WHITE PAPER

The Annotation Exchange Format for Images (AXFi)

Overview

Image annotation and segmentation is an important analytic process in many scientific, technical, and commercial fields. Nonetheless, there are few standard formats for describing and representing image annotations, and those which do exist tend to be used in specific, relatively narrow contexts.[1] This is not a new observation; Daniel L. Rubin *et. al.*, in 2007, note that:

> Images contain implicit knowledge about anatomy and abnormal structure that is deduced by the viewer of the pixel data, but this knowledge is generally not recorded in a structured manner nor directly linked to the image. [Moreover,] the *terminology* and *syntax* for describing images and what they contain varies, with no widely-adopted standards, resulting in limited interoperability. The contents of medical images are most frequently described and stored in free-text in an unstructured manner, limiting the ability of computers to analyze and access this information. There are no standard terminologies specifically for describing medical image contents — the imaging observations, the anatomy, and the pathology. [N]o comprehensive standard appropriate to medical imaging has yet been developed. A final challenge for medical imaging is that the particular information one wants to describe and annotate in medical images depends on the *context* — different types of images can be obtained for different purposes, and the types of annotations that should be created (the "annotation requirements" for images) depends on that context. For example, in images of the abdomen of a cancer patient (the context is "cancer" and "abdominal region"), we would want annotations to describe the liver (an organ in the abdominal region), and if there is a cancer in the liver, then there should be a description of the margins of the cancer (the appearance of the cancer on the image). (http://cedarweb.vsp.ucar.edu/wiki/images/d/d9/R_19.pdf, pp. 1-2).

These challenges inspired **AIM** (the "Annotation and Image Markup" project), which "provides a solution to the ... imaging challenges [of]: No agreed upon syntax for annotation and markup; No agreed upon semantics to describe annotations; No standard format ... for annotations and markup" (https://wiki.nci.nih.gov/display/AIM/Annotation+and+Image+Markup+-+AIM). However, **AIM** itself has not been widely adopted outside the specific field of cancer research and cancer-oriented image repositories.

One obstacle to formalizing image-annotation data is that annotations have a kind of intermediate status, neither intrinsic parts of an image nor merely visual cues supporting the presentation of the image within image-viewing software. Many applications exist which allow markup or comments to be introduced with respect to an image. From the application's point of view, these annotations are part of the application display, not part of the image — analogous to editing comments that might be added to a text document by a word processor or **PDF** viewer, which are records of user actions, not intrinsic to the document itself. Indeed, one mechanism for recording image annotations in **DICOM** (the "Digital Imaging and Communications in Medicine" format) is via "presentation state." The presentation state includes all details about how the image currently appears to **DICOM** workstation users, such as Radiologists, which could include markings they have made to indicate diagnostically significant image regions or features. Insofar as image-annotations are considered to be artifacts

[1] Current formats include **AIM** (Annotation and Image Markup), **CVAT XML** (**CVAT** is the Computer Vision Annotation Tool), **DICOM-SR** (Digital Imaging and Communications in Medicine Structured Reporting), **PASCAL VOC XML** (Pattern Analysis, Statistical Modeling and Computational Learning Visual Object Classes), and **COCO JSON** (Common Objects in Context).

of image-viewing software, rather than significant data structures in their own right, there is less motivation for imaging applications to support canonical annotation standards.

Nevertheless, in many scientific and technical areas image annotations *are* significant; they are intrinsic to the scientific value of a given image as an object of research or observation. Image regions, segments, and features have a semantic meaning outside the contexts of the applications that are used to view the corresponding images, which is why it is important to develop cross-application standards for describing and affixing data to image annotations.

It is also important for image-annotation models to be broadly applicable and multi-disciplinary. While image analysis serves different goals in different contexts (e.g., segmentation of microscope images to detect cancer cells serves different ends than segmentation of camera snapshots to study traffic patterns), there is always a possibility of analytic techniques developed in one subject area to be applicable for other image-processing problems, even if the practical outcomes desired of the analyses are very different. Furthermore, certain computational domains are similar enough to image analysis to warrant inclusion in a general-purpose image-annotation framework, even if the underlying data is not "images" in the conventional sense (not, for instance, captured via photographs or microscopy). For example, **PDF** document views, Flow Cytometry (**FCM**) data plots, and geospatial maps subject to Geographic Information Systems (**GIS**) annotations may all be considered images — by virtue of a semantic significance attributed to color and to geometric primitives as a way of characterizing phenomena observed or modeled through their data — even though such resources are not acquired by ordinary "image-producing" devices. The "Pantheon" project, characterized as a "platform dedicated to knowledge engineering for the development of image processing applications,"[2] offers one of the few attempts in imaging literature to rigorously define "imaging" and "image processing" in the first place. Pantheon (via its **PANDORE** component) also includes an image processing *objectives* ontology, which is partially incorporated into **AXFi**, so Pantheon's articulation of the general domain of imaging *per se* applies to **AXFi** as well. The problem of defining "images" as such, and therefore delineating the scope of image annotation, is addressed below (page 9). In brief, **AXFi** considers the realm of imaging to be more general than just graphics obtained by a direct recording of the optics of some physical scene via cameras, microscopes, or telescopes. That is to say, the image acquisition process is not necessarily one where data is generated by an instrument which produces a digital artifact by absorbing light, so that geometric and chromatic properties of the image are wholly due to the functioning of the acquisition device.

Systematically identifying the scope of "image annotation" is important for **AXFi** because doing so clarifies the sorts of domains whose semantics could reasonably be incorporated into **AXFi**, as well as the sorts of applications which would be reasonable candidates for supporting **AXFi** annotations (i.e., the capability to parse **AXFi** data and represent it vis-à-vis the relevant images). For example, if immunoflourescent Flow Cytometry (**FCM**) data plots are classified as images, then the numerical properties of the "channel" axis, with notions of "decades" and a "log/linear" distinction, become relevant to the **AXFi** vocabulary for representing spatial dimensions and magnitudes. In general, **AXFi** uses paradigms and terminology from "Conceptual Space Theory" as part of the process of formalizing geometric and dimensional notions.[3]

When defining the scope of **AXFi**, it is also important to distinguish the *data models* encapsulated by **AXFi** resources from the file formats where **AXFi** data is encoded. The choice of one file type or another to represent a data structure — **XML** versus **JSON**, for instance — does not fundamentally affect the data thereby communicated. Therefore, it is important to formalize data models in such a way that numerous different serialization languages might actually be used to share/express the data. However, in practice, format-specific standards, such as **XML** Schema Definitions, are often used as the basis for formalizing and enforcing compliance with data models. Therefore, **AXFi** cannot be completely unconcerned with the structure and requirements of files which convey **AXFi** data. This

[2] See https://hal.archives-ouvertes.fr/hal-00260065/document.
[3] See http://idwebhost-202-147.ethz.ch/Publications/RefConferences/ICSC_2009_AdamsRaubal_Camera-FINAL.pdf or https://arxiv.org/pdf/1801.03929.pdf.

is particularly true because AXFi seeks to incorporate the data models of other specifications, such as AIM and GATING-ML, which are formalized via XML specifications. Although AXFi is not primarily XML-based, in short, it intends to be (in the relevant contexts) compatible with XML languages that rely on XML schematization. Further details on how AXFi manages the relation to XML and other serialization languages are provided below (page 14).

