B978-0-32-385197-8.00015-5, 00008

CHAPTER

8

Image annotation as a multi-aspect case study

OUTLINE

8.1 Introduction

The previous chapter indicated many questions that should be answered by a comprehensive image-annotation framework. This does not mean that frameworks should enforce some choices at the expense of others, but they should at least identify details, where data models may diverge, and encourage specific models and implementations to document their individual policies. In other words, a framework does not need to be a definitive implementation; it could also be a protocol identifying details, which particular implementations should take into consideration.

8.1.1 Design questions for image-annotation modules

For the sake of discussion, we will reiterate in summarial fashion a number of the questions that were indicated in the last chapter.

LaTeX class: neustein, 2021/10/06 13:35:48 neustein driver • file: neustein08.tex (svetlana) • Prn: 2021/11/05;11:06:13 • p. 1
pro: NEUSTEIN • model: a10 • color: 4c • ref-style: Numbered (Style 1) • lang: en-US • docsubty: CHP •

1. What number systems should be recognized for defining magnitudes, so that basic quantitative data, such as distance between an image point and the image sides (allowing image-locations to be identified) can be notated? How many different number systems (integers, floating-point values with different degrees of precision, non-standard quasi-reals) should be recognized?
2. Should an annotation system allow multiple number systems to co-exist, e.g., an integer scale for image points, but a floating-point scale for ratios, angles, and other calculated values that lie outside an integer pixel grid?
3. What coordinate systems should be recognized for dimensions that have a spatial interpretation, such as pairs intended to represent horizontal and vertical position within an image? Certainly Cartesian coordinates are ubiquitous, but one can also consider homogeneous (projective) geometry, polar representation, orthonormal vectors not parallel to image sides, and so forth. Even with the most common orthogonal axes (parallel to the image sides) there are multiple options for the origin point, including all four image corners and the image-center (with direction of increase typically upward and to the right). Should arbitrary Cartesian origin-points be allowed inside (or possibly outside) the image interior, other than the center? Where should the center be located for an image with even pixel-height and/or width?
4. Should it be possible to designate points via tuples representing coordinates under numerical transforms, such as logarithms, scaling factors, hyperbolic arcsines, or biexponentials?
5. Should annotations possess a zoom dimension, which can vary independently of the ground-image zoom? That is, should annotations support a transformation, where they are rescaled, while the ground image remains the same (e.g., a circular or polygonal shape expand or contract relative to its center), or should any such change be executed simply by updating the point- and/or length-set? Likewise, should there be a mechanism for representing the annotation-scale (whether or not it mirrors the ground image) at different moments in time, e.g., when the annotation was first created—as it is currently being viewed—or are these presentation data that need not be recorded in the annotation itself?
6. What scale units should be recognized for coordinate positions and intervals, such as the nine possibilities standardized in SVG (centimeters, inches, and so forth)? Should annotation-sets potentially mix two different such scales (e.g., centimeters and percentages against image width/height)?
7. How should annotations' shape point-sets be ordered? Should the ordering be left to the discretion of any human user or software component, which defines an annotation? Should annotations that are identical modulo point-set permutations be deemed equivalent? How should point-sets with three or more colinear points be handled?
8. Should annotation shape be constructed solely via a point-set or should other magnitudes (what we last chapter called a "length-set") be allowed as a way of describing the geometrical qualities of the annotation shape?
9. How should special roles for particular points in a point-set or lengths in a length-set be described? Should roles refer to numeric positions in point- or length-sets modeled as ordered sequences, or should point/length sets be provided instead as key-value or key-multivalue associative arrays, or some combination?
10. Should annotations be able to refer to other annotations as a data point intrinsic to the annotation data, as would be the case with an arrow pointing to a separate annotation,

for example? If so, how should annotations be uniquely identified?

11. How should view-state data about annotations, such as zoom factors, colors, line widths, and cross-references between annotations, be registered as data structures complimenting annotations as opposed to intrinsic data *within* annotations? Which properties are in fact intrinsic and which are presentational?

12. How should we represent annotation-sets or collections, and should such sets be nested? Assuming that many low-level details (involving scales, coordinates, point-set operations, and so forth) are defined by the environment in which annotations are constructed, and therefore applicable to annotation-sets as a whole (rather than being details of individual annotations), to what degree should individual annotations be capable of overwriting the default properties of the set to which they belong? Can a single annotation use a different scale of measurement, or coordinate system, for instance, or treat as intrinsic data visual details that would normally be considered presentational?

13. What mechanisms should be employed to describe calculations performed as part of annotation data and/or within parameters stipulated through annotation data (e.g., the length of a line segment)? How should we attach representations of arbitrarily complex mathematical expressions and/or algorithms that may be involved in such calculations?

14. How should different kinds of curves that could be algorithmically described and/or rendered be supported as annotation shapes? Should an annotation system represent only a fixed selection of curve-types or should the options be open-ended, with a mechanism for supplying external procedures to perform calculations on special curve-types, whose mathematics are opaque to the system?

15. What varieties of calculations should be recognized by default as plausible operations that would typically be possible for annotations in general, such as area and perimeter, boundary-crossing counts, notions of geometric center (via minimum circumscribed circle, say), and so forth? Should the suite of such calculations be fixed *a priori* or could one facet of an annotation-environment or annotation-set be the collection of computations which could be activated as operations on typical annotations? In the case of user-designed curves, should implementations of calculations matching a particular purpose (e.g., area or perimeter) be required as a "contract" for recognizing a particular kind of curve as a valid shape-type?

16. How should textual data (whether free-form or constrained by controlled vocabularies) be represented when used to comment on or describe the role or the salient features of an annotation? Insofar as annotations are created to describe biological phenomenon for which the ground image is deemed to show evidence, how should annotation data be connected with data structures that describe such biological details in a rigorous fashion, e.g., through standardized terminologies or indicators, such as (say) **RadLex** diagnostic codes, or should connections between annotations and bioinformatic descriptions be asserted outside of annotations themselves?

17. How should details about the ground image be represented within the annotation, and how should the image that annotations target be identified? Should annotations targeting the same image be grouped together such that pertinent image-details (e.g., color depth and dimensions) are available as data within the group of annotations, or should that data always be provided through a separate object or structure representing the image itself?

18. Should annotations always refer to one single (ground) image, or should there be a mechanism for defining annotations either as applicable to more than one image (in the context of image-processing pipelines, or perhaps multiple similar 2D slides from a 3D or 4D resource) or as targeting one image, which is a transformed version of some previous image (so as, typically, to facilitate the derivation of annotations or the extraction of features, which the annotation calls attention to, for instance when analysis is performed on a morphological simplification of an original ground image)?

19. How should the provenance of annotations be described; factors such as whether they are created by people or algorithmically, what is their diagnostic (or explanatory or computational) purpose, their intended use (are they visual cues to observed features in an image or precise mathematical representations of image features or regions/segments)?

20. How should annotations model data which is best presented via derivative images with a similar format to the original image data, rather than via geometric structures such as shape data? For example, consider regions demarcated with a black-and-white or (for fuzzy regions) grayscale image overlaid on the ground image; thereby partitioning the ground into an interior point-space (black) and exterior (white). The purpose of a derived image in this case is to replace a geometric annotation-shape with a more granular and precise machinery to isolate distinguished regions. How can annotations defer their shape data to images in this sense?

21. How should annotations notate image features evinced in regions demarcated by the annotation, particularly in cases where the region is segmented specifically because it tracks the area where a given feature is present, e.g., region boundaries are the outermost points where a certain texture exists in (some neighborhood within) the image, or edges where the conformity of the image to a textural pattern crosses below some threshold? How should (potentially complex) computations or data structures endemic to image-features be oriented and connected to the annotation proper?

The above list amounts to circa 20 questions, or more given that some above items span multiple more specific questions. This is not necessarily an exhaustive list, but hopefully it indicates the scope of detail, which can arise when contemplating a general-purpose framework. It is also true that frameworks may adopt narrower scope, which could pre-emptively resolve some of the issues itemized above. For example, AIM does not deal extensively with image features or with computational geometry (apart from character-string notation of calculations), which obviates the need to bridge annotations with other kinds of computational data (at least within annotation-data proper). However, annotations are typically used in software environments, which have a broader scope than annotations alone, so the questions we have identified may still be in effect, merely deferred to the software utilizing the annotation framework than encompassed in the circle of concerns articulated by the framework proper.

With that said, the problem of integrating with other data domains and broader software ecosystems is a good rationale for an annotation framework to address a relatively wider scope, which draws in the suite of questions we have identified. Assuming going forward that these are all consequential questions, then, we will make some general observations about them.

