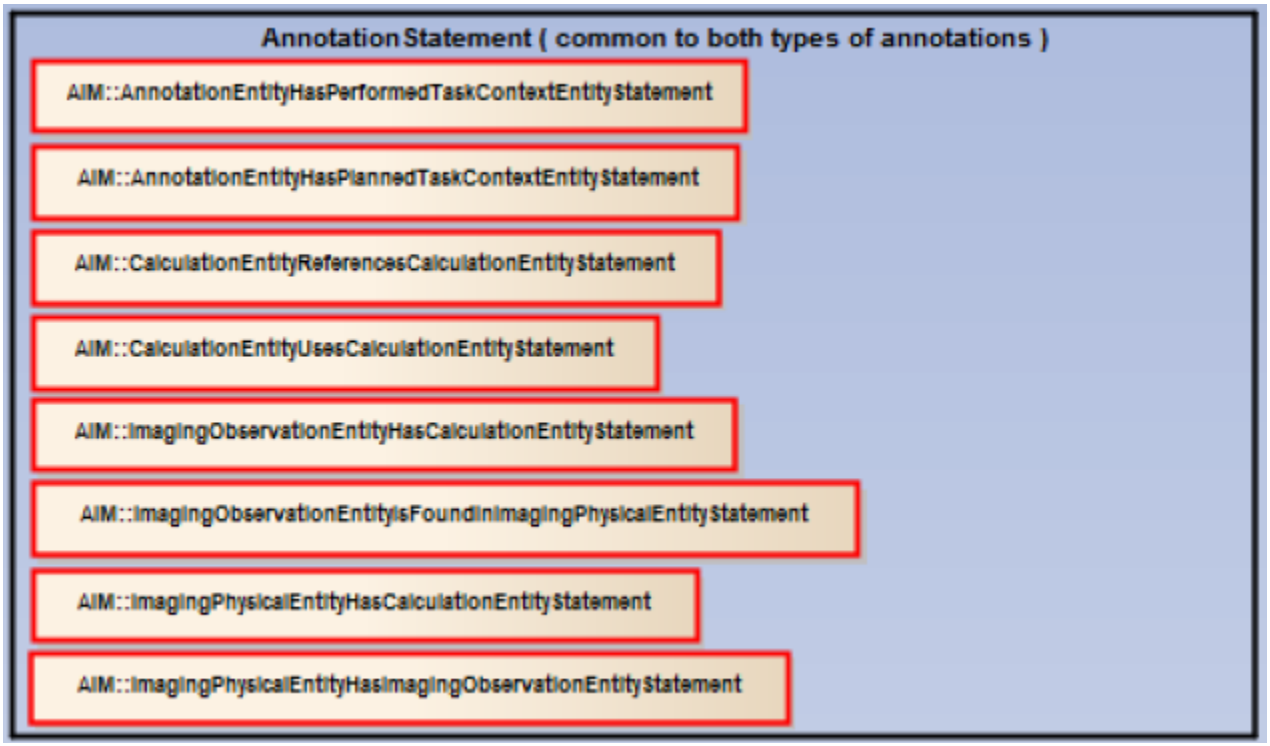

Figure 1. Annotation statements can be used for image annotations and annotation of annotations

The *AnnotationOfAnnotationStatement* class represents a group of statements used to describe annotation-of-annotation relationships between a subject and an object class entity that do not have direct reference to the image or images. Classes that can be used for annotation-of-annotation statements are as follows.

- *AnnotationOfAnnotation*
- *AnnotationRoleEntity*
- *CalculationEntity*
- *ImageAnnotation*
- *ImagingObservationEntity*
- *ImagingPhysicalEntity*
- *InferenceEntity*
- *TaskContextEntity*

Statements available from this group are shown in Figure 2.

Figure 2. Annotation of annotation statement can be used for AnnotationOfAnnotation

The *ImageAnnotationStatement* class represents a group of statements used to describe image annotation relationships between a subject and an object class entity that have direct reference to the image or images. Classes that can be used for image annotation statements are as follows.

- *AnnotationRoleEntity*
- *CalculationEntity*
- *DicomImageReferenceEntity*
- *DicomSegmentationEntity*

- *ImageAnnotation*
- *ImagingObservationEntity*
- *ImagingPhysicalEntity*
- *InferenceEntity*
- *TaskContextEntity*
- *TextAnnotationEntity*
- *UriImageReferenceEntity*

Statements available for the image annotation statement group are in Figure 3.


ImageAnnotationStatement
AIM::DicomImageReferenceEntityHasCalculationEntityStatement
AIM::DicomImageReferenceEntityHasImagingObservationEntityStatement
AIM::DicomImageReferenceEntityHasImagingPhysicalEntityStatement
AIM::DicomSegmentationEntityHasImagingObservationEntityStatement
AIM::ImageAnnotationHasCalculationEntityStatement
AIM::ImageAnnotationHasChildImageAnnotationStatement
AIM::ImageAnnotationHasDicomImageReferenceEntityStatement
AIM::ImageAnnotationHasDicomSegmentationEntityStatement
AIM::ImageAnnotationHasImagingObservationEntityStatement
AIM::ImageAnnotationHasImagingPhysicalEntityStatement
AIM::ImageAnnotationHasInferenceEntityStatement
AIM::ImageAnnotationHasTextAnnotationEntityStatement
AIM::ImageAnnotationHasThreeDimensionGeometricShapeEntityStatement
AIM::ImageAnnotationHasTwoDimensionGeometricShapeEntityStatement
AIM::ImageAnnotationHasUriImageReferenceEntityStatement
AIM::ImagingObservationEntityIsIdentifiedByThreeDimensionGeometricShapeEntityStatement
AIM::ImagingObservationEntityIsIdentifiedByTwoDimensionGeometricShapeEntityStatement
AIM::ImagingObservationEntityIsIdentifiedByTextAnnotationEntityStatement
AIM::ImagingPhysicalEntityHasThreeDimensionGeometricShapeEntityStatement
AIM::ImagingPhysicalEntityHasTwoDimensionGeometricShapeEntityStatement
AIM::ImagingPhysicalEntityHasTextAnnotationEntityStatement
AIM::ThreeDimensionGeometricShapeEntityIsComprisedOfThreeDimensionGeometricShapeEntityStatement

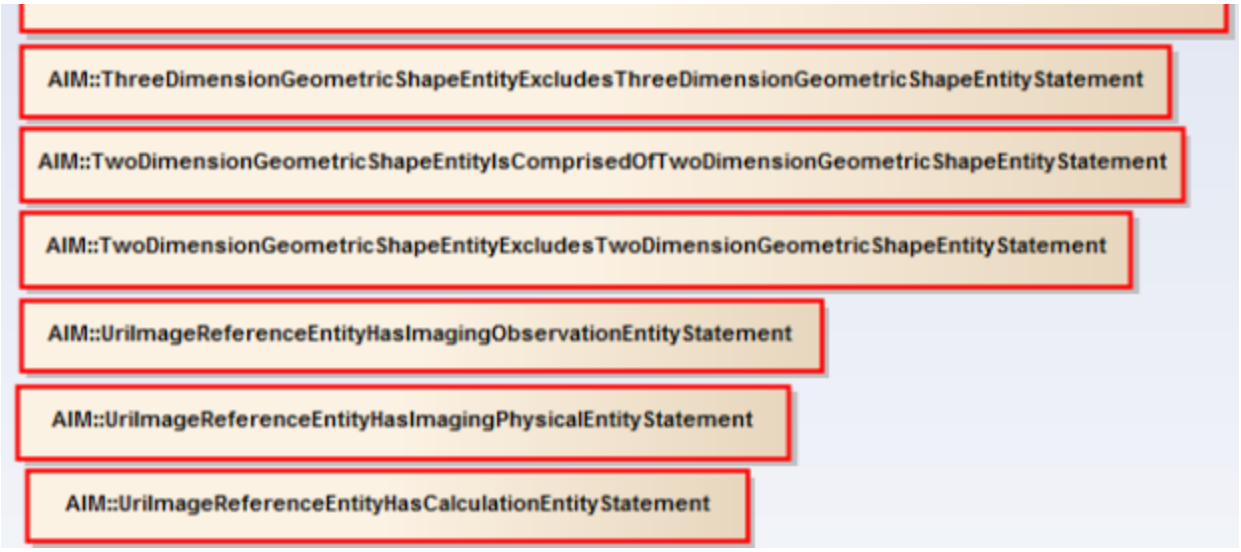

Figure 3. Image annotation statement can be used for ImageAnnotation

1. An AIM annotation, either stored as XML documents or DICOM Structured Reporting objects, can store more than one image annotation and annotation-of-annotation instance.
2. The AIM Foundation has a capability to store both two and three-dimensional markups placed on images. Each two and three-dimensional markup has its own graphical type. All spatial coordinate and graphical types comply with DICOM standard part 3 [6].
3. Calculation results are stored as individual data elements in AIM 3.0. This approach takes a lot of storage space resulting in a large file size if there is a large amount of data that must be saved. The AIM Foundation offers a new storage mechanism whereby compression of data is allowed.
4. The AIM Foundation model supports clinical and research workflow activity. The *TaskContextEntity* class contains identifying and descriptive attributes of the reading session and the reading subtask that result in clinical environment or trial results. The class consists of the overall task and the specific subtask. A task represents a unit of overall work, e.g. "Read all of the available timepoints for the subjects". It may have one or more subtasks. The class can be used to capture and record planned (scheduled) and performed tasks.
5. The *ImageAnnotationCollection* and *AnnotationOfAnnotationCollection* classes are required to store the same type of AIM annotations, image annotation, and annotation-of-annotation respectively. *ImageAnnotationCollection* stores AIM instances from the same imaging study and in a reading session. *AnnotationOfAnnotationCollection* stores AIM statements that reference image annotation and annotation-of-annotation instances as well as any image findings, calculations, inferences, workflow, annotation role, and/or adjudication observations.