A further detail that should be clarified prior to expositing a formal outline of AXFi is that of how image-annotations originate. Sometimes, of course, annotations are manually introduced on images by human users of image-viewing software. On the other hand, automated image segmentation — or similar algorithmic or AI-driven image processing without human intervention — yields partitions of images into regions, or identification of semantically important locations in an image, therefore generating annotations computationally. In short, AXFi should support both human-generated and computer-generated annotations. This becomes complicated, however, insofar as image-processing may yield analyses which overlap with annotation objectives but may not intrinsically produce annotations in the conventional sense. For example, an algorithm to count the number of cars in a highway picture may rely on statistical analysis of some quantitative image feature — such as "zero-crossings" — without in fact producing determinate image segments.

It is important to remember that image processing operations do not always act directly on images themselves; sometimes algorithms are based instead on mathematical complexes derived from the image, but with their own quantitative properties. For instance, color-valued pixels may be replaced by matrices measuring the gradient of some image-feature field in eight directions around each point (an example would be "Sobel kernels" applied to the image intensity function). For a given "semantic" task — that is, an image-processing objective whose end-result is not just image-related data but some empirical observation — image segmentation, or other analyses yielding annotations, are a means to an end: one *way* to count cars is to delineate the edges of distinct cars in distinct segments, and then count the number of segments which result. However, statistical image-analysis may produce largely accurate results for such semantic tasks, given large image corpora (e.g., estimating traffic flows from highway cameras), without yielding artifacts such as human-visible segment representations. Or, in a different domain, AI-powered analysis of FCS (Flow Cytometry Standard) data could establish a largely accurate count of "events" (i.e., discrete FCS measurements of light-scattering and/or flourescent properties of cellular-scale entities) without manual "gating" (referring to the conventional practice of scientists using geometric annotations of FCS data-plots to isolate and thereby count different event-types). An AI-powered analysis of image features, or likewise of Flow Cytometry data, may yield calculations similar to those which for *human* users are achieved via image segmentation, manual gating, and similar operations which clearly yield annotation data. For AXFi, the complication arises when these AI mechanisms do not themselves yield results that would normally be considered annotations, but rather yield the desired empirical results for which the annotations would be a preliminary step — e.g., an approximate count of the number of cars in a highway photo, without a precise segmentation of the image marking the cars' respective borders.

AXFi ultimate considers these AI-related complications to be issues resolvable at the level of software, rather than standardized annotation models *per se*. An image annotation is, among other things, a visual (or viewable) record of some image-processing activity. If a radiologist manually clarifies a report to the effect that a given CT shows a tumor by circling the area where the tumor is visible, he or she is using the image annotation to communicate to others the thought-process which motivated the diagnostic conclusion. This is different than an AI-driven processor which would automatically demarcate an image segment outlining the tumor and use geometric properties of that segment to derive a pathological finding. In short, the data conveyed in an annotation — an image segment, rendered precisely, or rendered indirectly via a circle or polygon around the segment — may be *intrinsic* to an image-processing operation: it may be data acquired *at one stage* in an analytic workflow. However, annotations may also be *retroactive*; if a radiologist circles a tumor, he or she has completed (at least mentally) the image analysis, and is using the analysis to summarize what occurred in the course of the analysis. Therefore, any image-processing task can be associated with

ex post facto annotations which summarize the process even if they are not intrinsic to it.

To continue the example of counting cars from a highway photo, an **AI**-powered observation could be retroactively *justified* by providing an image segmentation where the number of car-segments matches the **AI** count. In lieu of precise segments, it may be simpler to provide location-points for the "geometric center" of each car, or the points furthest apart in the direction of each car's front-to-back — these may be the statistical signals used for the car-counting process. Analogously, facial recognition does not need to rely one segmenting out regions (eyes, nose, lips), but rather can be based on distances between individual points (such as the inner corners of each eye). But in any case, depending on the analytic algorithm used, it is often possible to identify some spatial/geometric feature or object that can be visualized in the image context, and that summarizes or legitimizes the analytic operation. This summarial data, then, can provide *retrospective* annotations which allow human viewers to understand and review the algorithmic process. In short, simply because image-processing tasks may not generate annotation data as part of their internal activity, it is still possible (and may be desirable) for the software operationalizing these tasks to implement annotation generators, where the resulting annotations document the operations for the scientific record and/or summarize them in **GUI** objects for the benefit of human viewers.

In general, then, **AXFi** distinguishes human-generated from computer-generated annotations, and moreover leaves open the possibility that some computer-generated annotations are *retrospective*: that instead of being internal to an imaging computation they are indirectly produced subsequent to such a computation, for purposes of documentation and validation. Annotations which are not *retrospective* are called *internal*, as in, internal to a given image-processing workflow.

Related to the distinction between *internal* and *retrospective* annotations, **AXFi** recognizes a contrast between "intrinsic" and "descriptive" annotation. An example of an *intrinsic* annotation might be an image segment, where calculating the boundary of the segment is intrinsic to a specific image-processing objective, whereas an *descriptive* annotation might be an arrow *pointing toward* that segment. Here the descriptive annotation is introduced primarily for the benefit of human viewers.

The intrinsic/descriptive distinction is not always clear-cut. Consider the following two cases: in one scenario, an image-segmentation routine precisely delineates a region of interest (e.g., an outline of a red bird against blue sky), notating the segmentation result via a two-color-depth transform of the original image. Image-viewing software then shows the segment indirectly by encircling it, producing, in effect, a secondary annotation intended to call attention to the primary (segment) annotation. Here, clearly, the segment itself (encoding by a separate two-toned image) is intrinsic to the original analysis, whereas the secondary annotation has a purely descriptive purpose.

However, consider an alternate scenario where the original segmentation is done with less precision (this may be the case where there is a more muted color differential between foreground and background). Imperfect segmentation may still be adequate for some semantic task (e.g., identifying a bird's species). An analysis might obtain a rough segmentation by marking certain points highlighted by an edge-detector, then protruding the convex hull of the region outward so as to be sure of encompassing the whole bird-segment within a polygon, albeit allowing some background pixels into the polygon as well. This approximate segment is only an indirect representation of the region of interest (which would be the avian sub-image clearly outlined with no background included), but for analytic purposes the rough polygon might be a reasonable substitute for the finer-grained segment, analogous to how a cubic or quartic polynomial may be an adequate approximation to a more complex curve. Depending on how it is used, the approximate segment may therefore be considered merely a visual cue connoting the region of interest, or a significant region in its own right. This contrast would, most likely, depend on whether the approximation is used itself as a basis for further analysis, or instead is mostly a presentation device. This usage-context would therefore indicate whether an "imprecise" annotation (**AXFi** formally uses the term *approximative*) should be classified as *intrinsic* or *descriptive*.