8.1.2 Procedural data modeling (and the limitations of ontologies)

Our first claim is that image-annotations provide a case study in the limitations of "ontologies," at least in the semantic web OWL (web

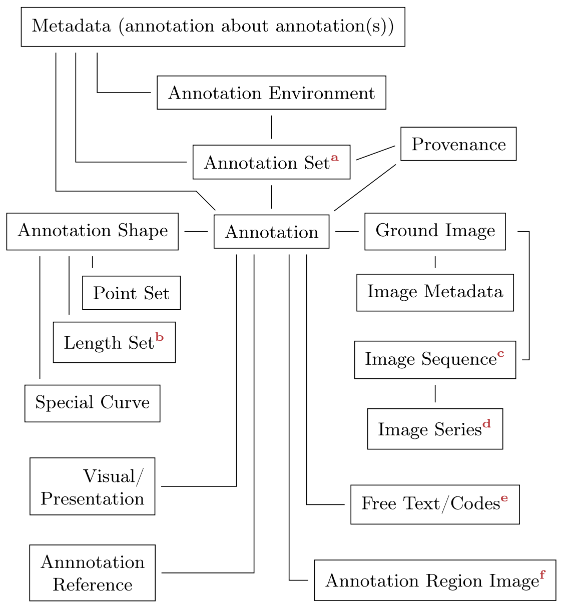

FIGURE 8.1 Diagram of annotation-associated datatypes. [a] Annotation Sets could potentially contain other annotation sets. [b] We use "length" set to denote the set of any magnitudes needed to geometrically specify the desired annotation-shape. [c] Image "Sequence" referring to transformed versions of a ground image (after grayscaling, morphological operators, or related modifications intended to make the image more amenable to analysis). [d] Image *Sequence* referring to logically related ground images. [e] Codes could be one or more diagnostic/prognostic terms selected from a Controlled Vocabulary or could be embedded as keywords in ordinary (free-form) text accompanying the annotation. [f] "Annotation Region Image" refers to a mask-image (e.g. a binary or grayscale image) intended to demarcated the contours of an annotation shape.

ontology language) sense (and certainly of specifications that might be designed to play a role akin to "light-weight" ontologies, such as XML schemas via document-type declarations).

To be sure, it is possible to define an ontology-like network of classes associated with image-annotations (Fig. 8.1 outlines the principal classes that might be involved). But such an outline does not articulate the range of implementational possibilities, which are intrinsic to the image-annotation domain (and summarized in part by the above 20 or so questions). Consider specifically, for example, the notion of annotation *sets* and annotation *environments*. The rationale for positing these classes is first that many low-level details concerning how annotations are defined (such as coordinate systems, the details of point-set construction, the range of recognized shapes, and so on) are unlikely to vary from one annotation to another. If annotations are constructed by a human user, for example, most of these details would probably be defined by the software that user uses. If they are constructed via computer algorithms, the code libraries providing those algorithms might similarly define a basic quantitative model that would underlie the overall process; this model would presumably be shared among all annotations generated as such.

It would then be memory inefficient—and an inaccurate logical representation lines as data structures *within* individual annotations. Even if it were possible to override certain defaults on an annotation-by-annotation bases, the correct logical gloss is still that a specific quantitative model (reflecting issues like coordinate systems and shape-geometry options) is endemic to the environment where annotations are constructed and will engender details shared amongst annotations by default. These are, in short, properties of annotations grouped by a common origin or environment, rather than properties sited in single annotations (in the default cases).

This therefore points to one aspect wherein annotations can be logically grouped together: those that emerge from a common environment will tend to share low-level operational and quantitative details. However, there are other criteria through which annotations could be aggregates: a common ground image or image-series, or gathering all annotations on a group of images brought together by the human annotator. Presumably, one annotation *environment* could include or engender many annotation *sets* in this sense. Insofar as every set would inherit low-level details from the environment where it is created, such details logically belong to the environment more so than any specific set. Moreover, annotation-sets can be collected for different reasons (e.g., a shared image or image-series) and could moreover be nested hierarchically; the criteria for aggregation and the parent/child relationships would seem to be points of information that are intrinsic to annotation-sets, but not to annotation-environments (insofar as the environment defines the basic operational and mathematical conditions wherein annotations are constructed). In this sense *sets* and *environments* play conceptually distinct roles, which is conveyed by modeling them as distinct classes.

However, such a rationale does not settle the question of what details should be modeled at the environment level or the set level, or indeed (to the degree that they may vary on a case-to-case basis) at the annotation level itself. Where, for instance, should the coordinate system (at least for default cases) and the Cartesian zero-point be defined, as a property of annotation environments or sets (or even individual annotations)? An ontology could simply select a “distribution” of information, which seems most appropriate for the widest range of cases. Ontologies need not be completely open-ended; it is the possibility of structural variations that allows there to be different ontologies. But this example illustrates how structural choices are not isolated; the effects of choices specifically local to one “part” of an ontology can propagate across the framework.

Assume, for example, that annotation *environments*, *sets*, and individual *instances* have a roughly hierarchical relationship: one environment produces many sets, and sets contain multiple instances. Low-level details would typically then be inherited from higher-up in the hierarchy downward. If information is distributed across all three levels, then it is only possible to use individual *instances* by taking information from the *set* to which they belong and the *environment* according to whose rules they are constructed.

In practice, this means that an application needs to obtain an environment object as a context for using an annotation *set*, and needs to obtain an annotation-set object as a context for using single annotations. Assume now that annotations are retrieved from a database: the query interface to that database would then have to be organized so that objects correspond to this order of initialization and inter-dependency. How should a query which intrinsically returns *one* annotation deal with the enclosing sets and environments? Should a query interface start by loading an environment, using that as a parameter to further queries, so that the handle to an environment object is prerequisite for a single annotation? Or should a query-result for individual annotations be augmented with indicators for the annotations’ respective sets and environments so that those objects can be retrieved if not already? In the latter alternative, given query results, including multiple annotations, how should data structures associating annotations with their sets/environments be described? Query types that are more complex (involving multi-layered structures, rather than individual values or simple list-like collections) require proportionately more coding steps to parse.

Moreover, should environment and set-level defaults be imposed uniformly, or should they be overridable down the hierarchy? The former option yields a framework, which could be too rigid for some use-cases. On the other hand, suppose we opt to make a variety of implementation defaults overridable. This entails *first* that procedures must be available to construct the alternative setup, where it departs from the defaults. And *second*, because it then cannot be assumed that annotations and sets thereof inherit low-level details ubiquitously from the environment, there must be a mechanism to flag whether a given annotation does or does not encompass any such overrides, and, if so, to define them.

This book's demo code allows annotations to include a "default override" object, which provides an interface for declaring an environment for a given annotation that differs from environment defaults in various ways. However, the demo annotation class also uses pointer-union types to restrict the memory-size of individual annotations, considering that many data structures needed by *some* annotations will be superfluous for many others. Analogous issues would come into play for databases persisting annotations: the flexibility of allowing numerous extra structures (for overrides, special curves, point/length-set roles, etc.) for *some* annotations has to be balanced by techniques for compactifying such structures when they are "empty" or unused vis-à-vis annotations where they are unneeded.

In short, choices such as how to distribute information across hierarchy levels tend to propagate to implementation choices involving memory organization, database queries, procedural requirements for data types that take on roles specified by ontology classes, and software engineering in general. It is difficult at the ontology level to represent the details of implementation choices, or the range of choices available to an application vis-à-vis specific concerns. One might reply that ontologies are intended to be schematic models of domains, not rigorous blueprints for software implementations. That is true as far as it goes, but implementation choices determine the degree to which software components are interoperable: if a given ontology can be realized via architecturally incompatible software, then ontology alignment alone (as opposed to software-engineering and data-exchange protocols) is not sufficient for interoperability.

The idea of *protocols* serving as a contrast to *ontologies* can be further illustrated with an example we alluded to earlier: consider the issue of annotations' point-set ordering. An annotation "domain model" can recognize different possibilities for enforcing or interpreting the order present among points in the annotation shape. These can take the form of axioms (all point-sets which are permutations of each other should be considered equivalent) or stipulations (point-sets should be automatically ordered during construction) or classifications (point-sets should be subtyped as clockwise, counter-clockwise, nonconvex, or self-intersecting, with different ordering-specifications depending on the subtype). However, implementation-wise, these various options become concretized through groups of *procedures*: procedures governing how point-sets are modify upon inserting a new point, in cases where it is required that point-sets remain ordered; to compute the proper position for a point relative to an existing set; to permute a point-set into a proper order; to determine whether an ordered point-set is oriented clockwise or counter-clockwise; or to determine if a point-set is orderable in the first place in terms of vertices of a convex polygon. Similar comments apply to questions about colinearity: however that issue is addressed, any resolution depends on procedures such as identifying when a given point is colinearly between two outer points, and filtering intermediate colinear points out of a point-set when that would be appropriate.

These examples point to how image-annotations are a representative case study for the general phenomenon which we highlighted in Chapter 6: rigorous documentation of data models tends to depend on code models that instantiate them. In particular, code models can describe the roles and pre-/post-conditions on individual procedures as well as how procedures are interrelated into logical groups. Decisions concerning how information is distributed among annotation-environments, sets, and instances translates into the procedures used to construct an environment anterior to individual annotations being processes, and into procedures for overriding environment defaults when necessary. Decisions concerning point-set orders and normalization become manifest in procedures through which point-sets are modified and geometrically examined. In these examples resolutions to concerns examined abstractly within generic data models can only be concretely documented at the procedural level.