Understanding the AIM Foundation Model

The model is used to capture image annotation and markup information relevant to images. It describes explicitly what kind of information the model can collect. An annotation describes information about the meaning of pixel information in images. It is captured as a coded term supplied by medical lexicons (e.g. SNOMED CT®, RadLex, LOINC, etc.) or user-defined terms. Annotations become a collection of image semantic content that can be used for data mining purposes. An AIM annotation is a collection of associated annotations, markups, calculation results that may or may not be directly related to a markup, a well-defined role of an annotation (e.g. baseline or follow-up), workflow activity, simple subject demographics, and creator and tools used to create AIM instances. This information is used to design and create AIM information model.

AIM UML Modeling

The AIM UML model illustrated as a UML class diagram is used to capture information about how images are perceived by human or machine observers. Our design process started with understanding the initial requirements [2] and the information in "From AIM 3.0 to the AIM Foundation Model," on page . We identified a set of information objects that are used to collect information about imaging annotations and markup. Classes are divided into image semantic content, calculation, markup, image reference, and AIM statements. If there are classes that do not pertain to a specific group, we classify them in the general information group. These classes contain information about the workstation used to create the AIM annotations, the user that creates the AIM annotations, patient identification, DICOM segmentation, annotation role, inference, workflow activity, adjudication observation, image annotation, and annotation-of-annotation.

The AIM Foundation Model, shown in figure 1, has evolved through an iterative feedback process since the release of AIM version 3, revision 11. The model has gone through many reviews and recommendation processes. Enterprise Architect can be used to view the AIM UML class diagram file, *AIM_Foundation_v4_rv47_load.eap*. One can also view this diagram in JPEG format. You can download the model from the NCI Wiki.

Based on the information in Figure 4, we can categorize the collection of classes into six groups: General Information, Calculation, Image Semantic Content (Finding), Markup, Image References, and AIM Statements. AIM statements are described in "From AIM 3.0 to the AIM Foundation Model" (need reference). Descriptions of the other five groups follow.

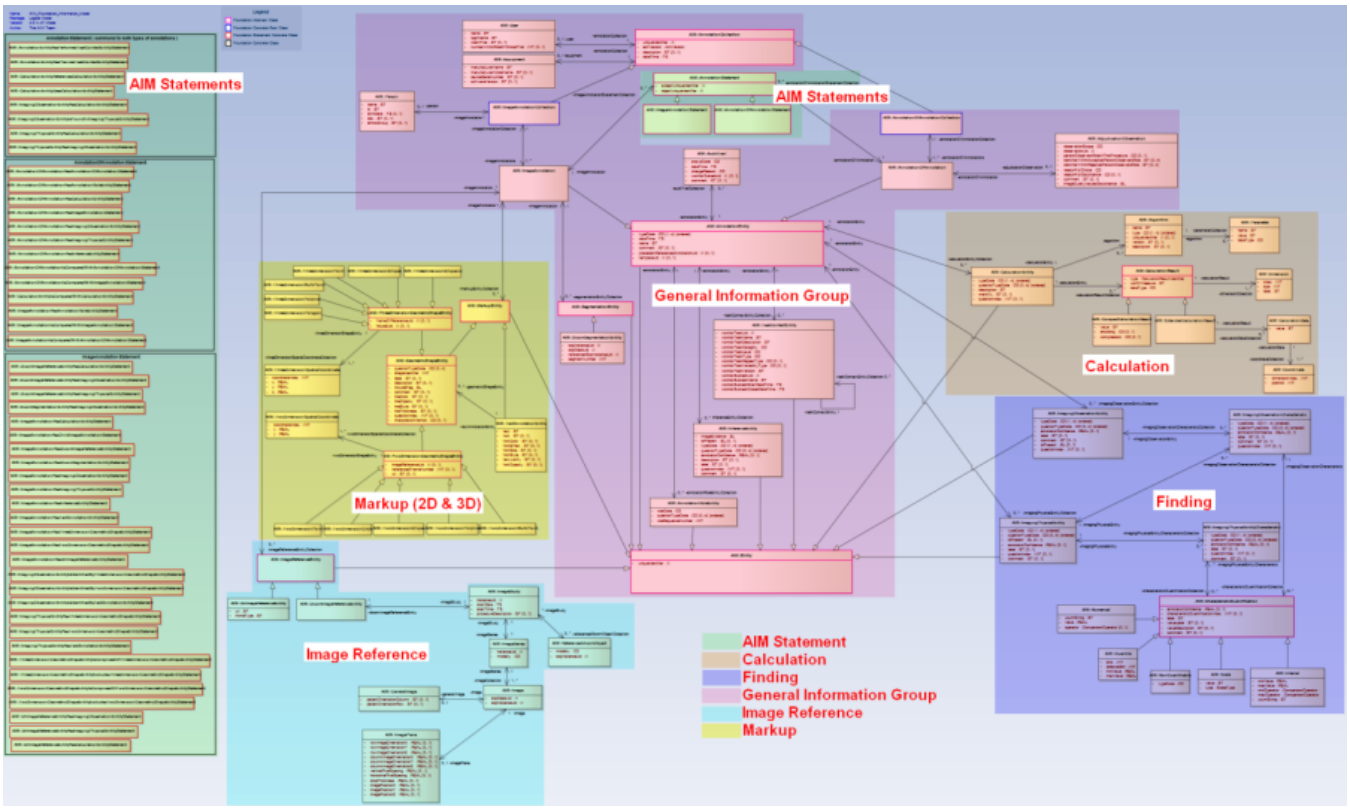

Figure 4. AIM Foundation Model

The General Information group, shown in Figure 5, contains a collection of classes that do not belong to any of the five other groups. Each class is self-described and does not work with other classes to provide a larger concept. We begin with *AnnotationCollection*. It is an abstract concept of a container that collects the same type of annotation. It contains a UID used to identify a collection, a version of the AIM model used for collecting annotation data, creation date and time, and description about the AIM annotation contained in the collection. *AnnotationCollection* is the parent of *ImageAnnotationCollection* and *AnnotationofAnnotationCollection*. *AnnotationCollection* associates with two optional classes used to capture the information about a person or computer program generating AIM instances as well as the manufacturer's information of the software used to create AIM instances, which are *User* and *Equipment* class, respectively. The *User* class represents a person or a computing resource that creates an AIM annotation. The *User* class is composed of a user's full name, login name with optional role in trial, and order within a trial. The *Equipment* class provides information about the system that is used to create AIM annotations. The *Equipment* class collects manufacture name, model description, and software versions. *ImageAnnotationCollection* stores instances of *ImageAnnotation*. It associates with the *Person* class that contains basic patient demographic information: patient name, identification string, birth date, and sex. The *Person* class represents both humans and animals. *AnnotationofAnnotationCollection* stores instances of *AnnotationOfAnnotation*. *AnnotationOfAnnotation* can be used to annotate both image annotations and annotation-of-annotation via the AIM statements *AnnotationOfAnnotationHas AnnotationOfAnnotationStatement* or *AnnotationOfAnnotationHasImageAnnotationStatement*. As described in the previous section, an AIM statement contains two UIDs for subject (which is an UID of an annotation-of-annotation) and object (which is an UID of either annotation of annotation or image annotation).

Next, the *Entity* class is an abstract class that represents an existence of a thing, concept, observation, calculation, measurement, or graphical drawing in AIM. The class is the base class that other main concept classes extend. A class derived from the *Entity* class can be used to create AIM statements.

The next class that we should examine is *AnnotationEntity* class. It is an abstract base class for the *ImageAnnotation* and *AnnotationOfAnnotation* classes. The *AnnotationEntity* class captures name, a general description of the AIM annotation, type of annotation via controlled terminology, creation date and time, a reference to the AIM template used to create an annotation, a reference to a previously related AIM annotation, and AIM annotation UID.

The *ImageAnnotation* class annotates images. The *AnnotationOfAnnotation* class annotates other AIM annotations for comparison and reference purposes. Both *ImageAnnotation* and *AnnotationOfAnnotation* have *AnnotationRoleEntity*, *AuditTrail*, *InferenceEntity* and *TaskContextEntity* classes. The *ImageAnnotation* associates with an abstract class *SegmentationEntity*. The model currently supports DICOM Segmentation via *DicomSegmentationEntity*. DICOM Segmentations are either binary or fractional. The *DicomSegmentationEntity* class represents a multi-frame image representing a classification of pixels in one or more referenced images. The class contains the DICOM SOP class UID that defines the type of segmentation. It references its own instance UID and the instance UID of the image to which the segmentation is applied. It also has an identification number for the segment that shall be unique within the segmentation instance in which it is created. An *ImageAnnotation* may have zero or more segmentation objects. The *AnnotationRoleEntity* describes a role of an annotation. Each instance can have a role associated with it, e.g. a baseline case. The *InferenceEntity* class provides a conclusion derived through interpreting an imaging study and/or medical history. The *AuditTrail* class captures the status of an annotation instance using a coded term, contains identifying and descriptive attributes of the reading session, and the reading subtask that results in clinical environment or trial findings. The class consists of the overall task and the specific subtask. A task represents a unit of overall work. It may have one or more subtasks.