All told then, annotations may be *internal* or *retrospective*, and *intrinsic* or *descriptive*. **AXFi** also

contrast *computer-generated* from *human-generated* annotations (which are related to but not the same terms as *manual* and *automated*; see below, page 11). These distinctions are indepenedent of one another; retrospective annotations for instance could be either intrinsic or descriptive, and either computer-generated or human-generated.

With these preliminaries concerning the scope of **AXFi** clarified, the following sections will (1) outline the kind of data encoded via **AXFi** and (2) the relation between **AXFi**, data types, and applications/libraries that may use **AXFi**. For purposes of exposition, the following specifications will discuss images mostly in two (spatial) dimensions; however, **AXFi** does not preclude annotations applied to image-sources with other dimensional profiles, such as **3D** graphics, or movies (**2D** plus time), or even audio files (consistent with **DICOM-SR**, which supports audio annotations because **DICOM** is sometimes used to share biomedical acoustical data, such as ultrasound).

Part I: Outline of the AXFI Data Model

Types of Image Annotations

The most fundamental kinds of image annotations are geometric primitives such as points, lines, and polygons. Many other forms of annotations are possible, however, mostly supplemental data which is associated with these primitives or, in some cases, with the image as a whole. In general, **AXFi** classifies annotations into the following groups:

Geometric Primitives These annotations delineate spatial/geometric regions in zero, one, or two dimensions (or potentially higher dimensions when working in non-**2D** contexts). At a minimum, **AXFi** data should support points, lines, polygons, polylines (considered a superkind of polygons where a polygon is a closed polyline), circles, and ellipses. Additionally, **AXFi** recognizes generic "closed" and "open" *curves*, which are nonlinear one-dimensional regions (or boundaries of two-dimensional regions) that are neither elliptical or circular arcs. Depending on context, **AXFi** can be extended to provide more detailed subkinds of curves generated by different sorts of mathematical equations, along with notations for the formulae (e.g., b-splines) that generate a particular curve.[4] More information about encoding ellipse data as well as other sorts of curves is outlined below (page 9).

A variation on the polygon/polygon-line alternative are polygon-lines demarcating spatial regions whose boundaries extend to one or more sides/corners of the image (e.g., a "quadrant" is defined via one central point with line segments parallel to and implicitly extended to the edges). These poly-lines may not explicitly represent the overall image boundary as part of their own boundary. Conceptually, treating the image-border itself as a different *sort* of boundary than those explicitly notated may be more accurate than eliding these distinctions. This applies also to segment boundaries. For instance, the sand/ocean border in a beach scene represents a physical discontinuity between two different material substances, and so it records an observable detail of the depicted scene. On the other hand, the edge of the sand-region at the bottom of the image is not a physical boundary, but an artifact of the camera position; we assume the beach itself extends further than what the camera captures. All told, then, in addition to polygons (or curves or segment-boundaries) being *open* or *closed*, **AXFi** recognizes a third form of closure dubbed "incomplete," meaning that a spatial region is geometrically closed by the edge of the image but that this closure has no observational or semantic significance. **AXFi** allows a notation that a spatial region is "closed by fiat" when the closure results from the intrusion of the global image boundary. A polygonal line may be closed by fiat when it does not explicitly include lines along the global boundary, but forms a closed polygon when such lines are included in practice; the result is called a *fiat polygon* (similarly an annotation can be classified as a *fiat curve* or *fiat segment*).

[4] In formal statements, this and other **AXFi** documentation will use the term "kind," as well as "superkind" and "subkind," to indicate groups of values/entities identified via a classification. Vocabulary based on "kind" is preferred to "type" or "class" because of the distinct meanings these latter terms have in computational contexts which are also discussed in reference to **AXFi**.

Segments and Regions A *segment* is considered to be, canonically, an integral subimage with a semantically precise separation of "foreground" from "background." There may be vagueness or approximation in how the segment is precisely individuated from its surroundings, but these ambiguities are considered to be practical limitations due to limited computer power, limiting pixel resolution, and so forth. A *region* or *region of interest* is similar to a segment, but defined more loosely; regions can have vague descriptors, and spatially disconnected parts of an image could be treated as parts of one same region. In a photograph of a flock of birds, for example, there may be multiple segments, each outlining one single bird. However, the flock as a whole may be outlined by one *region*, which could (without being deemed approximative) include some of the background sky. A *segment* can be seen as subkind of *region* with stricter granular and topological requirements.

A *partition* of an image is a segmentation which exhaustively classifies every point into one or another segment. A *selective* segmentation, unlike a complete partition, only isolates certain segments or regions of interests. AXFi adopts these terms as parameters that can characterize segmentation processes, and by extension the resulting segments/regions.

Locations In AXFi, *locations* are considered to be designations of areas within an image (of varying dimensions) which are significant by virtue of their directional, morphological, or topological relations to the rest of the image, rather than by virtue of their intrinsic shape. Conceptually, a *location* is in many cases similar to a *point*, but AXFi does not require locations to be zero-dimensional. A location may be designated by a small disk, or even a region/segment. The distinguishing feature of locations is that their spatial shape or extent are not semantically significant; instead, what is important about locations is their position in the image and how this position relates to surrounding image content. For instance, a location might be the point/position where two roads intersect, or it could be the leftmost point on the segment-boundary of a car's fender or a bird's wings, or the geometric center of a car's body or a bird's torso.

AXFi distinguishes location-annotations from secondary annotations used to identify locations (insofar as these may be visually distinct). For instance, a position may be modeled as a single point, but visually conveyed by an arrow, or by a circle hovering above the relevant point.

Focal Points The concept of *focal points* integrates locations and geometric primitives for certain analytic tasks. In general, focal points are important for positional rather than morphological regions, similar to locations. However, focal points may function more like segments or geometric objects when used as part of an analytic objective. For instance, points embodying the center of a car's body or a bird's torso could be used to count the number of birds or cars appearing in a photograph. In this case the concept of "geometric center" has no meaningful morphological properties, so it is analogous to a location. On the other hand, the center may be treated as a proxy for, or a most significant component of, a segment or region; in this sense the point is *part of* a segment. This mereological aspect makes focal points act conceptually more like geometric primitives than like locations. In short, the classification *focal point* is available for spatial objects which behave somewhere between locations and regions/points, and particularly when they are used in some proxying or indicative relation to other regions (e.g. for counting).

Secondary Images In some contexts, such as segmentation, image-transforms are used to convey image-processing operations, in contrast to geometric-style annotations that can be defined via vertex coordinates. A crisp segment can be defined via a two-toned image with the same dimensions as the annotated image, but with only two logical colors (colors are called "logical" insofar as the relevant detail is how the colors compare to one another, irregardless of the optical colors used to render them[5]). A "fuzzy" segment can likewise be defined via a greyscale image (anything which is pure-background becoming either pure white, or pure transparent, depending on context).