This book's demo code represents one possible implementation of an annotation class; it is not a definitive example of how annotations *should* be defined, but it serves as a case study in the *kinds* of procedures that are intrinsic to defining a relatively general-purpose annotation class. This image-annotation class and its peers use source-code annotations to document logical relations between procedures, and also employ various binary-representation techniques to make annotation data memory-efficient. The resulting *code* model is more fine-grained than an annotation *data* model, which is implicitly instantiated by this code; it would be possible to concretize similar data models with a code library that differs from our demo in many low-level details. As a result, it might seem that the particulars of our annotation-related classes are implementation details that should be separate from higher-level data models. The purpose of data models is less implementation-specific, but rather focused on defining common requirements so that components whose low-level implementations may differ can nonetheless interoperate and exchange data.

The more that data models are abstracted from implementation details, however, the more that particular implementations may need to provide bridge code, which marshals data into a common format for purpose of networking and data sharing. The degree of extra code demanded to get two distinct software components to interoperate tends to be proportionate to the degree to which their low-level implementations diverge from one another. The best data-integration protocols tend to be receptive *both* to abstract data models *and* to implementation concerns, in that sensitivity to propitious implementational design patterns can help protocols adopt formulations that minimize the bridge code needed for implementations to participate in the protocol.

8.1.3 Different aspects of image-annotation data

As we have proposed, in multi-aspect modular design, each module is responsible for managing software concerns that cut across multiple facets of software development. We are focusing on the four aspects of procedural exposure, data persistence, serialization, and GUI design (which together fit into the “semiotic saltire” schema that we introduced in Chapter 6). For the sake of discussion, we will assume that a module responsible for image-annotations adopts data models similar to what we outlined earlier in the chapter, with details involving ground images, visual presentations, and annotation sets/environments represented as contextual parameters coexisting with annotation data proper (see Fig. 8.1 for an outline).

Assuming that annotation data (and the relevant suite of contextual data) is organized along those lines, then, we can examine how this data model projects onto the different concerns of the “saltire.” Here is a summary:

Visual Objects and GUI Design First and foremost, of course, annotations have to be rendered against the background of their ground image. Analyzing GUI design patterns in a modular context requires some care, because it may not be obvious which GUI elements belong to which modules. In the simplest case, each module would be responsible for its own suite of self-contained GUI objects, particularly autonomous windows. That is, any given window within an application would be designed and populated with data entirely from a single module; insofar as the application integrates multiple modules, their coexistence would be manifest in (potentially) multiple windows being open at one time. In practice, however, modules may need to be more tightly coupled than strict separation of windows allows, particularly in the GUI context. In the case of image-annotations, it would be reasonable to factor details of image *acquisition* outside the scope of the annotation module, so that the latter would not be responsible, for example, for finding a specific image within an image-series or archive, or giving users a chance to load a new image to replace the one they are currently given. Concerns for annotations proper would then be woven with handlers for user actions involving searching for and loading images themselves (to be discussed further later this section).

Serialization We discussed some issues related to serializing image-annotation data in the context of AIM (the Annotation and Image Markup project) in Chapter 6. As we then intimated in Chapter 7, image-annotation data models could potentially be broadened in scope and made significantly more flexible than AIM itself, which would require extending AIM's serialization model to incorporate a wider range of contextual data and annotation details. Fine-grained details of annotation serialization is outside the scope of this chapter, but we can make a couple of general points. First of all, we contend that the structural rigor of serialization formats should be enforced by client libraries as much as (or more than) via document-level constraints on serialized data. In the case of image-annotations, the primary role of serialization is to encode annotations within one application so that they may shared with other applications running elsewhere and/or later. Assuming that the receiving application also obtains a copy of the relevant ground image, the serialized encoding should permit the second application to reconstruct the annotations in exactly the same form as they were (within the original application) constructed and superimposed on the ground image.

The best way to achieve the proper alignment between sending and receiving components is to use the same code base on both ends; therefore to implement low-level serialization details, it would be good for image-annotation modules to provide a code library that can be re-used across different endpoints, insofar as annotation is shared in serialized form (in principle such a library should be isolated from and not dependent on the module as a whole). The canonical example of a serialization-deserialization cycle would then be a situation where two different applications both use the same code library as the specific locale for procedures directly responsible for constructing and/or parsing the serialized formats.

Of course, one rationale for curating a standardized serialization format is to free applications from depending on one specific code library. As a result, the module's serialization format should be designed to facilitate alternative libraries, which could parse the same data. This is one reason why serialization formats often stress standard models, such as an XML document type declaration (DTD), which can serve as a reference-point for developing multiple different code libraries that share a common serialization format. In general, confirming that a document (i.e., a file or a character or binary stream) conforms to a given serialization format is a separate process from actually parsing such documents, and can serve as a useful preliminary step (so that parsers can assume as a precondition that any strings they are presented with conform to the standard). For these reasons, formats (or meta-formats) such as XML are popular, because these kinds of preliminary checks can be simplified by (say) DTD's; however, XML (and other popular formats, such as JavaScript object notation) can also be limiting in some respects. Modules could certainly turn to more flexible and expressive data formats instead. It is reasonable, however, in those cases, to provide code for document *validation*, which is less complete than (and preliminary to) full parsing/deserialization. In this manner alternative libraries designed for other programming environments (e.g., other programming languages) can mimic the validation code, separately from emulating (to whatever degree is appropriate) the actual parsers.

These comments apply to serialization aspects in general. For the more specific issue of image-annotations, note that data models in this context (at least insofar as they roughly follow our outline from earlier this chapter) are characterized by a relatively large number of default options that will generally be shared among multiple annotations, but with the possibility of many defaults being overridden on a case-by-case basis. In this scenario validation code for serialized documents may be relatively complex, because the core serialization for annotation proper may or may not have numerous residual structures encoding extra details. Techniques for managing variegated "shape constraints" in this specific context (see Section 6.4) are represented, albeit not with full rigor, in the demo code accompanying this book.

Data Persistence Questions about database representations for image-annotations can be intricate, because we have to identify which portions of annotation data are "queryable," in the sense we discussed in Chapter 6, that is, what parts of annotation data should be visible to queries against a database. Presumably, there would be few occasions where geometric minutiae (e.g., the precise details of annotation shapes) would be the subject of database queries. However, the text labels/descriptions and diagnostic information provided with annotations could certainly be represented in a queryable fashion. The simplest scenario would be encoding descriptions as ordinary textual data, but there is also a rather extensive literature on technologies such as "semantic DICOM" [38], which associate annotations with diagnostic codes (including treatment plans) and/or controlled vocabularies (e.g., RadLex, the "radiology lexicon"). Such codifications are intended to make it possible to search large-scale databases of annotated biomedical images via diagnostic, treatment, or biomarker terminologies.

Annotation characteristics such as *calculations* lie somewhere on the spectrum of detail between text and/or controlled-vocabulary descriptions and granular geometric representations of annotation-shapes. Proposed semantic DICOM protocols include the ability to query image data-sets for quantitative data that can be defined within (or via) annotations. A canonical example, used on the starting page of the **SeDI** project (https://semantic-dicom.com/starting-page/) is "Display all patients with a bronchial carcinoma bigger than 50 cm^3" (*see also*, e.g., [22], [37], [39], [11]). Calculations depend on fine-grained geometric details, so making this kind of quantitative data available for *queries*, where a query engine has to scan over many different annotations (without the option of reconstituting annotations individual to precisely examine their geometric properties) requires some functionality to classify calculations into queryable categories (e.g., "tumor size"). We will not examine this process in detail, but note that semantically this involves combining annotation-specific data with biological interpretations, and therefore belongs to the larger issue of representing biomedical findings warranted *through* annotations alongside annotations themselves. In this broader guise, we will return to this issue later in the chapter.

Procedural Exposure In keeping with our discussion in Chapter 6, a good starting-point for considering which procedures to "expose" for runtime reflection and remote-invocation scenarios is to look at those procedures that intrinsically implement, support, or operationalize important details of a modules' *data* model that informs its associated *code* model. At this point we can look at some concrete examples in the image-annotation domain.

As we reviewed above, one concern for a general-purpose annotation framework is how to specify the numeric and measurement dimensions applicable to annotations. There are several options, including using template classes, which would be specialized on number and scale types (e.g., **double**s and millimeters, as one possible combination). Alternatively, classes within annotation data, such as point-sets and length-sets, could be modeled as base classes, for which dimensional details are unspecified, allowing subtypes to be implemented with specifics such as (say) **double**/millimeter. The demo presents a third option, namely encoding a variety of dimension/numeric configurations within individual data types, which we will reference momentarily for sake of discussion.