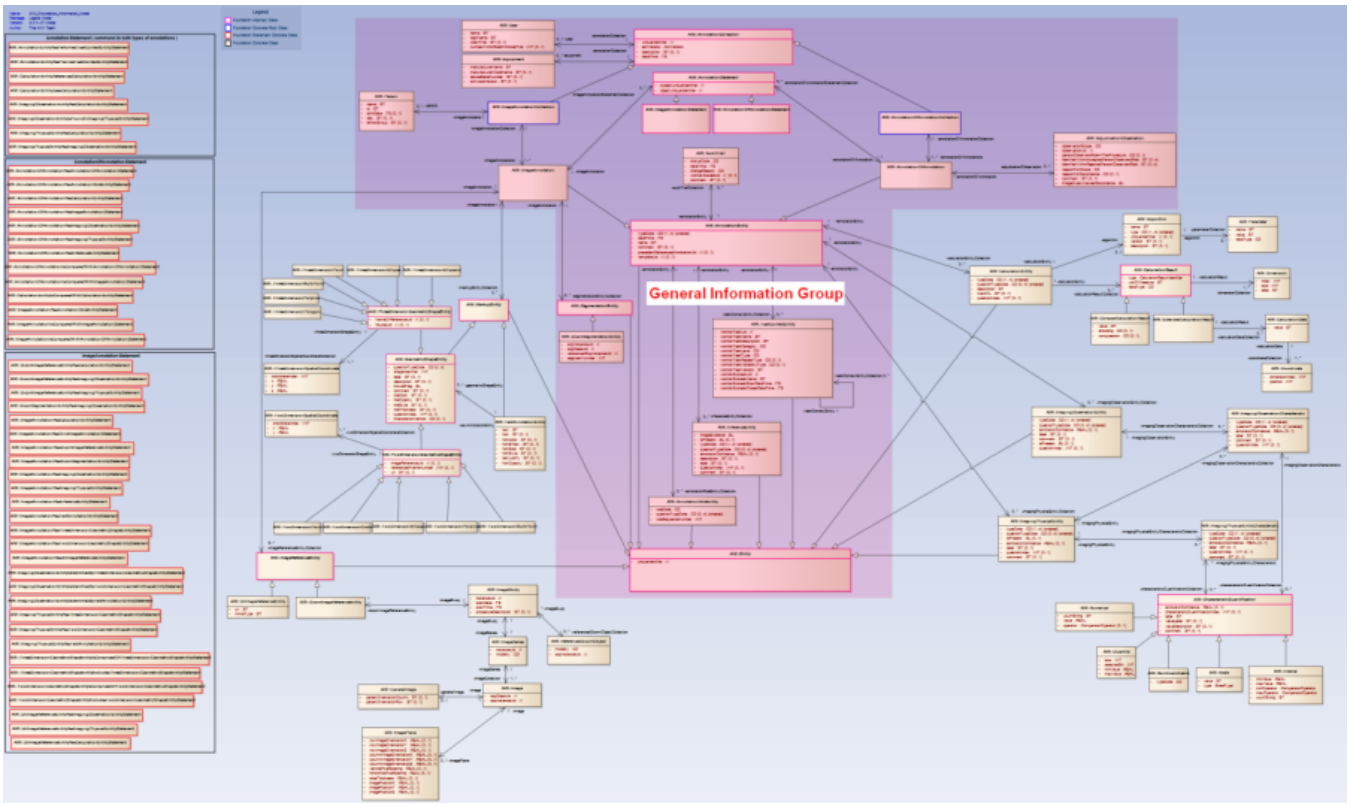

Figure 5. General Information Group

The Calculation group, shown in Figure 6, represents the calculation results of an AIM annotation. Calculation results may or may not be directly associated with graphical symbols or markups. For example, given an image with a single ellipse markup, calculation results can be an area in square millimeters and references maximum and minimum pixel values. As another example, an image has an arrow pointing to a specific location and two concentric circles, with an area measurement of the larger circle minus the smaller circle. The computation result is based on the independence calculation made on each circle. The AIM schema allows calculation results that are not directly related to markups. *CalculationEntity* class has overall information about a calculation performed related directly to image or images. It defines a type of calculation, such as area, height, radius, and volume of ellipsoid from UCUM, and takes the form of controlled terminology that can be captured in code value, code meaning, and coding scheme designator in a single attributed call, *typeCode*. It also captures MathML as a string attribute within the class. A *questionTypeCode* attribute captures a reason why a calculation is needed as code. A textual description can be stored in the description attribute. The *CalculationResult* abstract class contains the type of result (e.g. binary, scalar, vector, histogram, array, histogram, or matrix), unit of measurement, and coded data type (a primitive programming data type such as integer, double, etc. as well as other data type such as URI). A *CalculationResult* can be stored as a compact or an extended result, *CompactCalculationResult* and *ExtendedCalculation* result, respectively. *CompactCalculationResult* has three attributes: value, encoding, and compression. The result of a calculation is captured in a string format in a value attribute that can hold a value of array, binary, histogram, matrix, scalar, and vector. Encoding is an encoding method applied to the content of the value attribute. Compression is a method used to compress the content in value attribute. *CalculationResult* has an association with the *Dimension* class that states how many dimensions a *CalculationResult* has. *ExtendedCalculation* stores a result of a calculation individually with the precise location of each element in the result. The Data class is used to store the result value. The *Coordinate* class identifies location within a dimension for the Data class. A *CalculationEntity* may have a relationship with a markup or a collection of markups, other calculations, imaging observation, and imaging physical entity. These types of relationships can be captured as AIM statements.

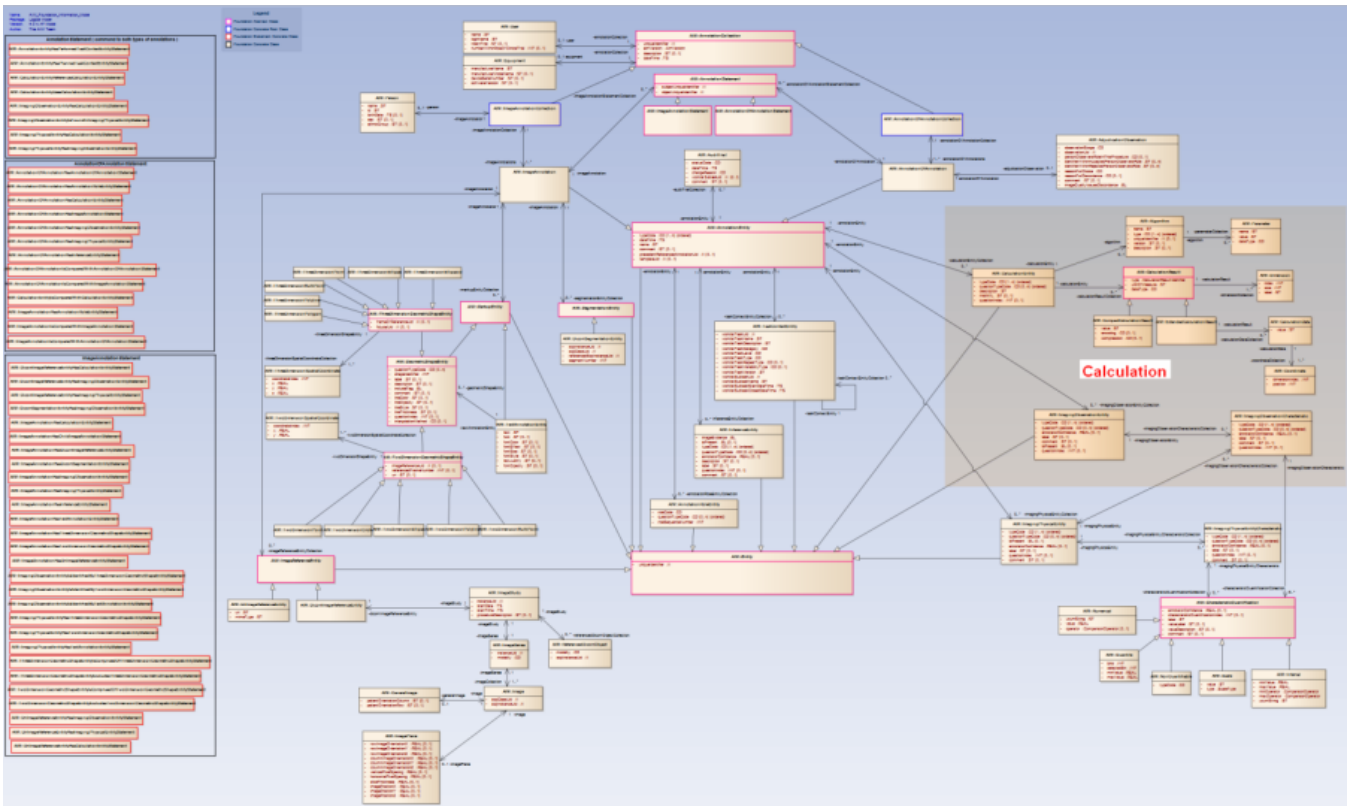

Figure 6. Calculation Group

The classes in the Image Semantic Content group, shown in Figure 7, are used to gather image findings or interpretations of images. The *ImagingPhysical Entity* class stores an anatomical location (e.g. femur) as a coded term from a recognized controlled vocabulary (RadLex, SNOMED-CT, UMLS, etc). The *I magingPhysicalEntityCharacteristic* class further describes the *ImagingPhysicalEntity* class such as "fracture". The *ImagingObservationEntity* class is the description of things that are seen in an image. "Mass," "Radiographic evidence of pleural effusion," "Foreign Body," and Artifact," are all examples of *Imagi ngObservationEntity*. The *ImagingObservationEntityCharacteristic* class includes descriptors of the *ImagingObservationEntity* class such as "dense," "heterogeneous," "hypoechoic," and "spiculated". Both *ImagingPhysicalEntityCharacteristic* and *ImagingObservationEntityCharacteristic* may be associated with *CharacteristicQuantification*. A quantification can be a numerical value, an interval (e.g. 34-67%), a scale (e.g. 1:None, 2:Mild), a quantile (e.g. 1(1-50), 2(51-100)), and a non-quantifiable (e.g. none, mild, mark).