[5] In Qt, for instance, the **QColorConstants::Color0** and **QColorConstants::Color1** color values are considered to be special "colors" (i.e., **QColor** instances) which define the foreground and background of a two-toned image; they are not formally assigned to a visual color, like black or white.

Secondary images can be used to denote annotations which are too granular to be summarized by any mathematical expression. In these cases, the actual annotation data should identify the secondary image (e.g., via a file path) and explain how it relates to the primary (or "ground") image, whereas the secondary file itself fills in the annotation details.

Secondary images may be derived from ground images merely to present annotations, but they may also be intermediate analysands which are themselves annotated. In the latter case, annotations on secondary images are usually meaningful also as annotations on ground images, so the interrelations between both images should be notated in the annotation data.

Proscriptive Annotations AXFi allows for the designation of certain annotations as "proscriptive" when they do not formally delineate a geometric object, but indirectly express such an object's shape or how it may be derived. A canonical example is the use of color — perhaps color-enhanced secondary images — to designate segments or regions of interest. For instance, one common segmentation technique uses color simplification; smoothing out color patches can facilitate image partitioning by reinforcing the boundaries between different regions. With sufficient color manipulation, image-segments can potentially be described chromatically: a region may for instance be identified as "the area all of whose pixels display a red channel above" some threshold. AXFi data should therefore allow such indirect designations of segments (and spatial/geometric regions in general) to be encoded as annotations.

The concept of proscriptive annotations is not only chromatic — one can imagine other scenarios where morphological features, such as symmetries, may be employed to similar effect. In describing a chess board, for instance, a set of image-regions (each square on the board) can be derived via translational transforms of an initial two-segment kernel (one black square and one white). This representation is possible because of translational symmetries in the underlying image. Such geometric transforms and symmetries can sometimes be used to "generate" meaningful image segments, so they should be notated in AXFi data when appropriate.

Semantic Labels Some annotations supply data about other annotations (what AIM calls "Annotation on Annotation" as opposed to "Annotation on Image"). In general, we can supply a label, description, or semantic classification indicating what an image segment or region is "about", i.e., what it "depicts" (a bird, a flock of birds, a car, a traffic jam, and so forth). Such meta-annotation may be seen as a semantic postlude to an imaging process, but it may also be seen as providing further detail about the process itself. For instance, suppose segmentation isolates a bird from the sky: the epilogue to this operation is an ability (for a human user or a computer algorithm) to classify the segment as "bird." However, we can also say that the given fact of the segment capturing the optics of a bird (with its apparent anatomical outline and coloration) is what allowed the segmentation to be possible in the first place. Therefore, the label "bird" serves both to utilize the segmentation for further analytic/classificatory purposes and also, potentially, to characterize the inner workings or effectiveness of achieving the desired processing objective. It should be kept in mind, in short, that label annotations are not exclusively intended to be used for practical labeling in the contexts of tasks such as subject-marking images in corpora; labeling could also be used to notate how semantic properties of the image make segmentation (and other processing objectives) more or less feasible, or computationally intensive/error-prone.

The semantics of labels themselves is outside the scope of AXFi. AXFi makes no effort to proscribe what sorts of terms are useful as labels ("bird," "car," "ocean," "tumor," etc.), or whether labels are simple strings/words or have internal structure of their own. AXFi does, however, make the following minimal stipulations about labels. First, AXFi distinguishes "mass," "count," and "plural" label targets — e.g., *ocean*, *one bird*, *two birds*. These distinctions are directly relevant to how segments are bounded and counted; for instance, labeling a body of water as *a lake* (e.g. on a satellite image where the lake's full shore is visible) versus *ocean* (continuing to the horizon) implies different desiderata for how the labeled region spatially extends in relation to the surrounding image. Secondly, with respect to semantic labels, AXFi recommends that applications and libraries enable labels to be typed values — instances of application-specific data types — as

well as simple character strings (like "bird.") This is consistent with AXFi's general approach to application integration and type systems, to be discussed further in Part II.

In addition to labels being second-order annotations — assertions attached to an image segment, for example, or any other image feature deemed semantically relevant in some context — labels may be applied to the image as a whole. In this context labels would be an example of *metadata* annotations (discussed next), which apply to the image itself rather than any proper part.

Image Metadata This category of annotation concerns any information "about" an image which is apart from the specific image content. Such details could encompass computational/encoding details such as color depth, pixel dimensions, and image file format; or acquisition details such as the make and settings of the camera whose photograph yields the current image. In theory, any information prerequisite to displaying an image may potentially be relevant to how annotations are used and interpreted. Therefore, AXFi should support representations of image metadata where such information is needed to fully specify annotation data; the metadata would then, typically, be considered an annotation on the entire image. It is outside the scope of AXFi to present image-related data which is applicable to the sharing or rendering of images but not, in a particular context, important to annotations themselves — unlike DICOM, for instance, which is designed as a standard for ensuring that images are correctly rendered and therefore includes detailed specifications of image formats and prerequisites, AXFi introduces image metadata only where such information can be considered intrinsic to the specific objectives and operations of image annotation.

Acquisition Context Among the class of annotations which apply to the image as a whole, AXFi recognizes a distinct kind of annotation covering what AIM calls the "context," or the purpose and concrete goal which compels users to initiate image-processing operations. In general, some of this contextual data should be preserved as a supplement to the overall information generated by image-processing operations. Often image processing itself is only one stage in a multi-faceted scientific workflow, and contextual information which exists prior to the image-annotation step may still be important for subsequent steps. For example, clinical data which motivates a radiographic study should remain linked to images and annotations which result from that study; this is why DICOM-SR includes clinical as well as imaging data.

This outline covers the different kinds of annotation data recognized by AXFi. The following section will fill in some missing details by addressing other sorts of information, apart from annotations themselves, which might be included in an AXFi data package.

Non-Annotation Data in AXFi

This section covers background information which may be addressed by AXFi-modeled objects to ensure that annotations are properly understood. The discussion will consider some details of non-annotation data and also clarify some details in how annotations are constructed.

Dimensions and Coordinate Systems Many different coordinate systems may come into play during an overall image-processing task. Even if attention is restricted only to 2D graphics — where each pixel is mathematically locatable with just two numbers — different algorithms or code libraries define 2D coordinates in different ways, so that representations of individual points need to be mapped across several different transformations. Coordinate systems differ in terms of whether locations are specified with integer or real-valued (via binary approximation) magnitudes, and whether designations of shapes and geometric structures are oriented to bounding-box points or centers; whether magnitudes increase left-to-right/top-to-bottom or vice-versa; and so forth. The application library with which an image is rendered may have a different system then the analytic library with which it is processed. These differences become significant insofar as, for example, annotation data should be modeled so that applications can display visualizations of annotations and respond to GUI events so that users can examine or manipulate them.

During the course of image processing, certain operations moreover generate new sorts of coordinates; for example, segmentation may result in one part of an image being isolated in such a way that a separate coordinate system can be used for the interior of that region. The "Insight Toolkit" (ITK), for example, distinguishes "object space" (internal to individual geometric objects) from "world space."