The basic idea here is to use flags and/or enumerations to differentiate number/scale possibilities when these are ambiguous. For example, the demo code uses quasi-real types (which can be **typedef**'d to something other than **double**) for length and coordinate values, but supports a flag restricting these values to integers. When (but only when) that flag is in effect, any procedure initializing a length or coordinate value is truncating to an integer.[1] This is an example of procedural locations enforcing a data-modeling choice, and the procedures that carry out these checks (as well as those manipulating the relevant integer-only flag) are likely candidates then for exposure. Similar comments apply to measurement-units. The demo assumes that lengths and coordinate values can be provided in numerous different units, which are an intrinsic part of the value,[2] but also provides procedures to convert each value to a different choice of units, and to synchronize two different values when it would be unreasonable to have a mixture of disparate units (e.g., for arithmetic operations and for composing coordinate pairs; one operand or element of the pair is recalculated in terms of the other's unit scale).

With respect to GUIs, determining the proper scope of an annotation module's responsibilities is complicated by the fact that image *annotations* might be separated from management of *images* themselves. Annotations and ground images obviously have to be displayed together, but other software-engineering aspects between the two facets (images versus annotations) do not necessarily coincide; serialization and data persistence for annotations can be separated from the corresponding operations for ground-images, apart from the annotation maintaining a reference to the corresponding ground image through some sort of identifying code or file/web location.

This leaves open the question of how exactly to implement GUI components when the visual artifacts seen together may be the responsibility of distinct modules. It would be theoretically possible for multiple modules to interact within single GUI components; for example, consider a scenario where the ground image is a node on a **QGraphicsScene** object (this assumes that the relevant applications is composed in C++ with the Qt GUI libraries). Annotations can then be further nodes drawn on the same graphics scene, and procedures in the annotation model can be presented with (a pointer or reference to) the **QGraphicsScene** instance, without managing other GUI objects.

For the sake of discussion, however, assume that the image-annotation and ground-image handling modules are more strictly separated, and that the annotation module handles its own GUI windows. In that sort of setup, this module would receive an object encapsulating a ground image (or perhaps a binary package with the raw image data), but would otherwise work in isolation to display the image, and then render annotations with reference to it. Most image-management operations, on the other hand,

[1] By "coordinate value" here we mean one value in a coordinate vector, canonically x and y as pair-elements in an image location.

[2] That is, each value encodes a magnitude and also a code marking which measurement scale to apply to it.

would be delegated to separate modules responsible for images themselves, as will be outlined in the next few paragraphs.

8.2 Annotations and radiomics

We feel that image-annotations serve as a good case study for issues surrounding modular design, which is why we have devoted space to annotations and their associated data structures. However, the practical consequences of modular design-choices may be more apparent when considering the larger bioinformatic systems, wherein annotations are a relatively small part, so we turn to this larger picture next.

8.2.1 GUI operations involving images and image-annotations

An annotation module meeting the general profile just outlined—maintaining a self-contained window showing ground images and annotations drawn on them, but delegating image-management to other modules—would presumably need to be paired with modules through which users find images-of-interest in the first place. Applications would invoke procedures in the annotation module only after loading image requested by the user through a database query, file system search, or simply opening a local image file.

In this scenario the annotation module would have capabilities to display each ground image itself, but in general would not implement separate GUI classes for other operations related to images proper (as compared with operations pertaining to actual annotations, that are its proper domain). For example, the annotation module might skip implementation of functionality to open a new image (replacing the current ground image with a different one), or to search for images (in a database or file system) that users would load to replace the ground image currently viewed.

With that said, however, it would certainly be plausible that responsive GUIs would allow users to indicate the desire to load and/or search for a new image via actions performed on the GUI objects managed by the annotation module. For example, a context menu activated on points within the background image (at least those not covered by an annotation) could have, as one of its menu items, the option to look for a new image to load in lieu of the one currently being rendered (along with its annotations). The module would in that case, presumably, indicate via a signal-emission or procedure-call that the user intends to search for a new image, and delegate to other modules or components the process of showing dialog boxes or similar tools responding to that request.

Assuming this overall plan for routing requests, there are then two different scenarios where an annotation module would be rendering a new image and its annotations (new in the sense that the image is not currently visible on-screen). One scenario is that the module was not active previously (during the current run of the surrounding application, anyway) or at least that any windows showing ground-images have been closed. An alternative scenario is that the module is managing an open window with its ground image and annotations initialized, and receives a signal that the user now wishes to view a different image which the user has opened or selected (potentially that signal could be indirectly a response to an earlier signal from the module itself asserting the user's preliminary desire to switch images). Identifying these two different sorts of "entry" scenarios can be useful for building a procedural model relating to the module's "procedural exposure" concerns.

Fig. 8.2 sketches out a few of the more important procedural steps that would be involved in proceeding from when the module receives an identifier or data structure for a new image to the point where this ground image (and its annotations) is rendered. This is not a very

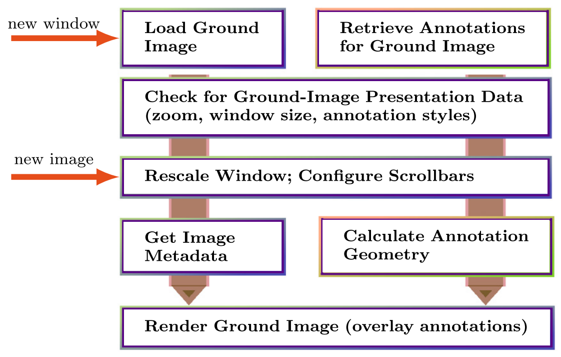
new window
Load Ground Image
Retrieve Annotations for Ground Image
Check for Ground-Image Presentation Data (zoom, window size, annotation styles)
new image
Rescale Window; Configure Scrollbars
Get Image Metadata
Calculate Annotation Geometry
Render Ground Image (overlay annotations)

FIGURE 8.2 Entry points for hypothetical image-annotation module.

schematic picture (leaving out many potential steps and not really clarifying the branching logic between showing an initial image and replacing the image on an already-visible window), but it is intended simply to give an impression of how procedures can fit together into logical sequences, and also how a module may have specific "entry" points. By this terminology, we mean that at entry points the module is receiving data from an external source and is also receiving notice that some procedural chain should be initiated. In this running example, there are two kinds of "entries" in this sense: one is initializing a new window with the image-data provided, and the other is replacing the image with another viewed within the former image's windowing context.

In general, it is worthwhile to tag every point where a module may receive data from an external source; it is also worthwhile to tag where an external source may trigger (or request) a chain of actions that the module implements. An "entry point" would then be a location (e.g., a join-point; see Chapter 6) where both of these conditions are met. Rigorous procedural models should accordingly identify such entry points, and the code should be designed to facilitate this process. One way to do so is providing specific procedures for each entry point (i.e., each site in a logical model of what the module should accomplish, which would serve as an entry point). The entry-point procedures might call other procedures, of course, but providing entry-point procedures that encapsulate the initial steps in a "logical" picture of modules' functionality (abstracting from implementation details) can then make it easier to trace the "flow of information" within the module. Moreover, once entry-point procedures are specified and identified as such, they are obvious candidates for being exposed to scripting, testing, and remote-invocation engines.

Many systems exist to diagram procedures in a module (or code library, software component, or analogous body of interrelated code) and the logic of how data passes between them, as well as how groups of procedures follow in sequence to perform a specific task.[3] Analyzing these sorts of diagrams (their vocabularies and the attributes they may identify per procedure) is outside the scope of this chapter, but suffice it to say that once a procedural model is created (perhaps informally), such a model can serve as a template for establishing a module's "procedural exposure." Procedures that appear to be particularly important in recurring sequence-patterns are likely to be the ones that could be targets of script-based adapters, unit tests, and other scenarios, where such procedures might be invoked dynamically. Likewise, the collection of exposed procedure serves as an introduction to the overall functionality of a module, so that procedures that seem most functionally important in a module should be exposed, partly because they provide an overview for developers trying to become familiar with the module's implementation.

Image annotations have only limited use in isolation; in contexts related to bioimaging, the main purpose of annotations is to call attention to biomedically significant image features,

[3] As examples—without implying that these formal systems align completely with our summaries here—consider Petri Net techniques mentioned in Chapter 6, or, say, [15], [27], [5], [35], [23], [13], [36], [17], etc.

which are identified either by a human expert evaluating the image or by image-processing algorithms. Once identified, these image-features suggest additional facts about the biological material sampled within the image, and accessing such non-image data requires capabilities that presumably lie outside the scope of the annotation module proper. Accordingly, this module would need to provide users opportunities to follow up on image-view focused actions to those handled by other modules.

Because image markers can be one of the more effective means of discovering and presenting biomedical facts or predictions, hypothetical application-sessions, where users start with capabilities provided through an image-annotation module is a reasonable case study for analyzing issues in how modules would interoperate. This provides a rationale for emphasizing image-annotation as we have here, though the remainder of this chapter will transition to other forms of biomarkers.