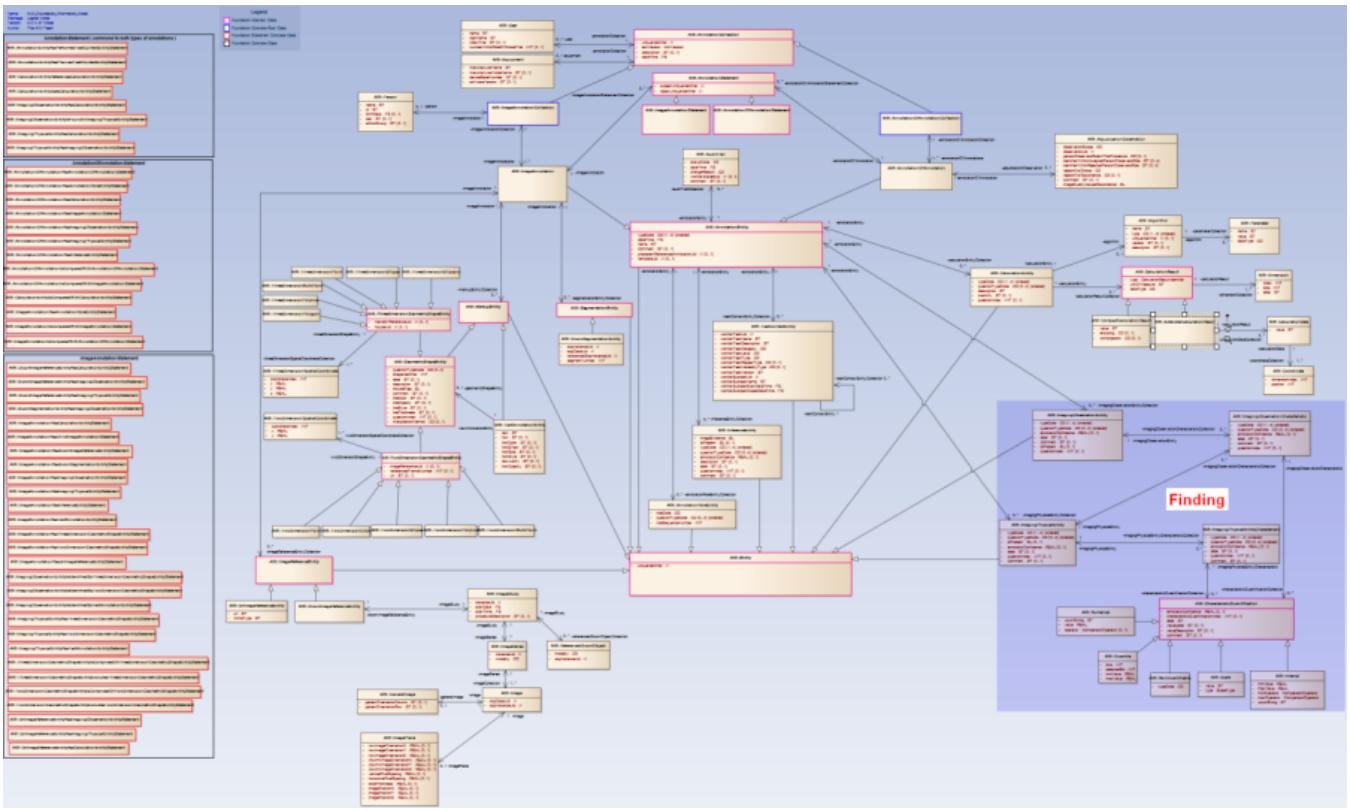

Figure 7. Image Semantic Content Group (Finding)

The Markup group, shown in Figure 8, captures textual information and graphical representation as DICOM SR value type SCOORD and SCOORD3D for two dimensions and three dimensions, respectively. The available graphic types for two dimensions are point, multipoint, polyline, circle, and ellipse. Each drawing type has an x and y coordinate defined in the *TwoDimensionSpatialCoordinate* class. The *TwoDimensionGeometricShapeEntity* class has *TwoDimensionSpatialCoordinate*, the SOPInstance UID of the image that contains the pixel and the frame number within the referenced SOP Instance to which the reference applies. The first frame shall be denoted as frame number 1. In the case of a multi-frame image, we use the frame number from the DICOM header. Available two-dimensional graphic types [6] are as follows.

Graphical Type	Description
POINT	A single pixel denoted by a single (column,row) pair
MULTIPOINT	Multiple pixels each denoted by an (column,row) pair
POLYLINE	A series of connected line segments with ordered vertices denoted by (column,row) pairs
CIRCLE	A circle defined by two (column,row) pairs. The first point is the central pixel. The second point is a pixel on the perimeter of the circle
ELLIPSE	An ellipse defined by four pixel (column,row) pairs, the first two points specifying the endpoints of the major axis and the second two points specifying the endpoints of the minor axis of an ellipse

The available graphic types for three dimensions are point, multipoint, polyline, polygon, ellipse, and ellipsoid. Each drawing type has x, y, and z coordinate defined in the *ThreeDimensionSpatialCoordinate* class. The *ThreeDimensionGeometricShapeEntity* class has *ThreeDimensionSpatialCoordinate* as well as the frame of reference for a Series. Available three-dimensional graphic types [6] are as follows.

Graphical Type	Description
POINT	A single location denoted by a single (x,y,z) triplet
MULTIPOINT	Multiple locations each denoted by an (x,y,z) triplet; the points need not be coplanar
POLYLINE	A series of connected line segments with ordered vertices denoted by (x,y,z) triplets; the points need not be coplanar

POLYGON	A series of connected line segments with ordered vertices denoted by (x,y,z) triplets, where the first and last vertices shall be the same forming a polygon; the points shall be coplanar
ELLIPSE	An ellipse defined by four (x,y,z) triplets, the first two triplets specifying the endpoints of the major axis and the second two triplets specifying the endpoints of the minor axis
ELLIPSOID	A three-dimensional geometric surface whose plane sections are either ellipses or circles and contains three intersecting orthogonal axes, "a", "b", and "c". The ellipsoid is defined by six (x,y,z) triplets, the first and second triplets specifying the endpoints of axis "a", the third and fourth triplets specifying the endpoints of axis "b", and the fifth and sixth triplets specifying the endpoints of axis "c"

The *TextAnnotationEntity* class has coordinates captured as SCOORD or SCOORD3D graphic type as *TwoDimensionMultiPoint* or *ThreeDimensionMultiPoint*, respectively. A *TextAnnotationEntity*'s MultiPoint implementation is expected to have no more than two coordinates that can be represented as an arrow connecting *TextAnnotationEntity* to a point on an image. Only the *ImageAnnotation* class can have markups.

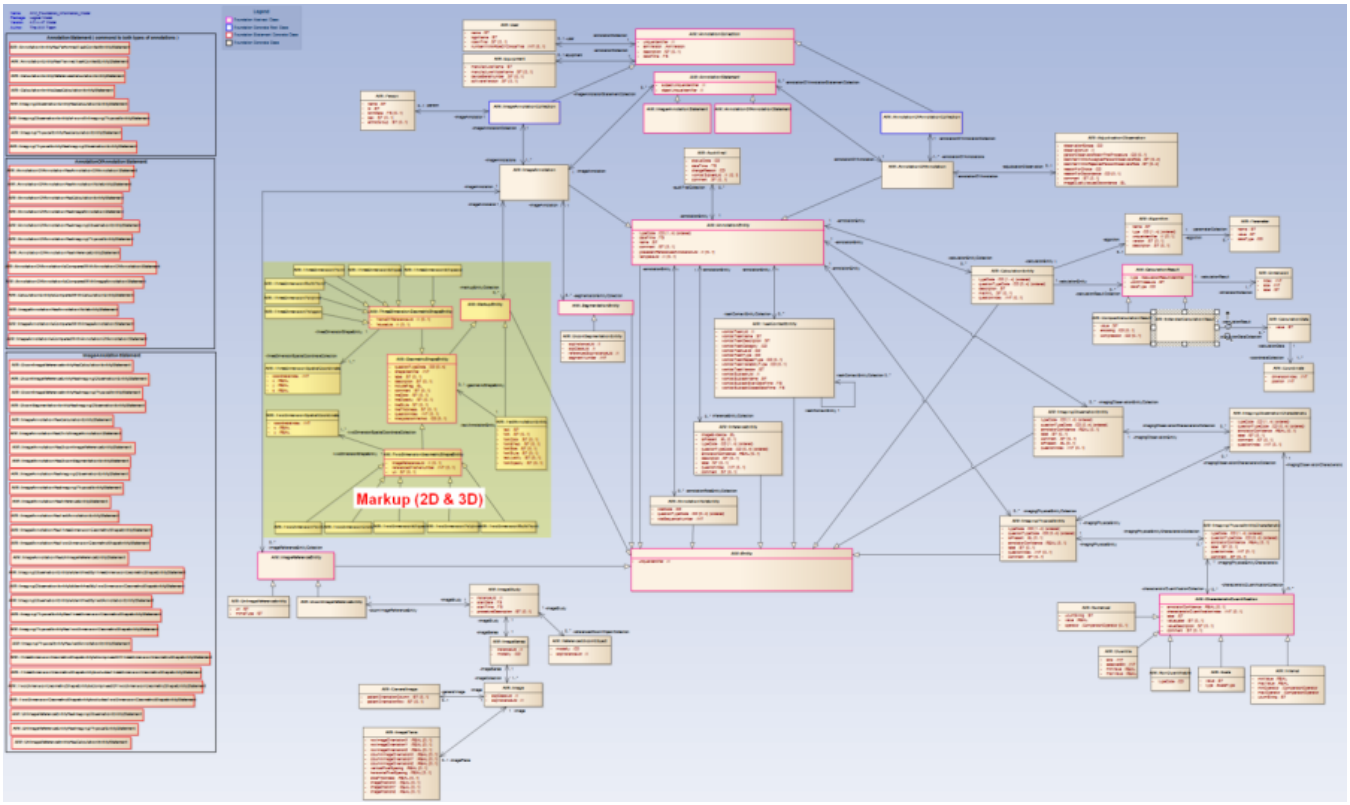

Figure 8. Markup Group

The *ImageReference* group, as shown in Figure 9, represents an image or collection of images being annotated. The two possible types of references are DICOM and URI or web image reference. First, *DICOMImageReferenceEntity* associates with other classes that mimic the DICOM information model. It has one *ImageStudy* object that has one *ImageSeries* object, which in turn has one or more *Image* objects. The *ImageStudy* class has study instance UID, start date and start time, and procedure description. *ImageStudy* may have zero or more references to DICOM objects via the *ReferencedDicomObject* class. The *ImageSeries* class has series instance UID. The *Image* class has SOP class UID and SOP instance UID. The *Image* class has two associations with *GeneralImage* and *ImagePlane*; both classes came from the DICOM module general image and image plane, respectively. They are used to store frequently-used DICOM tags such as patient orientation, pixel spacing, and image position. The second image reference type is *WebImageReference* that contains a URI to an image.