Furthermore, additional coordinate systems may be relevant to the context or empirical situation wherein an image is acquired. Geospatial images, for example, can associate image-space locations with geographical coordinates. In some contexts, the coordinate frame applicable to some image can be contextualized with reference to parameters of the acquisition equipment; consider microscope configuration. The ground image may also be the result of intermediate processing subsequent to data acquisition; consider Flow Cytometry transformations. In these cases, the coordinates and dimensions considered "internal" to the image can be associated with other dimensions pertaining to the real-world context and/or image-acquisition protocol.

To make these connections explicit, AXFi employs a distinct category of *dimension* objects, which can ground multiple parameters related to dimensions/axes, their scales, units of measurement, kinds of magnitude, transformations, and so forth. One or more dimension objects can then be merged into an object representing a specific *coordinate system*.

Of course, an image's coordinate system is in part a product of the equipment used to acquire the image. Therefore, metadata about coordinates may include descriptions of the acquisition process and the instruments/settings used. In general, the domain of imaging overall can be characterized by stipulating that geometric meaning in the sense of spatial orientation/topological relations (left, right, above, intersecting, around, inside) and/or chromatic value (color as an indicator of light hue/intensity) are semantically and observationally meaningful in the image data — they reflect the process whereby empirical data becomes digitally manifest in the image. The actual image data may be a computational transformation of the empirical details (as in the wavelength alterations involved in visualizing infrared camera pictures).

Vertices A vertex is an image-position which can be expressed in one or more coordinate systems. Because image-processing environments typically utilize multiple coordinate systems, an intrinsic facet of vertex data is how a single position will map between different coordinate systems. For this reason, AXFi utilize a notion of vertex objects which are distinct from any specific coordinates; a concrete representation of one position within an image then derives from pairing a vertex object with a coordinate-system object.

When integrating with applications and/or code libraries, AXFi vertex objects will typically be connected to context-specific data types, and translations between coordinate systems may be provided via object methods of these context-specific types. Examples in this case would be **mapFcsToScreen** in FACSanadu (a Flow Cytometry application); **mapToGlobal/mapFromGlobal** in Qt (where "global" in this context refers to screen coordinates); and **convertWindowToPDFCoords** in XPDF (recall that AXFi considers PDF page views to be annotatable images).

Geometric or spatial primitives such as points, lines, poly-lines/polygons, and image-locations are defined in terms of one or more vertices. This means that the objects representing such primitives are built on top of vertex-objects, which in turn are built on top of dimension and coordinate-system objects. Any geometric object in AXFi should therefore be understood to have a coordinate-system context which specifies how the numeric values defining the primitive are to be interpreted, even if the surface-level representation of the object does not explicitly designate this context.

Circles, Ellipses, and Curves AXFi does not preclude representing linear or curved geometric shapes in different ways. AXFi *recommends*, however, that representations build off of the dimension/coordinate/vertex stack as organically as possible. This has several consequences for representing curves of different varieties.

In the case of ellipses (with circles as a special case) a vertex-centric representation is based on elliptical foci (which reduce to circle-centers in the circle context; i.e., a circle is an ellipse whose foci

occupy the same position). The full characterization of the curve requires one additional scaling parameter, such as minor-axis length (for ellipses, collapsing to diameter for a circle). AXFi object libraries should convert ellipse-representations from this foci-oriented approach to others which are common in scientific computing, such as covariance matrices and bounding rectangles.

More general curves can sometimes be constructed out of poly-lines, as is the case with B-SPLINEs. Although B-SPLINE "control points" are not part of the curve, they are geometric locations which determine the shape of the curve, so that representing B-SPLINEs does not require any descriptive capacity apart from that of poly-lines. Other kinds of curves, however, may call for more elaborate mathematical descriptions. Therefore, AXFi does not restrict the sorts of formulae which may be encoded in annotation descriptions.

This does not mean, however, that any and every mathematical expression should be representable via AXFi data specifically. For an analogous scenario, GATING-ML (which is used to demarcate gates or, in effect, "segments" within Flow Cytometry data/data-plots) "Originally [reused] MathML to universally describe any kind of mathematical transformation. While this would be a very flexible solution, it is generally impossible to evaluate expressions describable by MathML ... Transform[ing] gate vertices into the data space and describ[ing] (using mathematical formulas) how edges 'have been curved'" was avoided because the "description (formulas) would be significantly difficult to create."[6] That is, translating arbitrary formulae that may be concretized by application-level code into a serialization/markup format can be very difficult. Essentially the same problem will arise with any attempt to define arbitrary geometric overlays on a two-dimensional image-like space, as required for AXFi annotations.

The solution favored by AXFi is simply to allow procedural code to serve as a proxy for describing the relevant curve, coordinate transformation, or geometric object. An AXFi annotation context can provide a unique identifier for a procedure which evaluates any given mathematical equation, and then use this procedural reference as an indirect way of denoting the parameters of an annotation. This makes the proper use of the annotation dependent on users having access to the code in question, and integrating that code (or functionally equivalent code) into the libraries which process AXFi objects. However, as discussed below, application-level integration is considered an intrinsic part of deploying AXFi, so the reliance on application context to fully disambiguate AXFi semantics is not regarded as a limitation.

Application-Level Algorithms, Procedures, and Data Types AXFi explicitly assumes that any annotation data is created and consumed by applications which display and/or analyze images. In this sense there is no conclusive reason to exclude application-specific data from a collection of AXFi objects. Any application-level procedure or data structure could potentially be used to define annotation contents and/or targets, or to supply contextual information about annotations and image-processing objectives.

Of course, the usefulness of AXFi data will be limited if applications introduce idiosyncratic information which cannot be understood in other AXFi environments. For this reason, it is recommended that applications and code libraries which use AXFi construct mappings between AXFi data and application/library-specific types and procedures in a manner which prioritizes transparency and reusability. This goal shapes the architecture and terminology used by AXFi, as will be reviewed in Part II. In general, an application/library utilizing AXFi should provide an "AXFi Object Library", i.e., a collection of data types whose instances (objects) are serialized and deserialized via AXFi. As much as possible, this Object Library should be autonomous from the application/library in whose context it is developed.

Notwithstanding this criterion of autonomy, AXFi Object Libraries can include designations of application/library-specific data types and procedures. As much as possible, these computational artifacts should also be autonomous from the overall application context. It is recommended, that is, that types and procedures directly referenced by AXFi Object Libraries be factored into a separate

[6] See http://flowcyt.sourceforge.net/gating/latest.pdf, pp. 12 and 15-16.

"AXFi Host Library" which wraps the Object Library and bridges it with the main application. The Host Library should then be annotated rigorously enough that code-annotations in the Host Library provide as basis for image-annotations in AXFi data, insofar as such annotations rely on application-specific objects or procedures.