8.2.2 Image processing in the context of broader-scale workflows

Bioimage analyses are usually most valuable when they can yield relatively simple data structures that have unambiguous biomedical interpretations. For example, image segmentation in the context of cervical cancer yields a classification of imaged nuclei into different bins, where one category corresponding to likely cancer cells, typically on the basis of one or two derived parameters, such as measurements of nuclear enlargement and deformity (relative to healthy cells).[4] Similarly, the density of blood vessels within tumor microenvironments can be estimated via "fractal dimensions," which measure the cumulative length of blood-vessel structures connecting tumors to surrounding tissues.[5] In general, we want image biomarkers to yield statistical measures from images that correspond to bioinformatic quantities yielding diagnostic and/or prognostic results: What proportion of cells in a blood sample are cancerous? How aggressively has a tumor started to "colonize" surrounding tissue? Where has scarring diminished blood supply to damaged heart tissue? How extensively has SARS-CoV-2 infection compromised lung functioning (visible due to "ground-glass opacity")?[6]

Often these sorts of biomedical questions, for which bioimaging can provide partial answers, may also be answered in part by other means, such as genomic tests or biochemical assays. This opens the possibility of bioimaging and (say) biomolecular results mutually reinforcing one another. For example, immunohistochemistry staining (IHS) is often used in biopsies diagnosing many forms of cancer: antibodies are introduced into a tissue sample, causing the sample to be stained whenever specific proteins are present that bind to the introduced antibodies. There are hundreds of different IHS antibodies that can potentially be utilized, depending on which specific proteins are targeted. Assessments of the quantity of target protein evident in a sample can itself depend on image-analysis, usually comparatively simple measurements of the degree and intensity of sample staining. Immunohistochemistry (IHC) and IHS are one of several antibody protocols; flow cytometry, for example, provides an alternative (*see*, say, [40] or [24]). In this context immunostaining is based on immunofluorescence antibodies, rather than visible staining, resulting in cells containing the target proteins emitting fluorescent signals that are detected using FCM equipment.

The visible final stages of an immunostaining assay may therefore be a microscopy image showing obvious staining patterns that can be

[4]See Section 4.2.2 earlier in the book for references.

[5]*See*, e.g., [42], [9], [14], [43], as well as [1, page 4] and [34, page 6] cited last chapter.

[6]*See* [20], [29], [31], etc.

graded by human or software analysts, or may potentially be FCM plots, where each cell corresponds to an "event," wherein the cell has generated fluorescent signals captured by an FCM channel (this data can then be modified via FCM methods, such as dimensional transforms and gating to yield an intuitive visualization of the event-data).

The resulting visualizations, however, are only the end stages of complex assays that require disciplined lab methods during sample preparation and analysis. Bioimaging in the context of biochemical assays is different from non-invasive image-process using radiology, for example, to picture solid tumors. With diagnostic assays, modifications are induced within a tissue sample to control how the sample's physical properties (e.g., the presence of one or more specific proteins) becomes visually expressed under bioimaging. Scientists can physically manipulate the samples to be imaged, instead of or in addition to tweaking the image-analysis process, to improve diagnostic accuracy. At the same time, this means that data concerning how tissues are sampled and prepared to be modeled alongside of the image-data proper.

In a paper devoted to more precise IHC quantitative methodology, for example, [18] (also discussed in Chapter 4) presented the following argument:

> The commercially sourced antibodies in [IHC] protocols often vary in their specificity and sensitivity, thus requiring meticulous optimization and testing. Hence, there is a need to enable users to rapidly screen parameter spaces to determine practical assay conditions for the specific biochemicals used. The ability to *quantitatively characterize* detected signals, perform *multiplexed detection of various antigens simultaneously*, and *rapidly implement IHC protocols* that are standardized or modified according to user need are facets of research that would be essential to fostering the next generation of methods in cancer research and diagnostics. ***(page 1)***

In other words, the authors are implicitly calling for greater standardization *both* of "assay conditions" and assays' "parameter spaces" *and* of the algorithms generating their quantitative results. For their specific workflow, [18, page 3] identify several adjustable parameters within the assay protocol, such as temperature, "the flow velocity of the processing liquid" and "incubation time" (because these authors' methods depend on a two-step binding process involving two different antibodies, the ratio of the second and first incubation times is also a consequential data point (page 8)). In a prior article [12], the same authors also discuss novel quantitative techniques for measuring assay results, such as what they call "saturation approach matrix" (SAM). The SAM metric considers not only staining intensity once antibodied have been maximally bound to their target "analytes" (e.g., proteins), but also the rate at which these reactions occur. In this sense, SAM data is not only a form of image-biomarker derived from static images, but an (indirect) measurement of biochemical reactions occurring prior to the assay's final state (whereas conventional analyses would consider only the final state in isolation).

The quantitative techniques introduced in [12] depend on "microfluidic probes" (MFPs), which can freely move over the surface of a sample and (if desired) modify the sample at specific points, e.g., by introducing new material, while also picturing a local region of the sample around its current target point via an inverted microscope [10], [32], [3], [7], etc. Although MFP devices capture image-data via integrated microscopes, they may also be equipped with sensors that read signals in other media; [28, page 7], for example, propose "MFP ... combined with microelectrode array technology to study the electrical changes in neuronal networks in response to topical application of neuromodulators" (they cite in turn [26]). The MFP probes, in turn, are one example of "biosensors" that collect data about tissue samples by detecting optical, electric, or fluorescent signals. Whereas older biosensors tended to be fixed in place, recent technologies, such as MFP enable probes to freely move around a sample's surface. In some cases, biosensor data is mapped against single locations in a sample, so that the data has characteristics of a bioimage (wherein individual pixels roughly correspond to individual "points" on the sample). However, this image-like data is not necessarily acquired via the same physical mechanisms as a microscopic or radiographic picture. For example, a popular class of sensors is constructed according to the physics of "surface plasmon resonance" (SPR), where sensors detect "modulations" in patterns of light waves' resonance against a biological sample (when it is introduced into a specific configuration of instrument layers involving glass and gold) [41]. The "picture" that emerges infers properties of the underlying sample via properties of light waves, but instead of only capturing wavelength (color) as in an optical microscope, SPR results in more complex refraction-modulation data, which are then subject to digital processing.

As examples, such as SPR and fluorescent flow cytometry illustrate, there is a wide range of physical mechanisms by which properties of a biological sample can be investigated via properties of light waves that emerge from or interact with biological materials. What we conventionally call "images" are only one manifestation of the larger principle that information about objects is encoded in the light that they emit, refract, and/or reflect. Consequently, it becomes unexpectedly difficult to identify the proper scope of domains, such as "bioimaging" and "image annotation."

Should *images* in these contexts refer to the classical sense of a "picture," which encodes information via *colors*, with the color of a single pixel being considered (as an idealization) the color of light reflected from a miniscule patch on the surface of the imaged medium, analogous to human vision? Or should we accept a broader notion of *images* encompassing a wide range of technologies, which acquire spatially extended data about a sample by probing light waves for different physical properties, not only in the sample's "natural" state, but potentially after probing the sample with external energy-sources, such as lasers? Moreover, what about physical phenomena *other than* light waves, such as piezoelectric sensors?

These questions are not philosophical speculations on the "nature" of an image; more concretely, they address problems of integrating biomedical information acquired from a heterogeneous suite of devices. If we remain within the context of "classical" images, e.g., those derived from optical microscopy or radiography, there is a relatively well-defined ecosystem of digital formats and software capabilities, reflecting shared assumptions about the nature of "image data," which different software components will all be acting upon. For example, images are encoded within several canonical formats, such as JPG, PNG, and TIFF, and there are specific kinds of meta-data, which are ubiquitous as preconditions for properly reading image data, such as dimensions/resolution, color depth, and the orientation of orthogonal axes relative to the pixel matrix. Analytic notions, such as regional contours, morphology operators, color transforms (such as grayscaling), image-masks, color averages and interpolations, region areas and perimeters, and so forth, are relatively consistent across different image formats and image-processing methods. We can, for example, ask about (say) the area of a region measured as a proportion of the area of its minimum enclosing circle, or about the morphology of its convex hull, whether we are calculating these results via OpenCV or ITK (to mention two popular image-processing libraries) and whether the underlying image is saved as a PNG or TIFF file.

Because of the relative consistency among different image-processing and image-viewing applications—if we stay within the "classical" imaging scope—it is possible to investigate image-analysis pipelines as generic workflows abstracted from the specific software tools with which the workflows would ultimately be implemented. Moreover, analytic libraries can be paired with image-viewing software and image-acquisition sources in different combinations. The process of obtaining raw image from, for example, a DICOM image series, subjecting those images to a predefined analytic method, and visualizing the results via image-viewing software, is sufficiently standardized that some variant of this methodological outline can be implemented on a wide range of components. This standardization allows users' understanding of image-related software to largely carry over to other software (even if it is implemented in a different programming language, distributed in a different commercial or open-source environment, and so forth), and allows image-related software components serving different roles to interoperate.

Widening the scope of "bioimaging" *outside* classical image-acquisition modalities, on the other hand, has the effect that this degree of standardization and interoperability becomes diminished. The terminology and mathematical models for manipulating "image" data deviates from "classical" images the more the physical mechanisms of acquiring this data deviate from microscopy or radiography. For example, flow cytometry gating evinces idiosyncratic terminology and quantitative frameworks despite the fact that geometrical principles behind gating overlap in many respects with concepts from image annotation (leading to proposals in the FCM community, for example, to integrate their own data models within DICOM; [16, page 1, e.g.] argues "The large overlap between imaging and flow cytometry provides strong evidence that both modalities should be covered by the same standard"). As we suggested earlier in the chapter, expanding the scope of data models too far runs the risk of such models being over-complicated with special use-cases and special-purpose extensions, which are tangential to the core foci of the model. On the other hand, failure to synthesize interrelated data models can result in the unnecessary "fragmentation" of software ecosystems, which give rise to (and are shaped by) data models, as we argued in Chapter 4.