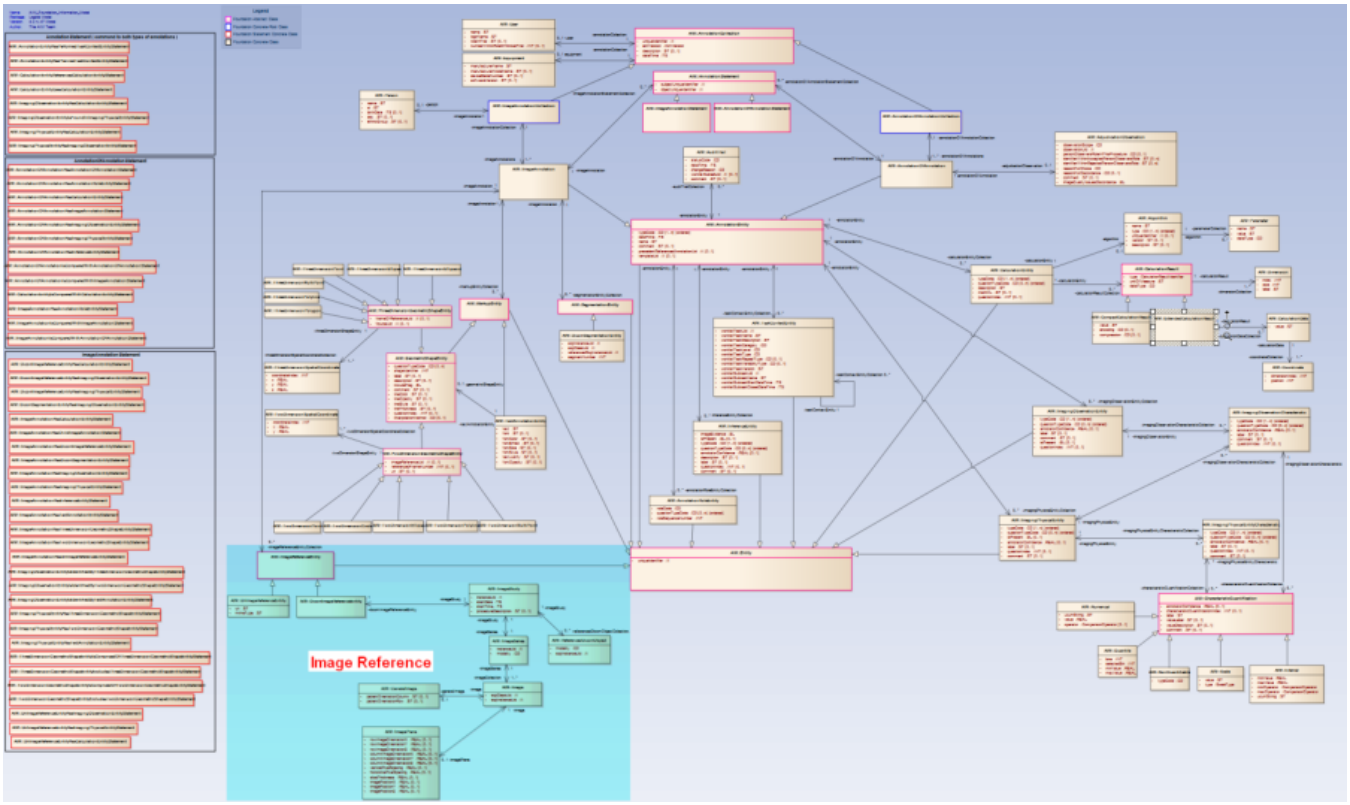

Figure 9. ImageReference Group

Note that abstract classes in the AIM schema are *AnnotationCollection*, *AnnotationStatement*, *ImageAnnotationStatement*, *AnnotationOfAnnotationStatement*, *AnnotationEntity*, *CharacteristicQuantification*, *CalculationResult*, *Entity*, *GeometricShapeEntity*, *ImageReferenceEntity*, *TwoDimensionGeometricShapeEntity*, *ThreeDimensionGeometricShapeEntity*, *MarkupEntity*, *SegmentationEntity* and *SpatialCoordinate*. [0..1] denotes an optional occurrence of an attribute.

How to Create and Use AIM Annotations

AIM annotations can only contain either one or more image annotation or annotation-of-annotation type. An image annotation is represented by the *ImageAnnotation* class. The class is used to annotate images. Annotation-of-annotation is represented by *AnnotationOfAnnotation* class. The class is used to annotate other AIM annotations for comparison and reference purposes. Each class inherits all properties of the abstract class, *AnnotationEntity*.
There are two root classes that can contain either *ImageAnnotation* or *AnnotationOfAnnotation*; namely *ImageAnnotationCollection* and *AnnotationOfAnnotationCollection* respectively. These are the two kinds of annotation collections that can be instantiated. An instance of *ImageAnnotationCollection* class contains one or more instances of related *ImageAnnotation* class. An instance of *AnnotationOfAnnotation* class may contain a collection of instances of *ImageAnnotation* and/or *AnnotationOfAnnotation* class.

Both annotation collections may contain information about the equipment (*Equipment* class) used to create an AIM instance and the user (*User* class) who created the AIM instance.

An instance of *ImageAnnotation* object has annotation and markup information on one or more images in the same series and study. Annotation and markup information describes particular findings of a single thing found on an image or images. For instance, if there are two nodules found on an axial image, two *ImageAnnotation* instances must be created. Image markups of the same nodule put on different images in the same study can be captured in a single *ImageAnnotation* instance. *ImageAnnotation* must have one or more *ImageReference* objects, which can be either DICOM image objects or URI (Uniform Resource Identifier) image objects. *ImageAnnotation* may be associated with one instance of a *Person* class. *ImageAnnotation* may have zero or more *AuditTrail*, *SegmentationEntity*, *MarkupEntity* (e.g. text, two-dimension geometric shape or three-dimension geometric shape), *ImagingPhysicalEntity* (e.g. anatomic entity), *ImagingObservationEntity*, *InferenceEntity*, *CalculationEntity*, *TaskContextEntity*, and *AnnotationRoleEntity* instances.

An *AnnotationOfAnnotation* has annotation information about one or more AIM annotations, which can be *ImageAnnotation* or *AnnotationOfAnnotation*. It is used for the purpose of comparison, reference, and annotating additional information to existing AIM *ImageAnnotation* or *AnnotationOfAnnotation* instances. An example of this type of annotation can be used to compare measurement results of the same tumor from two studies from timepoint one and two of the same patient. *AnnotationOfAnnotation* may have zero or more *AuditTrail*, *ImagingPhysicalEntity* (e.g. anatomic entity), *ImagingObservationEntity* , *InferenceEntity*, *CalculationEntity*, *TaskContextEntity*, *AnnotationRoleEntity* and *AdjudicationObservation* instances.

An *AnnotationOfAnnotation* has references to instances of *ImageAnnotation* or *AnnotationOfAnnotation* via two kinds of AIM statements, namely *AnnotationOfAnnotationHasAnnotationOfAnnotationStatement* and *AnnotationOfAnnotationHasImageAnnotationStatement*.

How to Extend the AIM Foundation Model

The previous section describes the AIM foundation in detail. This section provides information used to extend the AIM foundation model to store other information that is not currently available in the foundation model. As described in sections 4 and 5, the AIM information model uses UML to depict the type of information it is currently capable of collecting. The model explicitly describes how image annotation semantic contents can be stored. A class name in the model was intentionally created to inform a reader of what information is stored in the class. Each attribute in a class supports the class purpose. A relationship between classes indicates how one class works with other classes to form and capture a larger concept and information.

Guidelines to the AIM Foundation Model

This section provides a set of guidelines to extend the AIM Foundation Model. The foundation model conceptually orders classes in the AIM UML model into six groups: AIM Statements, General Information, Calculation, Image Semantic Content (Finding), Image References, and Markup. You may extend the foundation model to fit your own needs by following the criteria. If you want your additions to be included in the official NCI AIM model, please enter your request directly at https://tracker.nci.nih.gov/browse/AIM.

1. Naming Convention:
    a. A name of a class must explicitly describe what information the class will collect. It must start with a capital letter. If part of the class name has a capital abbreviation, only the first character of the abbreviation is capitalized, e.g. DICOM should be Dicom see figure 10.

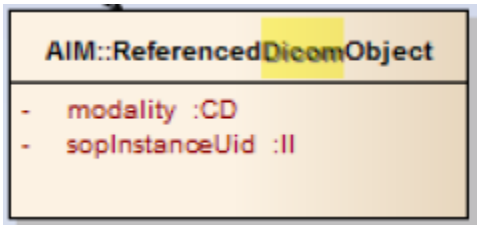

Figure 10. A Class Name

Association name, figure 11, of a source class has the same name as the source class name with the first character being a lower case.