Annotation Body and Targets Although AXFi is specifically concerned with image annotation, it is desirable to model the image-annotation process as much as possible within the more generic environment of resource-annotation as a whole, such as the "OpenAnnotation" model and the Linguistic Annotation Framework (LAF).[7] To conform with these precedents, AXFi vertices — as well as additional measures which delineate geometric shapes together with vertices, such as circle diameters — are called "anchors." A geometrically-described subimage or geometric primitive (or also a secondary image used to demarcate primary-image regions, as when a two-toned bitmap is used as a pixel bitmask) is called the "target" of an annotation. The *body* of an annotation is designated via image-processing objectives terminology to clarify the analytic significance of a delineated region/primitive.

As a concrete example, suppose an annotation environment uses disks to notate image locations. A location-annotaton would then have, as *target*, the description of the desired disk as a geometric object (i.e., via a circle with center-position and radius). The *body* of the annotation would then be a label or descriptor indicating that the geometric object (the circle) is demarcated so as to define a location annotation (and not, say, an image-segment).

Note that this notion of annotation body only covers semantics intrinsic to the base-level annotation process. In AXFi, the "body" of an annotation applied directly to an image would not include higher-level semantic or contextual information; instead, those assertions are provided by further annotations attached to an image-specific annotation. For example, suppose a location-annotation is constructed so as to assert that a particular image-location represents the main entrance to a photographed building. The architectural concept "main entrance" (or "door," etc.) would be asserted via a semantic annotation whose *target* is the preliminary (purely "geometric") annotation which fixes the location. These paradigms imply that certain descriptions which in other contexts (such as linguistics) would be represented via a single annotation are instead, in AXFi, constructed via a network of two or more distinct annotations.

The body of an annotation may also contain metadata about the annotation, such as a clarification of whether it was generated by a human user or an algorithm (or some combination). A "manual" annotation is produced solely by a human user, whereas an annotation is called "human-generated" if it is defined by a user non-programmatically (i.e., through a GUI), but with the aid of some computational/algorithmic process.

Annotation Graphs In AXFi, as just alluded to, complete information about a specific annotation often spreads across multiple annotations. Associated with each annotation is an annotation *object*, i.e., a computational artifact conveying information and parameters specifying the annotation. Each annotation object is moreover a *node* within an overall annotation graph. Between any two annotation objects, one can via AXFi assert a relation or connection, establishing a graph-edge between the nodes. Moreover, AXFi uses a "hypergraph" data-modeling framework wherein nodes can be grouped together into different sorts of aggregates. Specifically, AXFi serializations can use the "Hypergraph Exchange Format" (HGXF) to be developed by Linguistic Technology Systems, or else some functionally similar hypergraph modeling paradigm. HGXF is outside the scope of this outline, but hypergraphs models are discussed further in the context of graph-serialization in Part II.

Similar in spirit to DICOM-SR, using a node-based and graph-oriented data model allows AXFi to flexibly connect image-annotations to whatever contextual, semantic, and "real-world" data is

[7] See https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4485195/ on the connection between bioimaging annotation and the OpenAnnotation framework; and https://dash.harvard.edu/bitstream/handle/1/11879601/3889917.pdf?sequence=1, https://www.cs.vassar.edu/~ide/papers/ide-suderman-LRE-LAF.pdf, or https://www.cs.vassar.edu/~ide/papers/ide-romary-clergerie.pdf on LAF.

important for the larger image-processing objectives. For instance, if the annotation upon a CT scan circles or outlines (what is diagnosed to be) a tumor — visually calling attention to the malignancy and perhaps providing a measurement — the raw annotation has obvious conceptual/empirical links with a large body of contextual and follow-up information, such as the patient's clinical data, the prescribed Radiation Treatment Plan, and other kinds of data about the cancer (e.g., genomic profiles of the cancer cells) obtained by other sorts of biomedical investigations. It would be impossible to formalize all such representations of contextual data for all image-annotation environments: the kinds of supplemental data relevant in one biomedical context (such as oncology) will be different in other contexts (such as immunology or osteoporosis) — let alone imaging contexts beyond bioimaging.

These considerations suggest that AXFi should provides objects as affixation points for open-ended contextual data, where the inner structure of such supplemental data is outside AXFi's scope. In the above oncology case, for instance, the tumor-annotation would be annotated with a labeling to the effect that the image Region of Interest evinces a diagnostically significant finding. That is, the annotation is situated within the processing objectives that influence the overall imaging process, namely, in this case, to diagnose a pathology or malady of some kind. The tumor-image — i.e. an image segment identified by a person or a computer as suggesting the presence of that tumor — represents the culmination of this image-processing activity, insofar as that specific segment (having been outlined or otherwise pointed to, e.g. via a circle or arrow) summarizes the entirety of the image-processing in relation to its objectives. An annotation isolating that segment, and then labeling it as diagnostically meaningful, then serves as a "pivot" connecting the image-processing activity to any other clinical or course-of-treatment information spaces. In short, the secondary annotation object (the node labeling the annotation itself as a diagnostic finding) can be connected to any other object, whose type and semantics is at the discretion of the relevant AXFi object library.

While allowing for these open-ended interrelationships between image-annotations and other data models, however, AXFi would be most useful in contexts where non-imaging data is managed according to paradigms consistent with AXFi itself. That is, the overall architecture of AXFi systems — in terms of AXFi Object Libraries, Reference Implementations, and "grounded" serialization, all to be discussed in Part II — can be applied to other contexts outside of imaging proper. Code libraries which employ these paradigms will be able to integrate with AXFi more easily.

Part II: AXFI Library Architecture

AXFI Object Libraries and Serialization

AXFi is designed around the premise that the primary goal of any data modeling system (and, as a surface-level encoding of data to be shared, any markup language) is to encapsulate data which is used by and present in one application, to be subsequently re-used by a different application — or the same application at a future time. There is an application which *creates* an annotation, and a (not necessarily different) application which *consumes* an annotation. "Consuming" in this context means rendering the annotation so it may be viewed by a human user and/or reading annotation data so that such data may be incorporated into some automated computational process. Of course, the consumer-application may also be a creator, adding new annotations or revising the ones it has consumed.

The simplest use-case is where the same application both *creates* and *consumes* each annotation. However, there would be less need for standard models if every application only shared data with itself. We can assume, then, that any standard such as AXFi is primarily intended to ensure that two *different* applications correctly share data. Data sharing is "correct" insofar as the consumer-application reads, understands, displays, and allows human users (as well as, potentially, computer algorithms) to manipulate the annotation in ways which are functionally equivalent to the creator-application. When this functional equivalence is achieved, the two applications can be said to be *aligned*.

Creator-to-consumer alignment can happen in one of two ways. One option is that the data model shared between the two applications is formalized so that different applications are naturally interoperable because they are implemented according to similar guidelines. This may be called "coincidental alignment." There are, in turn, two pathways toward coincidental alignment. On the one hand, data-model formalization may stipulate criteria on creator and consumer applications to ensure that any creator is aligned with any consumer. Such a data model would essentially provide a "checklist" of requirements to be satisfied, procedures to be implemented, and so forth, so that applications are only considered to be creators and/or consumers by fulfilling the checklist contract.