8.2.3 Data profiles for annotation and image markup

Chapter 6 sketched the rationale for **aim-client**, a code library included as supplemental materials for this book which facilitates the use of data sets employing AIM annotations. This section will continue that discussion by delving further into the AIM data model and how we try to refine certain elements of AIM's semantics with **aim-client**. In practical terms, **aim-client** is first and foremost a utility library for deserializing AIM data. We demonstrate **aim-client** through sample data sets published alongside this book, where **aim-client** can be used to programmatically study **AIMLib**, aside from the extra features added by **aim-client**.

Since it was standardized, **AIMLib**'s adoption in radiology and related bioimaging fields has been driven mostly by clinical software, which adopted the AIM format (as one mode for exchanging annotation data, among others, such as DICOM-SR (DICOM structured reporting)). We are not aware of software packages that provide access to AIM data in a standalone fashion, outside the context of a larger bioimaging application, which could potentially stymie researchers who wish to examine library code directly to get a more thorough grasp on AIM data and architecture. Potentially **aim-client** can help in this regard, because **aim-client** sets up an environment where AIM-compliant data (such as XML files) can be read and parsed, yielding instances of AIM classes that can be examined at runtime (e.g., through a debugger).

This book's republished data set is bundled with code in the form of a Qt project, which links against both **AIMLib** and **aim-client**; users can accordingly load sample AIM-compatible XML files in a Qt environment. For example, a natural way to use the **aim-client** project is by running the code in QT creator, a C++ integrated development environment (IDE) particularly targeted at QT projects. The IDE's debugging features (for setting breakpoints, examining local variables, searching the code base for symbol-names, and so forth) can then be exploited to decipher AIM data structures after they have been initialized from XML files.

Given a properly formatted XML serialization, then, we can use the AIM **XmlModel** class, which provides a **ReadAnnotationCollectionFromFile** method that returns an **AnnotationCollection**. This latter type is a base class, whose subtypes differentiate annotations-on-annotations (metadata) from annotations-on-images, which are our primary concern. The **ImageAnnotationCollection** type is a convenient way to accommodate the fact that some images may have multiple image-annotations (multiple regions-of-Interest), even though in practice an image may have only one, so the "collection" is actually a holder for one *single* annotation-instance (this is the case for all the files in the sample data set). Most of the important structure, then, lies with this **ImageAnnotation** class. Each image annotation is a holder for collections of several object-types, including "image references," which connect the annotation to the image it annotates; "annotation statements" that assert the annotation's significance (i.e., its biological/diagnostic rationale); and "segmentation" or "markup entities" (which carry information about image-regions defined as geometric objects on or segments of the image).

In addition to the actual annotation, AIM needs to account for many data points concerning the utilization of annotations in a diagnostic setting, such as diagnostic codes, confidence-levels that the image-interpretation is correct, metadata concerning the graphics properties of the image itself and how it was acquired, biological descriptions of the phenomena or entities (such as cells, tissues, lesions, etc.) visible at the region-of-interest (assuming correct interpretation), and so forth. As a result, the overall AIM data model encompasses many data types, which are not explicit representations of annotation geometry itself. However, to organize the overall data model exemplified via AIM, it is useful to start with the model specifically representing geometrically described annotations, and then introduce diagnostic and biomedical metadata as refinements of that core model. In **AIMLib**, these fundamental data types are implemented via subclasses of a base **MarkupEntity** class. These subclasses, such as **GeometricShapeEntity**, are therefore a natural starting-point for examining the profiles of AIM data.

For the sake of discussion, we will mostly consider images and corresponding annotations in 2D (the 3D cases are similar, but harder to visualize). Positions in **2**-space are represented by AIM's **TwoDimensionSpatialCoordinate** class. Unlike other libraries that work with spatial data, such as the computational geometry algorithms library (CGAL), AIM only recognizes a single scalar magnitude for spatial intervals (a **double**, i.e., double-precision floating-point number). This appears to be an implementational choice, not a "semantic" one, in the sense that encodings of spatial regions and magnitudes can potentially utilize a diversity of quantifying strategies and coordinate systems. For instance, CGAL supports both normal Cartesian coordinates and also "homogeneous" coordinates, which are based on projective geometry, and allows coordinate systems to be parametrized on different kinds of scalar values, such as integers, rather than pseudo-reals (which arguably better matches the image domain for many purposes, since one cannot have a fraction of a pixel); or rational/floating-point numbers with varying degrees of precision (a simple example would be using single-precision **float**s in lieu of double-precision, which makes geometric representations more memory-efficient, or in the other direction using more exotic numeric types for greater mathematical precision) [6, page 14]. Also, some image-analysis algorithms are expedited using polar coordinates [25, for example], implying that polar-valued annotations also have a role in documenting image-features tagged by those algorithms.

These comments are tangential to AIM *per se*, because a **double**/Cartesian coordinate framework is probably the most sensible default option, and AIM use-cases may not warrant generalizing the code for other coordinate options that would rarely be leveraged. Nevertheless, this example points to how even a seemingly simple construction, such as two-dimensional spatial points can encompass a fairly detailed space of semantic alternatives. Code libraries that seek to operationalize a relatively thorough semantic model of the space-coordinate domain, encapsulating the diversity of representations that are computationally useful in different contexts for designating spatial points, regions, and magnitudes, would need to recognize a wider range of coordinate systems and numeric types than exemplified by AIM.

This case illustrates the kinds of situations where data-mismatch problems can arise: exporting data back-and-forth between **AIMLib** and other software components might require bridge code to handle coordinate conversions. Data-integration projects, for which AIM annotations represent one data source, would correspondingly need to anticipate the possibility of mismatches involving coordinate systems and the need for suitable bridge code as part of the integration workflow. Since AIM does not internally support a choice of coordinate-systems, using **double**/Cartesian exclusively, AIM also does not explicitly notate its choice of coordinate systems, so that it takes some exploration of the **AIMLib** code to grasp what may be necessary for interoperating **AIMLib** with other libraries.

In any case, 2D coordinate vectors define geometric shapes straightforwardly. One noteworthy design choice is that the class representing 2D points (likewise for 3D) includes an extra data field asserting the "index" of that specific point in the coordinate vector to which it belongs. Depending on the geometric shape spanned by the vector, coordinate indices could have a fixed patten; for example, AIM describes circles by asserting the location of the center point, and then a location on the circumference. By giving coordinate points a predetermined order, shapes such as circles and ellipses can be defined by a simple coordinate vector, rather than by assigning separate data fields for points playing specific roles, such as centers or focal points; the coordinate index for each point implicitly indicates its role, and fixing this index for distinct roles ensures that the coordinate vector unambiguously encodes the respective roles (a circle center cannot be confused with a circumference-point, for example). The AIM shape subclasses provide their own **enum** types mapping indices to roles, so that code using these types can refer to the indices via descriptive names (e.g., **CenterPoint**), rather than an index number.[7]

With this system (perhaps designed in part to facilitate interoperation with DICOM structured reporting), AIM recognizes five principle types of shapes, mimicking DICOM-SR [4, page 94]: points, circles, ellipses, multipoints, and polylines. The difference between the latter two is that polylines must be closed (there is a presumed line connecting the last point in the collection to the first). These shape-types are derivatives of AIM's **TwoDimensionGeometricShapeEntity** class (the 3D case is similar, but supports additional "polygon" and "ellipsoid" types, the former enforcing that vertices be co-planar) and are also identified via an **enum,** naming each of the five options, so given a pointer to the **TwoDimensionGeometricShapeEntity** base one can determine the nature of the shape represented by that object (however, given a generic **MarkupEntity**, which has several different subclasses, one cannot deter-

[7]Shape-designations that rely only on point-declarations to construct the relevant geometry do not need an extra notion of "length sets," as we discussed last chapter; and indexing point-sets by roles alleviates the need for a separate role key-value mapping, although these choices arguably limit annotations' flexibility to some extent.

mine without type-reflection whether the encoded shape is 2D or 3D, for example). The range of shape-types is therefore expressed both by the **ShapeType** enumeration values and by subclassing the generic (two or three dimensional) base. Note that hypothetical extensions to AIM intending to introduce different shape-types would need to modify these **enum** values as well as provide the appropriate subclasses.

As mentioned in Chapter 6, AIM's coordinate vectors for representing geometric shapes is logically separate from graphical declarations affecting how the shapes are visually displayed. Properties such as line width and line opacity, accordingly, are treated as visual artifacts that are not intrinsic to the corresponding annotation; there is no notion of lines, or in general one-dimensional (potentially curved) paths, with an optional measure of line-thickness, being a form of annotation in themselves. The **GeometricShapeEntity** class provides a range of information *other than* the actual coordinates; in this sense **GeometricShapeEntity** is more general than **TwoDimensionalGeometricShapeEntity** (or its 3D equivalent), but is not a base class of these, being rather an ordinary data member sibling to the coordinate collection.