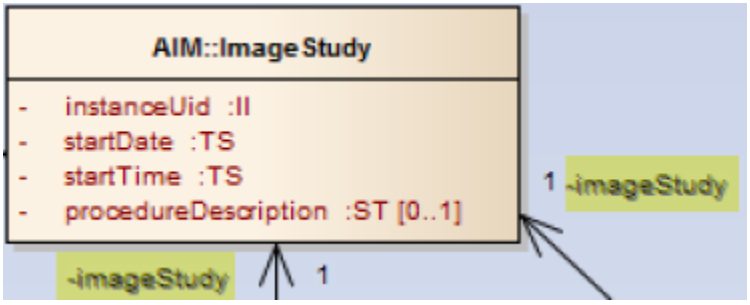

Figure 11. An Association Name of a Source Class

Association name of a target class, figure 12, has the same name as the target class name with the first character being a lower case.

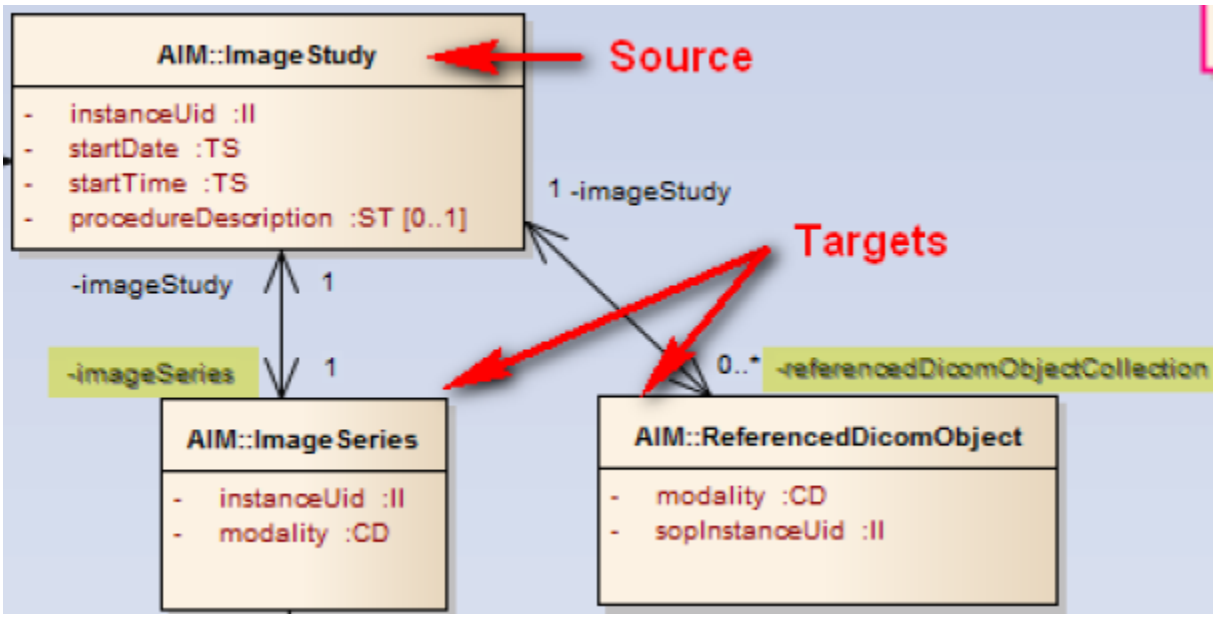

Figure 12. An Association Name of a Target Class

- A source class may have or contain 1-to-0..* (zero-to-many) or 1-to-1..* (one-to-many) associations to a target class. The target association name must append "Collection", figure 13, at the end of the class name.

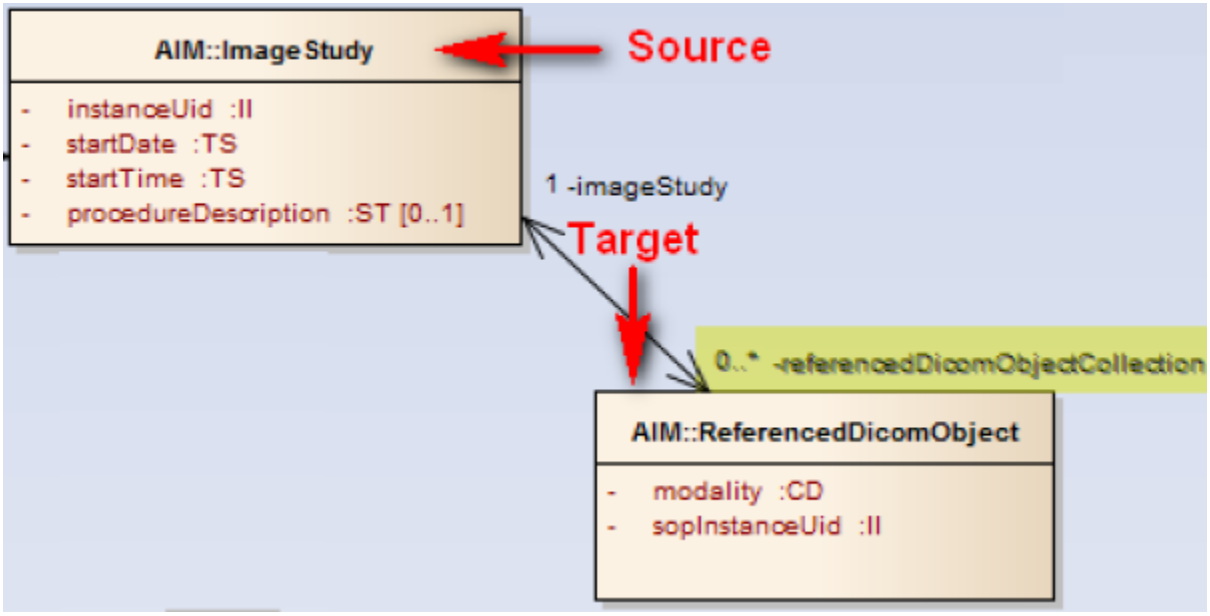

Figure 13. An Example of Collection

1. It must be possible to map a new class to DICOM SR data element(s). See "AIM DICOM SR templates" for more information. It is important for a user to have knowledge about DICOM SR and how to modify a DICOM SR template properly because new classes added to the model will need to be stored in DICOM format.
2. Use ISO 21090 data type for every attribute in a class. Using ISO 21090 data type provides a good foundation to convert AIM information to HL7 Clinical Document Architecture (CDA).
3. A name of any class extending from the *Entity* class must end with the word "Entity". A class derived from the *Entity* class represents an existence of a thing or concept that is not currently captured in the AIM foundation model. A new entity class can also be included if a user wants to explicitly express what information the new model will be able to store.
4. A class that can be used to construct an AIM statement must extend or inherit from the *Entity* class.
    a. There must be a subject and object class.
    b. A predicate must be selected from a list of existing predicate from section 4.b.4. You may use your own predicates.
    c. A name of an AIM statement shall be a concatenation between the name of a subject class, predicate and the name of object class.
5. AIM markup is modeled after DICOM 2D and 3D spatial coordinate geometries [2]. New additional markups must be able to be stored in DICOM SR format.

Extending AIM Foundation Model to AIM 4.0

AIM 4.0 model, Figure 14, is an official extension of the AIM foundation model. This extension explicitly captures lesion results and measurements derived during image-based clinical trials [7,8]. There are nine new classes that cover lesion annotation needs. Three classes were created for lesion observation as follows.

- *LesionObservationEntity*

The class is an abstract class that stores observations made about lesions in both clinical trial and day-to-day clinical treatments. For detailed information, see *DICOM Clinical Trials Results Reporting Supplement (Working group 18)*.

- *GeneralLesionObservationEntity*

This class contains general observations made about lesions in clinical trial results and day-to-day clinical treatment that are not specific to timepoint. For detailed information, see *DICOM Clinical Trials Results Reporting Supplement (Working group 18)*.

- *TimePointLesionObservationEntity*

This class contains observations made about lesions in day-to-day clinical interpretations and clinical trial results at a specific timepoint. It also includes "lesions" that are created for the purpose of calibrating scanned film or other secondary capture images. For detailed information, see *DICOM Clinical Trials Results Reporting Supplement (Working group 18)*.