On the other hand, the formalization may focus on the medium through which data shared between creator and consumer is structured and encoded. This second mode of standardization — coincidental alignment based on data-serialization formats — is arguably the most common strategy for enforcing alignment (even if it is not identified in these terms). For example, **XML** schemata represent constraints on any creator of information encoded by conformant **XML** documents; given these constraints, the **XML** *consumers* can guarantee alignment with creators by incorporating the schema as a contract the creator obeys. The consumers need not have any knowledge of the creators "apart from" the fact the **XML** they create conforms to the schema.

Whether by explicit application-level requirements or via contracts on the medium by which shared data is encoded (e.g., **XML** schemata), coincidental alignment can occur without any explicit coordination between the code used by creator and consumer applications, respectively. As an alternative model, however, we can identify *deliberate* alignment, wherein the consumer aligns with a creator by explicitly referring to and mimicking the creator's data models and procedures. To facilitate deliberate alignment in this sense, creator applications can architect the relevant code to prioritize transparency and reuse — e.g., factoring this code into a semi-autonomous library which the consumer application can use directly, or use as the basis for its own library.

In contrast to many data models, **AXFi** openly embraces deliberate rather than coincidental alignment in this sense. This is expressed through the idea of **AXFi** Object Libraries: a creator of **AXFi** is encouraged to provide (or reuse) a code library which implements logic for constructing, serializing, and deserializing **AXFi** objects. Each **AXFi** Object Library may add its own data profiles and capabilities to the underlying **AXFi** system. Linguistic Technology Systems provides a default **AXFi** library, called "**D-SPIN**" (Data Structuring Protocol for Image-Analysis Networking). This **D-SPIN** library serves as a *reference implementation* for a generic or canonical **AXFi** format. Other **AXFi** Object Libraries can provide reference implementations for alternative/extended versions of **AXFi** by adding their own features which are not present in **D-SPIN** itself.

Unlike other frameworks, then, **AXFi** does not define a standard data structure or serialization format except insofar as each reference implementation, exposing a given **AXFi** Object Library for cross-application reuse, yields an archetype that can be copied or modified. However, any data shared between heterogeneous endpoints needs to be encoded somehow, and **AXFi** does model certain patterns in how **AXFi** data should be represented in textual or binary files/streams.

All **AXFi** data exists in the context of some **AXFi** Object Library. When networking between separate applications, **AXFi** assumes that the creator and consumer either uses the same Object Library or two libraries which are functionally equivalent, at least in all ways affecting the shared data. So the role of an **AXFi** *package* — that is, a file, or some similar text or binary stream carrying **AXFi** data — is to allow objects created by an Object Library to be serialized/encoded and then later deserialized by the same (or an equivalent) Object Library. Any binary or textual format which permits this "co-serialization" — i.e., proper serialization and deserialization between aligned libraries — is a reasonable surface-level encoding of **AXFi** data.

In short, for each **AXFi** data package there is a collection of **AXFi** objects being serialized and (later) deserialized, and the package has a *surface representation* whose primary importance is to be a convenient vehicle for the relevant co-serialization. Serialization formats can be chosen to minimize the amount of biolerplate code needed for the Object Library to marshal **AXFi** data between runtime

and serial encodings.

While any markup language, or other encoding mechanism, could potentially be used for co-serialization, AXFi also recommends that Object Libraries employ serialization formats which document how the data-sharing logic is implemented. In other words, the data format should make the process of marshaling information between runtime and co-serialization environments transparent and reusable. This is different than (say) an XML schema, where the serialization format is the *basis* of multi-application alignment. In AXFi, the Object Libraries, not any markup standardization, are responsible for application alignment. Nevertheless, while serialization format has no constraints to enforce alignment on a technical level, AXFi Object Libraries should be considered easier to re-use if they employ a conceptually nuanced and self-documenting format for expressing/sharing data.

To this end, Linguistic Technology Systems provides a new markup format, in conjunction with the D-SPIN reference implementation for serializing AXFi data, called GTagML ("Grounded" TAGML), where TAGML is an existing XML-like language (but with a non-XML syntax and a richer semantics, e.g., concurrent markup). GTagML is "grounded" in the sense that each GTagML file presents the co-serialization of certain instances of certain data types, where the relation of the explicit contents of the GTagML code and the data types which are present as the code is generated — and will again be present (or have aligned, corresponding types which are instantiated) when the code is de-serialized — is explicitly modeled as part of the GTagML expression. That is, GTagML data packages are "grounded" in explicitly-referenced data types, co-serialization logic, and code libraries using GTagML for inter-application communications.

A thorough discussion of GTagML is outside the scope of AXFi, but a brief summary of how GTagML's notion of grounding applies to AXFi can be sketched out. Any AXFi data package represents the serialized encoding of one or more objects defined/constructed by an AXFi Object Library. The GTagML code which serializes these AXFi objects is therefore a tool to allow the objects to be lifted from their runtime context and re-created at some other time and/or place. But the GTagML code also bears a formal relationship to the data type instances which it serializes: a given set of GTagML nodes, for instances, encodes some particular object (e.g., some annotation or some geometric primitive), an object which is an instance of a given data type, provided by a given code library, linked in to a given creator application. The serialization occurs at a given moment in time, and in a given application context (the specific session, user, or provenance data which might be relevant for the overall co-serialization context). The idea behind GTagML is that such semantic relations between GTagML markup and the data type/application-level context where this markup is generated is formally modeled within the markup itself. When serialized via GTagML, AXFi data includes a structured description of how the AXFi objects relate to the GTagML code which encapsulates them.

The relation between AXFi and GTagML can be further illustrated by considering domain-specific extensions or versions of AXFi, which are discussed in the following section.

Extending AXFI

In some contexts, AXFi can be used as an alternative format for serializing data conformant to other standards related to image annotation. When these use-cases call for refining the AXFi specifications, the architecture of AXFi Object Libraries and GTagML can be employed to make these extensions compatible with the core AXFi model, and to facilitate application integration. This section will consider several extensions along these lines allowing for AXFi being adopted as an alternative to GATING-ML (in the Flow Cytometry context), to ARCGIS (for geospatial data), and to DICOM-SR or AIM (in the biomedical imaging context).

Given any AXFi extension which sustains the design principles of AXFi itself, the extension's central technology should be an AXFi Object Library which includes object classes modeling the relevant extended semantics as well as the primary, geometric-based AXFi data. It is recommended that the Object Library be implemented in a programming language and environment which minimizes external dependencies. In particular, C or C++ are preferred to scripting languages unless the library

can be developed in "pure" scriptin code (without importing packages or modules which are mostly composed in C/C++). For C++ libraries, it is recommended that most dependencies be limited to a single overarching framework, like **boost** or Qt, without *mixing* dependencies. In Qt, for instance, the IO, Regular Expression, and XML support (and so on) is sufficiently mature and full-featured that there should be no need for dependencies on **boost::filesystem**, **Boost.Regex**, or etc.