In addition to visual/presentation details, **GeometricShapeEntity** supplies data (and meta-data) concerning *calculations*, such as lengths, areas, and volumes (more complex calculations are also possible, such as the area difference between an enclosing and enclosed circle [21, page 698]). In addition to notating the calculation results, AIM supports meta-data explaining the purpose (e.g., diagnostic significance) of the calculation, including "QuestionTypeCodes" based on one data point stipulated in the **iso 21090** health data exchange standard.

As this overview demonstrates, geometric, visual/presentation, and biomedical data tend to be woven together in the context of bioimage annotations. For example, consider a simple one-line annotation with a given length and start/end points. The vertices themselves are *geometric* data, whereas the bioinformatic interpretation accorded to the length-calculation depends on the image's biological context (e.g., that the length of the line measures the width of a tumor or lesion, say). Extrinsic to both geometric and bioinformatic details, the "presentation state," or visual display parameters, governing how an annotation is currently being viewed (or should be viewed by default) within an image-viewer, represent graphics data (colors, line-widths and opacity, and so forth) which should be consumed by image software, but is not significant for interpreting the annotation itself. This visual data would determine how the line is rendered when viewed as a graphic superimposed on the underlying image.

Continuing with data vis-à-vis one line segment, the *scale* through which the line's length would be interpreted depends on the image-resolution and on the image-acquisition process. For example, a one-centimeter length would correspond to a line segment 1 cm long when viewed on the image at its regular size (the visible segment would be a different length if the image is zoomed in or out). In the event that an annotation is performed by a radiologist viewing the image at a different scale, the visual form as seen by the annotation's creator would need to be scaled accordingly (AIM does not appear to support a record of the viewing conditions under which annotations are first made, information that could potentially be relevant for double-checking the original work). Moreover, 1 cm *on the image* would have different interpretations as a length within the actual tissues (or organs or microscopy slides) viewed through the image. All of these details surround a single-line annotation, which is geometrically the simplest shape (other than a single point); similar comments would apply to more complex annotations as well.

As this review illustrates, a data model for image annotations will actually encompass several different parts, which are mostly autonomous from one another. One could picture an annotation data model as lying at the *intersection* of several larger data models, each of which have semantics that overlap, but also extend beyond the concerns of image-annotation itself. These various domains, including details of images' optical properties, their biomedical/laboratory provenance, their clinical origins, and so forth, would be represented in finer detail than provided by AIM (or similar annotation frameworks) in the context of their own domain-specific applications; image-processing software for image biomarkers, for example, or decision-support systems for clinical data. Such applications would employ AIM as a data-sharing mechanism when needing to export or import image annotations in particular; but because each application has its own domain focus and internal data models targeted at their specific domain, it is likely that applications' information would need to be restructured to some degree to match AIM's serialization format.

Insofar as there are at least four larger domains which "intersect" vis-à-vis image-annotation—shape geometry, image-acquisition and image-encoding details, visual/presentation environments, and clinical context—there are at least four areas where "bridge" code may be needed to marshal data between **AIMLib** and applications using AIM as an image-annotation standard. Each situation along these lines, for which implementing bridge-code is unavoidable, presents a context where data integration may require special-purpose programming, rather than following seamlessly from the coordination of inter-related data models. Because it was formulated to serve as a common standard for biomedical image annotation in general, AIM facilitates data integration in that specific context. However, its influence as a data-integration paradigm is bounded by the scope of its primary focus (on bioimage annotation in particular).

8.2.4 Tradeoffs between data models' narrower and wider scope

Our point here is not that AIM is too narrowly focused, but rather that it provides a case study in the trade-offs between standards' complexity and their scope as data-integration tools. In particular, AIM coexists with other formats that address concerns related but not identical to image-annotation. For example, *gating* in the context of flow cytometry uses geometric regions as classifiers for cytometric plots (these were discussed in Chapter 4), which are geometrically similar to image annotations, but the underlying data takes the form of cytometry event matrices, rather than image pixels. Accordingly, flow cytometry has its own coordinate systems and coordinate transforms, with domain-specific encodings, such as GatingML. Similarly, image analysis using computer vision can result in feature-vectors, which are analogous to image-annotations in that they are superimposed on underlying pixel-data and often represent segments or regions-of-interest in an image. However, image-features can represent patterns that have mathematical qualities distinct from the geometric shapes, calculations, and textual descriptions characteristic of image-annotations. Image feature-vectors and cytometric gating are examples of domains that are similar to image-annotation proper, but sufficiently removed from annotation concerns in that they require their own data structures and processing algorithms.

In the last two chapters, we have examined more widely scoped image-annotation possibilities that could potentially integrate multiple data profiles that are in some sense "similar" to AIM-style annotations proper, such as image feature (and image biomarkers interpreted from them) and cytometry gating. Wider scope makes data-integration paradigms more substantial, because there is a commensurately wider range of scenarios where data conformant to the relevant standards can interoperate, but wider scope also makes standards more complex and harder to implement.

The case of AIM illustrates how data standards' scope is typically driven by specific use-cases; for example, AIM was motivated by problems in sharing annotations derived from diagnostic environments, particularly those created by pathologists and radiologists to explain diagnostic findings. Neither cytometry statistics nor AI-driven image analysis are directly relevant to this specific genre of diagnostic workflow. It is therefore understandable that AIM's data model would not incorporate the kind of details that would be necessary to include (say) cytometric and image-biomarker annotations within its overall scope.

Nevertheless, it is certainly possible that *applications* using AIM would also need to recognize cytometric and/or image-biomarker data. For example, consider software to manage patients' clinical records, which attempts to provide visual tools for multiple kinds of diagnostic evaluations, which were used for a given patient. Such an application might certainly benefit from distinct components to view flow cytometry plots, image-annotations, and image-analysis feature summaries, insofar as all three of these analytic modalities supply various kinds of clinically relevant biomarkers. Similarly, an application for viewing biomedical research data sets might want to encapsulate capabilities for importing and displaying data sets derived from cytometry, automated image-processing, and manual image-annotation respectively.

General-purpose software as just described would therefore be working with three (or more) analytic and imaging domains, which are similar in some ways but noticeably different in others. Such a mixture of similarity and divergence raises its own architectural questions. Certainly applications could support distinct data domains via separate modules: a clinical or data-set visualization program could incorporate AIM for annotations, ITK for image-processing, and libraries such as **cytoLib** or **immunoClust** (both part of **bioconductor** [8], [2], [33], [19], [30]) for cytometry. They would therefore have capabilities to read and manipulate data structures in each of these domains, but only in isolation from one another.

The problem here is that the domains *do* overlap in some nontrivial respects. For example, the GUI logic, wherein a user can modify an annotation (by dragging GUI handles targeting annotation elements, such as points and lines) would be very similar for AIM annotations and for cytometric gates. Likewise, the GUI logistics for displaying image metadata could well be identical for the cases of image-annotation and image-processing/radiomics. Keeping the annotation, processing, and cytometric components fully isolated would therefore result in significant code-duplication in contexts such as GUI (similar comments could be made for database persistence).

Recall our picture of the "semiotic saltire" from Chapter 6: data structures tend to expand outward to encompass concerns such as database integration and GUI design. Continuing the example of image-annotation, radiomics, and flow cytometry as three related but distinct domains, the structural differences among their *core* data models do not propagate outward along the "rays" of the saltire as much as they are evident in their central data profile. While there are logistical rationales for separating out these data models (in that a hypothetical combined standard, which encompasses all three would be more complex than existing standards for each in isolation), there are also rationales for integrating these models in contexts such as database persistence/queries and GUI implementations: to avoid code-duplication and the co-existence of multiple similar but autonomous GUI and/or database "modules" in the same application.

Standardization projects, such as AIM, often fail to consider GUI requirements in any detail, presumably because they are conceived as protocols for sharing information *between* applications, whereas GUI design has to do with how individual applications interact with their users. Different applications can have different GUI layouts and styles, even if they adhere to the same data standards; in this sense, GUI design (or "visual object" models, using our terms from Chapter 6) is more idiosyncratic, less standardized, across diverse applications than are data models themselves. Similar points could be made about data persistence: strategies for encoding annotations in a relational (or NoSQL) database may be noticeably different than image feature-vectors, for example. This difference can obscure opportunities for applications to have a unified interface for interacting with a database back-end (as opposed to developing data-persistence and query logic separately for annotations and for feature vectors, assuming that there are among the domains which an application seeks to support).

Nonetheless, the trade-offs between complexity and scope are still in effect: wider-scoped data models can help reduce code-duplication in areas such as GUI design and database integration, but engender more complex data models in the core domain. These trade-offs can serve as impediments limiting the scope of data models, which become popular as common standards. The desire to keep standards intended for wide adoption relatively simple is understandable, but details of the trade-offs involved may not be fully evident without taking such concerns as GUI design and data-persistence into consideration.