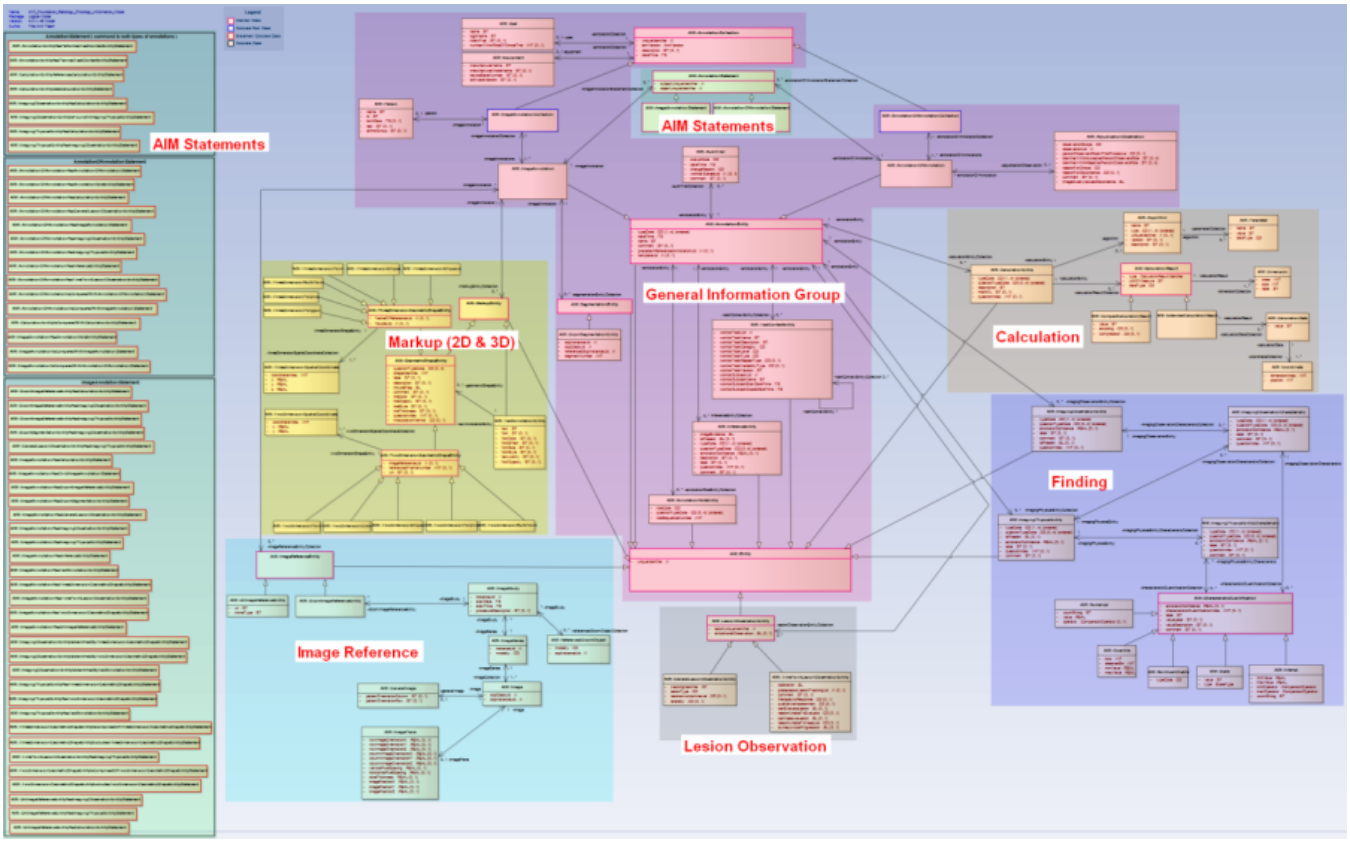

Figure 14. AIM 4.0 Model

Six classes were created for AIM statements as follows.

- *AnnotationOfAnnotationHasGeneralLesionObservationEntityStatement*

An instance of annotation-of-annotation may have one or more general lesion observations associated with it. *AnnotationOfAnnotationHasGeneralLesionObservationEntityStatement* represents a direct relationship between an instance of annotation-of-annotation and general lesion observation. If you have two general lesion observations, you will need to create two statements.

A use case: An adjudicator wants to create a general lesion observation statement from an annotation-of-annotation.

Assumption:

- An annotation-of-annotation, each with general lesion observation, was created earlier from a reader.
- There is a system capable of reading and extracting information from the annotation for further displaying, computing and manipulating purposes.

Working with AIM:

1. Create a general lesion observation instance.
2. Create an *AnnotationOfAnnotationHasGeneralLesionObservationEntityStatement* statement linking the annotation-of-annotation (subjects) to the general lesion observation (objects).

- *AnnotationOfAnnotationHasTimePointLesionObservationEntityStatement*

An instance of annotation-of-annotation may have one or more timepoint lesion observations associated with it. *AnnotationOfAnnotationHasTimePointLesionObservationEntityStatement* represents a direct relationship between an instance of annotation-of-annotation and timepoint lesion observation. If you have two timepoint lesion observations, you will need to create two statements.

A use case: An adjudicator wants to create a timepoint lesion observation statement from an annotation-of-annotation.

Assumption:

1. An annotation-of-annotation, each with timepoint lesion observation, was created earlier from a reader.
2. There is a system capable of reading and extracting information from the annotations for further displaying, computing and manipulating purposes.

Working with AIM:

1. Create a timepoint lesion observation instance.
2. Create an *AnnotationOfAnnotationHasTimePointLesionObservationEntityStatement* statement linking the annotation-of-annotation (subjects) to the timepoint lesion observation (objects).

- *GeneralLesionObservationEntityHasImagingPhysicalEntityStatement*

The class is used to record a relationship between a general lesion observation entity and imaging physical entity. Each lesion observation can only be directly related to one imaging physical entity.
A use case: An imaging interpreter wants to link a general lesion observation and imaging physical annotation.

Working with AIM:

- Create an imaging physical entity instance.
- Create a general lesion observation entity instance.
- Create a *GeneralLesionObservationEntityHasImagingPhysicalEntityStatement* statement linking the general lesion observation entity (subjects) to the imaging physical entity (objects).
- *ImageAnnotationHasGeneralLesionObservationEntityStatement*

An instance of image annotation may have one or more general lesion observations associated with it. *ImageAnnotationHasGeneralLesionObservationEntityStatement* represents a direct relationship between an instance of image annotation and general lesion observation. If you have two general lesion observations, you will need to create two statements.

A use case: An adjudicator wants to create a timepoint lesion observation statement from an image annotation.

Assumption:

1. An image annotation, each with timepoint lesion observation, was created earlier from a reader.
2. There is a system capable of reading and extracting information from the annotation for further displaying, computing and manipulating purposes.

Working with AIM:

- Create a timepoint lesion observation instance.
- Create an *ImageAnnotationHasTimePointLesionObservationEntityStatement* statement linking the annotation-of-annotation (subjects) to the time-point lesion observation (objects).
- *ImageAnnotationHasTimePointLesionObservationEntityStatement*

An instance of image annotation may have one or more timepoint lesion observations associated with it. *ImageAnnotationHasTimePointLesionObservationEntityStatement* represents a direct relationship between an instance of image annotation and time-point lesion observation. If you have two general lesion observations, you will need to create two statements.

A use case: An adjudicator wants to create a timepoint lesion observation statement from an image annotation.

Assumption:

1. An image annotation, each with timepoint lesion observation, was created earlier from a reader.
2. There is a system capable of reading and extracting information from the annotation for further displaying, computing and manipulating purposes.

Working with AIM:

- Create a timepoint lesion observation instance.
- Create an *ImageAnnotationHasTimePointLesionObservationEntityStatement* statement linking the image annotation (subjects) to the timepoint lesion observation (objects).
- *TimePointLesionObservationEntityHasImagingPhysicalEntityStatement*

The class is used to record a relationship between a timepoint lesion observation entity and imaging physical entity. Each lesion observation can only be directly related to one imaging physical entity.
A use case: An imaging interpreter wants to link a timepoint lesion observation and imaging physical annotation.

Working with AIM:

1. Create an imaging physical entity instance.
2. Create a timepoint lesion observation entity instance.
3. Create a *TimePointLesionObservationEntityHasImagingPhysicalEntityStatement* statement linking the timepoint lesion observation entity (subjects) to the imaging physical entity (objects).

You can download the AIM foundation and 4.0 models from https://wiki.nci.nih.gov/x/z4X3Ag.

AIM Software Toolkit

Based on the above model, an AIM programming library has been constructed to create, validate, and transform between AIM XML documents and DICOM SR. Implementors should familiarize themselves with the AIM schema and be able to deduce relationships between classes in the schema. The library is a set of APIs. It is independent from the Graphical User Interface (GUI) and workflow of an application. All interactions with the AIM library are done through AIM library APIs. In addition to the AIM library, the application would need to provide for:

- creating, displaying and converting application markups to AIM markups
- presenting and collecting AIM annotation of image findings
- capturing and displaying annotation information
- computing and translating its computation to AIM calculations.

The AIM software library is a collection of C++ application programming [10] interfaces used to construct the AIM information model based on a UML class diagram. ANIVATR is a referenced implementation of the AIM library. The ANIVATR software application validates AIM annotations and transcodes them into different artifacts, namely native AIM XML and DICOM SR. We have developed AIM annotations in DICOM SR such that they can be created and displayed in a variety of medical imaging workstations, notably the AIM on ClearCanvas workstation and eXtensible Imaging Platform (XIP) as well as in clinical imaging devices. ANIVATR reads and transcodes DICOM SR into an AIM XML representation and vice versa.

AIM Library

The AIM library is a C++ [9] module. It consists of two logical parts: implementation of the AIM Schema as an object model and definition of a set of operations, which can be performed on the object model.

The AIM library can be used as a linked dependency of another application. All exported library APIs are thought to conform to ANSI C++ (1998/2003). Various STL containers are used extensively throughout the library and in the public APIs.

The object model implementation creates an ANSI C++ class for each class in the AIM Schema. The Class hierarchy closely follows the AIM Schema. Each object's model class provides mutation methods (Set and Get method) for every attribute in the corresponding AIM Schema class. All changes to the class states are done via those mutation methods. The whole AIM Schema is represented by the object model through containment and inheritance. The set of object model operations supported by the AIM library includes persisting the model in XML and DICOM SR formats. The reverse set of operations of reading XML and DICOM SR instances into the object model is supported as well.

Using AIM Library

To instantiate an AIM model, a software developer should start with creating either an *AnnotationOfAnnotationCollection* or *ImageAnnotationCollection* object and populating its related objects' content. All required attributes need to be populated with valid data. Optional attributes are depicted in the AIM schema with [0..1].

ImageAnnotation

The *ImageAnnotationCollection* object is required to have at least one *ImageAnnotation* object of type *DICOMImageReferenceEntity* or *WebImageReferenceEntity*. The *DICOMImageReferenceEntity* object must have one imaging study. The *ImageStudy* object may have one series objects. Each *ImageSeries* object may have one or more *Image* objects. *It is implied in the model that all images originate from the same study of the same patient.*

ImageAnnotationCollection object may have a *Person* object. An *ImageAnnotation* object may have *DicomSegmentationEntity* objects, which contain references to its own instance UID and referenced instance UID of the image to which the segmentation is applied. It also has the segmentation type, DICOM SOP class, UID, and an identification number of the segment. The identification of the segment shall be unique within the segmentation instance in which it is created. *ImagingObservationEntity* object is being captured as DICOM code sequence with a possible textual comment. An *ImagingObservationEntity* may have zero or more *ImagingObservationEntityCharacteristic* objects, which are captured as DICOM code sequences. An *ImageAnnotationEntity* may store conclusions derived by interpreting an imaging study and/or medical history in a collection of *InferenceEntity* objects that store the information as code sequence based on a controlled terminology.