For sake of discussion — and to remain consistent with the default D-SPIN implementation — assume therefore that prototypical AXFi Object Libraries are implemented in Qt-based C++ code. The core of any new (potentially extended) library, then, is a *co-serialization interface* which includes procedures for serializing and deserializing data structures (including C++ class instances assuming certain template procedures are instantiated for those classes), as well as "querying" encoded data as part of the deserialization process. For D-SPIN, the co-serialization logic is implemented using LTS's ConceptsDB, a hypergraph database engine. An overview of this technology is outside the scope of AXFi, but the key detail is that ConceptsDB allows data structures to be modeled and then encoded via certain hypergraph constructions, and that the resulting hypergraph models are manipulated via an interface which incorporates Functional Programming techniques (more so than conventional Object-Oriented styles). An AXFi Object Library mimicking D-SPIN, accordingly, would implement logic to serialize and deserialize AXFi objects according to this hypergraph-based interface.

Such an Object Library would then have, at its core, a suite of annotation objects (plus objects expressing an extended semantics) and would have procedures for serializing and deserializing these objects as hypergraph data structures. This co-serialization interface would then be augmented to support GTagML by re-implementing the (de)serialization procedures to generate/parse GTagML instead of the ConceptsDB hypergraph format. The GTagML interface should be tested to confirm that it functions equivalently to that of ConceptsDB. Once that is verified, the generated GTagML code can be provisioned with "grounding" metadata, that is, with parameters indicating the relation of the serialization content to the data types and procedures of the Object Library. In the simplest case, a GTagML node is "grounded" by being marked as encoding one instance of an object class.

An important aspect of an AXFi Object Library, then, is functional equivalence between the ConceptsDB serialization and the GTagML serialization. This notion of functional equivalence may then be extended when the purpose of an Object Library is to incorporate features from other data standards, or to allow AXFi to replace other data formats. For example, Object Libraries designed to adopt AXFi in lieu of GATING-ML, AIM, or DICOM-SR, respectively, would need to demonstrate that they serialize and deserialize structures in a manner functionally equivalent to the formats being emulated.

In these scenarios, Object Libraries need to orient themselves not only to standards for existing markup formats, but also to existing libraries where data in such formats is generated and consumed. For example, the de facto reference implementation for reading GATING-ML data appears to be the R/Bioconductor package **flowUtils**. Likewise, the AIM data format was initially engineered via an open-source C++ library (**AIMLib**), which is paired with **AIMConverter**, a utility to express AIM data via DICOM-SR (rather than AIM XML). For DICOM-SR, the principle C++ library (used e.g. by **AIMConverter**) is DCMTK, which is divided into multiple modules. The **dcmsr** module is the core DCMTK implementation of DICOM-SR, although **dcmpstate** (DICOM Presentation State) is also relevant for image annotation. Finally, in the GIS context, the ARCGIS Runtime SDK for Qt is a standard model for managing GIS annotations in a C++ context.

In short, these four libaries — **flowUtils**, ARCGIS, **AIMLib**, and **dcmcr** — supply the most canonical implementations that would be referenced by AXFi Object Libraries incorporating Flow Cytometry, GIS, AIM, and DICOM-SR data repectively. Recall that AXFi paradigmatically aims for alignment at the level of creator and consumer libraries, rather than at the level of markup format — the AXFi analogs for these libraries need not use the same serialization formats; indeed, if they use GTagML, they will not use an XML-based language at all. However, the respective AXFi Object Libraries can try to emulate the programming interface offered by the "target" libraries (**flowUtils**, ARCGIS, **AIMLib**, **dcmcr**). This entails duplicating procedure and type names as much as possible, and constructing

the interface such that the steps needed to serialize and deserialize data are largely congruent when a programmer is using the AXFi-compatible library to the emulated target library.

When emulating external components, AXFi Object Libraries should be attentive to the connections between emulation targets and their surrounding computational context. In the case of GATING-ML, for example, gating data is only meaningful in the presence of cytometry information (typically conveyed via FCS files) establishing the baseline data (analogous to a ground image) against which gates are defined. To preserve autonomy, an AXFi Object Library which supports FCM gating would not necessarily read or display FCS files itself; instead, such a library would be deployed as embedded within a larger FCM programming context, which would be responsible for managing FCS data and also any GUI operations (including responding to user events on gates).

The gating-specific Object Library therefore needs to define a protocol for sharing gating information with this host application. Such would be an example of an "AXFi Host Library" as described in Part I. The Host Library would serve as a bridge between the Object Library and the host application. When the Object Library acquires AXFi (including gating) data, it would construct gating objects and pass them to the host software, leveraging the fact that the Host Library provides a runtime overlap between the Object Library's and host application's data types and accessible computational resource (e.g., heap allocations). In the opposite direction, the host application could communicate gate modifications (due to user GUI actions, for instance, such as dragging gate handles) back to the AXFi Object Library if it is needed to update (re-serialize) the gating data.

Architecturally, then, there are three distinct components involved: an AXFi Object Library; a host application; and an AXFi Host Library which encircles a communication protocol and capability overlap between those two. The Object and Host libraries should be designed so that communication between the Object Library and a host application is rigorous and transparent, and so that the communication protocol can be reused in different contexts. Moreover, the Host Library should be designed so that implementational objectives can be clarified — criteria for the data management and HCI features expected of the host application. In the case of FCM gating, this would include formal models of how gates are visualized, what sorts of user events (e.g., mouse-events) are intrinsic to the semantics of the FCM system, and how user events translate into data modifications which lead to the gating data being updated.

Even though FCM gating is mathematically different than annotating photographic images, many of these architectural details are functionally parallel between the FCM and imaging (according to traditional image-acquisition processes) contexts. This is one reason why developing FCM gating annotations as a special case of AXFi is productive; it allows the overall application-integration between annotation, serialization, and HCI/GUI features to be systematically profiled, in parallel to analogous models applicable to other image-related annotation domains. Complementary comments apply to GIS, which, like FCM, evinces annotation requirements which are geometrically similar to systems such as AIM, but apply to an expanded notion of "imaging" (and image acquisition).

The value of treating these various imaging and annotation domains as alternative manifestation of one underlying computational space is that the requirements shared across these domains — particularly at the HCI and GUI levels — then become foregrounded, whereas they are largely left implicit or unstated in existing annotation frameworks. When formulating a multi-purpose image-annotation model, the relevant standardization applies not only to annotation *data*, but also to annotations as user-interaction artifacts: how they are displayed, the event model which systematizes how human users interact with annotations, and the logic for interconnecting event handlers with underlying annotation data. The AXFi architecture is organized such that this event model is grounded in AXFi Host Libraries, meaning that event-processing is guided by a canonical protocol. Such HCI focus is harder to achieve for formats such as GATING-ML or AIM whose standardization is driven by canonical serialization models rather than by canonical library models.

For more information please contact:
Amy Neustein, Ph.D., Founder and CEO
Linguistic Technology Systems
amy.neustein@verizon.net • (917) 817-2184