ImageAnnotationEntity object may have zero or more *TextAnnotationEntity* objects. Each *TextAnnotationEntity* object may have a two or three-dimensional Cartesian coordinate set defined as a MultiPoint type object. *TextAnnotationEntity* is used as a text markup that can be shown on an image. Graphic markups are stored as *TwoDimensionGeometricShapeEntity* and *ThreeDimensionGeometricShapeEntity* objects, which extended from *GeometricShapeEntity*. Two dimension graphic types are MultiPoint, Point, Circle, Ellipse, and Polyline objects. These inherit the *TwoDimensionGeometricShapeEntity* abstract class properties and methods. Each two-dimensional graphic type contains one or more *TwoDimensionSpatialCoordinate* instances. Three-dimensional graphic types are Point, MultiPoint, Polyline, Polygon, Ellipse, and Ellipsoid. These objects inherit the *ThreeDimensionGeometricShapeEntity* abstract class properties and methods. Each three-dimensional graphic type contains one or more *ThreeDimensionSpatialCoordinate* instances. The *coordinateIndex* attribute in *TwoDimensionSpatialCoordinate* or *ThreeDimensionSpatialCoordinate* class signifies the order in which a coordinate appears in the shape. *GeometricShape* class closely follows DICOM 3.0 part 3, C.18.6.1.2 and C.18.9.1.2 Graphic Type. *TwoDimensionSpatialCoordinate* contains SOP Instance UID and frame number (multi-frame image) to identify which image a geometric shape object belongs to.

ImageAnnotationCollection inherits *AnnotationCollection* that may have at most one Equipment and User objects. *ImageAnnotation* inherits properties and methods from the *AnnotationEntity* class. It may have zero or more *ImagingPhysicalEntity*, *ImagingObservationEntity* and *CalculationEntity* objects.

A *CalculationEntity* object can be related to a single markup or to a collection of markups and other calculations, which are not related to markup. A calculation may reference other calculations by using *CalculationEntityReferencesCalculationEntity* and *CalculationEntityUsesCalculationEntity* objects, which contain a referenced *CalculationEntity* object UID. A *CalculationEntity* object may have zero or more *CalculationResult* objects. It is possible for a Calculation to have no *CalculationResult*. This means that the information provided in the *CalculationEntity* object is sufficient to describe the calculation.

A calculation result can be a scalar, vector, matrix, histogram, or array. Dimensionality of calculation results is represented by *Dimension* objects. A *CalculationResult* object must have at least one *Dimension* object. The *Index* attribute in the *Dimension* object is a zero based unique index of the dimension. The *Size* attribute in the *Dimension* object specifies how many members a dimension has. Label attribute provides textual meaning to a dimension.

A *CalculationResult* object may have zero or more *Data* objects. The absence of any *Data* object means that result is an empty set. Each *Coordinate* object specifies a dimension index and a position within the dimension. The number of *Coordinate* objects for each *Data* object cannot exceed the total number of *Dimension* objects in a *CalculationResult*. A *Data* object cannot have more than one *Coordinate* object with the same *dimensionIndex*.

AnnotationOfAnnotation

The *AnnotationOfAnnotationCollection* object is required to have at least one *AnnotationOfAnnotation* object. *AnnotationOfAnnotation* works very much the same way as *ImageAnnotation* for calculation group and image semantic content group (see section 7.c). The *AnnotationOfAnnotation* object must have at least one AIM statements that contains a UID of *ImageAnnotation* or *AnnotationOfAnnotation* object. *AnnotationOfAnnotation* may store a conclusions derived by interpreting an imaging study and/or medical history in a collection of Inference object, which stores the information as a code sequence based on a controlled terminology.

AnnotationOfAnnotation may refer to a collection of *ImageAnnotation* objects that can come from different studies.

The AIM model and DICOM templates do not explicitly address the issue of Study Instance UID, Series Instance UID and SOP Instance UID creation of an *AnnotationOfAnnotation* object. These three UIDs can be generated by AIM implementers for the purpose of creating an AIM DICOM object. When an AIM DICOM object is transformed to AIM XML or HL7 CDA, these three UIDs are not being used.

AIM Library Objects

AIM library is created using C#. The library objects and operations are in the aim_lib namespace. All object model files are located in the model/ sub-directory. For convenience purposes, there is a model/AimHeaders.h header file that includes all headers of the object model. Below is a sample of how one can populate parts of the object model.

```
#include "AIMLib/model/AimHeaders.h"
……
// Calculation and its dependents
aim_lib::Dimension dim;
dim.SetIndex(0);
dim.SetLabel("Centimeters");
dim.SetSize(1);
aim_lib::DimensionVector dimColl;
dimColl.push_back(dim);
aim_lib::CalculationResult calcResult;
calcResult.SetUnitOfMeasure("cm");
calcResult.SetType("CalculationResultTYpe::Scalar");
calcResult.SetNumberOfDimensions(1);
calcResult.SetDimensionCollection(dimColl);
aim_lib::Coordinate coordinate;
coordinate.SetDimensionIndex(0);
coordinate.SetPosition(0);
aim_lib::CoordinateVector coordColl;
coordColl.push_back(coordinate);
aim_lib::Data data;
data.SetValue(150.0);
data.SetCoordinateCollection(coordColl);
aim_lib::DataVector dataColl;
dataColl.push_back(data);
calcResult.SetDataCollection(dataColl);
aim_lib::CalcResultVector calcResults;
calcResults.push_back(calcResult);
aim_lib::ReferencedCalculation refCalc;
refCalc.SetReferencedCalculationUID("1.23.5698.24546.231365.74654");
aim_lib::ReferencedCalcVector refCalcs;
refCalcs.push_back(refCalc);
aim_lib::Calculation calc;
calc.SetUID(AimUidGenerator::GenerateNewUID("33.333"));
calc.SetCodeValue("Value::Length");
calc.SetCodeMeaning("length");
calc.SetCodingSchemeDesignator("CALC_SCHEME");
calc.SetDescription("Description of the Calculation One - Length");
calc.SetMathML("MathML for the Calc One goes here");
calc.SetCalculationResultCollection(calcResults);
calc.SetReferencedCalculationCollection(refCalcs);
calculations.push_back(calc);
```

AIM Library Operations

Operations provided by the AIM library reside in the operations/ folder of the library. All operations are performed on the object model via *DCMModel* and *XMLModel* classes.
AIMLib/operations/DCMModel.h contains operations for DICOM SR.

To read an AIM library object from the DICOM SR file, use one of the two available APIs:

1. *ReadAnnotationsFromFile* / *GetNextAnnotation* pair will read all available annotations from a file and will make the annotations available as an object model one by one.
2. *ReadAnnotationFromFile* will read a single annotation object from a file.

Two more APIs are available to write AIM library object(s) to a DICOM SR file:

- *WriteAnnotationToFile* / *WriteAnnotationsToFile* will serialize AIM library object(s) into a DICOM SR.

AIMLib/operations/XMLModel/.h contains operations for AIM XML.
To read an AIM XML file into an AIM library object use:

1. *ReadAnnotationsFromFile* / *GetNextAnnotation* pair will read all available annotations from a file and will make the annotations available as an object model one by one.
2. *ReadFromXmlFile* will read a single annotation object from a file.
3. *ReadFromXmlString* will read a single annotation object from a string buffer.

To write an AIM library object to an AIM XML file use:

1. *WriteAnnotationToFile* / *WriteAnnotationsToFile* will write AIM library object(s) to file.
2. *WriteAnnotationToString* / *WriteAnnotationsToString* will write AIM library object(s) to a string buffer.

Environment Configuration

The AIM development environment on Windows systems requires a few configurations. An example of how an environment may be set up follows. Environment variables:

JAVA_HOME=c:\jdk6
BOOST_ROOT=C:\Program Files\boost\boost_1_34_1

Sample Code

The AIMTestLib project contains AIMLibTest/AIMLibTest.cpp, which shows examples of how to use AIM Library for generating/reading/writing AIM library objects and files.

References

1. Channin, D. S., Mongkolwat, P., Kleper, V., Sepukar, K. Rubin, D. L., The caBIG^{TM} Annotation and Image Markup Project. Journal of Digital Imaging, Vol 23, No. 2, April, 2010.
2. Annotation and Image Markup Version 3 Project: Requirements, Design, Implementation and Usage; September 2010, URL: https://ncisvn.nci.nih.gov/svn/files/trunk/aim/aim/AIMToolkit3.0.2/AIMToolkit_v3.0.2_rv11.rar Access 2013-09-30.
3. Rubin, D. L., Mongkolwat, P., Kleper, V., Supekar, K., Channin, D. S., Medical Imaging on the Semantic Web: Annotation and Image Markup, Association for the Advancement of Artificial Intelligence, 2008 Spring Symposium Series, Stanford, CA, 2008
4. Channin D. S., Mongkolwat P., Kleper V, Rubin D. L., The Annotation and Image Mark-up Project: Radiology. 2009; 253:590–592
5. Clunie, D. A., DICOM Structured Reporting, PixelMed Publishing, 2000.
6. DICOM 2011. Digital Imaging and Communications in Medicine (DICOM) [online] URL: http://medical.nema.org/standard.html Accessed 2013-09-30.
7. Clunie, D. A., DICOM Structured Reporting and Cancer Clinical Trials Results. Cancer Informatics, pp. 33-56, 2007.
8. Clunie, D. A., Clinical Trials Results Reporting, unpublished DICOM supplemental from Working Group 18, http://www.dclunie.com Access 2013-09-30.
9. C++: URL: {+} http://www.cplusplus.com/+ Access 2013-09-30.
10. AIM Toolkit: URL: https://wiki.nci.nih.gov/display/AIM/Annotation+and+Image+Markup+-+AIM#AnnotationandImageMarkup-AIM-AIMToolkit Access 2013-09-30.