Our proposals for "multi-aspect modular" design may not produce *a priori* solutions to these issues, but they can potentially introduce a framework for exploring software engineering solutions, which find an effective balance among the competing priorities of standards' simplicity and code-reuse. Multi-aspect modules are, by design, relatively self-contained and multi-featured, but distinct modules can share code that is applicable across their respective domains. As such, the combination of modular autonomy and code-reuse provides an infrastructure for optimizing domain-specific code where necessary, but identifying code-sharing strategies when possible.

References

[1] Marie-Laure Boizeau, et al., Automated image analysis of in vitro angiogenesis assay, Journal of Laboratory Automation 18 (2013) 411–415, https://journals.sagepub.com/doi/pdf/10.1177/2211068213495204.

[2] Ethan Bommarito, Michael J. Bommarito, An empirical analysis of the R package ecosystem, https://arxiv.org/pdf/2102.09904.pdf, 2021.

[3] Ayoola Brimmo, et al., 3D printed microfluidic probes, Nature Scientific Reports (2018), https://www.nature.com/articles/s41598-018-29304-x.pdf.

[4] David A. Clunie, DICOM structured reporting, http://www.dclunie.com/pixelmed/DICOMSR.book.pdf.

[5] Chris Cummins, et al., PROGRAML: a graph-based program representation for data flow analysis and compiler optimizations, in: Machine Learning Research, Proceedings, vol. 139, 2021, http://proceedings.mlr.press/v139/cummins21a/cummins21a.pdf.

[6] Andreas Fabri, et al., On the Design of CGAL, the Computational Geometry Algorithms Library, Max Planck Institute, 1998, https://core.ac.uk/download/pdf/210674597.pdf.

[7] Harry Felton, et al., Negligible-cost microfluidic device fabrication using 3D-printed interconnecting channel scaffolds, PLoS ONE (2021), https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0245206.

[8] Robert C. Gentleman, et al., Bioconductor: open software development for computational biology and bioinformatics, Genome Biology (2004), https://backend.orbit.dtu.dk/ws/files/4751234/gb-2004-5-10-r80.pdf.

[9] José Guedes da Silva Júnior, et al., Fractal dimension as tool for vascular diagnosis in health, Hematology & Medical Oncology 4 (2019) 1–4, https://www.oatext.com/pdf/HMO-4-187.pdf.

[10] David Juncker, et al., Multipurpose microfluidic probe, Nature Materials 4 (8) (2005) 622–628, https://pubmed.ncbi.nlm.nih.gov/16041377.

[11] Petros Kalendralis, et al., FAIR-compliant clinical, radiomics and DICOM metadata of RIDER, interobserver, Lung1 and head-Neck1 TCIA collections, Medical Physics 47 (11) (2020) 5931–5940, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7754296.

[12] Aditya Kashyap, et al., Quantitative microimmunohistochemistry for the grading of immunostains on tumour tissues, Nature Biomedical Engineering

3 (2019) 478–490, https://www.nature.com/articles/s41551-019-0386-3.

[13] Jens Krinke, Mining control flow graphs for crosscutting concerns, in: 13th Working Conference on Reverse Engineering: IEEE International Astrenet Aspect Analysis (AAA) Workshop, Proceedings, 2006, http://www0.cs.ucl.ac.uk/staff/j.krinke/publications/aaa06.pdf.

[14] Nina Kristine Reitan, et al., Characterization of tumor microvascular structure and permeability: comparison between magnetic resonance imaging and intravital confocal imaging, Journal of Biomedical Optics 15 (3) (2010), https://pubmed.ncbi.nlm.nih.gov/20615006.

[15] Neeraj Kumar, Graph-Theoretic Properties of Control Flow Graphs and Applications, Thesis, University of Waterloo, 2015, https://core.ac.uk/download/pdf/144148533.pdf.

[16] Robert C. Leif, CytometryML with DICOM and FCS, in: SPIE (International Society for Optics and Photonics), Proceedings, 2018, https://spie.org/Publications/Proceedings/Paper/10.1117/12.2295220?SSO=1.

[17] Roland Leißa, et al., A graph-based higher-order intermediate representation, in: IEEE/ACM International Symposium on Code Generation and Optimization, 2015, https://compilers.cs.uni-saarland.de/papers/lkh15_cgo.pdf.

[18] Robert D. Lovchik, et al., Rapid micro-immunohistochemistry, Microsystems & Nanoengineering 6 (2020), https://www.nature.com/articles/s41378-020-00205-2.pdf.

[19] Aaron T.L. Lun, et al., beachmat: bioconductor C++ API for accessing high-throughput biological data from a variety of R matrix types, PLoS Computational Biology 14 (5) (2018), https://pubmed.ncbi.nlm.nih.gov/29723188.

[20] Stephen Machnick, et al., The usefulness of chest CT imaging in patients with suspected or diagnosed COVID-19: a review of literature, CHEST Reviews (2021), https://journal.chestnet.org/article/S0012-3692(21)00689-9/pdf.

[21] Pattanasak Mongkolwat, et al., The national cancer informatics program (NCIP) annotation and image markup (AIM) foundation model, Journal of Digital Imaging 27 (2014) 692–701, https://europepmc.org/backend/ptpmcrender.fcgi?accid=PMC4391072&blobtype=pdf.

[22] Manuel Möller, Saikat Mukherjee, Context-driven ontological annotations in DICOM images: towards semantic PACS, in: International Conference on Health Informatics, Proceedings, 2009, pp. 294–299, https://www.dfki.de/fileadmin/user_upload/import/4088_healthinf2009.pdf.

[23] Animesh Nandi, et al., Anomaly detection using program control flow graph mining from execution logs, in: 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Proceedings, 2016, pp. 215–224, http://www.kdd.org/kdd2016/papers/files/adf1233-nandiA.pdf.

[24] Erika A. O'Donnell, et al., Multiparameter flow cytometry: advances in high resolution analysis, Immune Network 13 (2) (2013) 43–54, https://www.immunenetwork.org/pdf/10.4110/in.2013.13.2.43.

[25] Mariusz Paradowski, et al., Capillary blood vessel tracking using polar coordinates based model identification, in: Computer Recognition Systems, vol. 3, 2009, pp. 499–506, https://link.springer.com/chapter/10.1007/978-3-540-93905-4_59.

[26] Thomas M. Pearce, et al., Integrated microelectrode array and microfluidics for temperature clamp of sensory neurons in culture, Lab on a Chip 5 (2005) 97–101, https://pubs.rsc.org/en/content/articlehtml/2005/lc/b407871c.

[27] Silvius Rus, et al., Scalable array SSA and array data flow analysis, in: International Workshop on Languages and Compilers for Parallel Computing, Proceedings, 2005, pp. 397–412, http://www.csc.lsu.edu/lcpc05/papers/lcpc05-paper-49.pdf.

[28] Mohammad A. Qasaimeh, et al., Microfluidic probes for use in life sciences and medicine, Lab on a Chip (1) (2013), https://pubs.rsc.org/en/content/articlelanding/2013/lc/c2lc40898h, 2013.

[29] Monjoy Saha, et al., AI-driven quantification of ground glass opacities in lungs of COVID-19 patients using computed tomography imaging, medRxiv (2021), https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8282108.

[30] Thomas D. Sherman, et al., CancerInSilico: an R/Bioconductor package for combining mathematical and statistical modeling to simulate time course bulk and single cell gene expression data in cancer, PLoS Computational Biology (2019), https://www.math.uwaterloo.ca/~msatrian/papers/cancerinsilico.pdf.

[31] Heshui Shi, et al., Radiological findings from 81 patients with COVID-19 pneumonia in Wuhan, China: a descriptive study, The Lancet (2020), https://www.thelancet.com/pdfs/journals/laninf/PIIS1473-3099(20)30086-4.pdf.

[32] Kenta Shinha, et al., A microfluidic probe integrated device for spatiotemporal 3D chemical stimulation in cells, IEEE Transactions on Visualization and Computer Graphics (2021), https://www.mdpi.com/2072-666X/11/7/691.

[33] Till Sörensen, immunoClust – automated pipeline for population detection in flow cytometry, https://www.bioconductor.org/packages/release/bioc/vignettes/immunoClust/inst/doc/immunoClust.pdf.

[34] Matvey Sprindzuk, et al., Computer-aided image processing of angiogenic histological samples in ovarian

cancer, Journal of Clinical Medicine Research (2009), https://core.ac.uk/download/pdf/8701857.pdf.

[35] Andrew Stone, et al., Automatic determination of may/must set usage in data-flow analysis, in: Eighth IEEE International Working Conference on Source Code Analysis and Manipulation, Proceedings, 2008, http://cgi.cs.arizona.edu/~mstrout/Papers/Papers05-09/scam08.pdf.

[36] Yulei Sui, et al., Flow2Vec: value-flow-based precise code embedding, Proceedings of the ACM on Programming Languages 4 (2020), https://dl.acm.org/doi/pdf/10.1145/3428301.

[37] A. Traverso, et al., FAIR quantitative imaging in oncology: how semantic web and ontologies will support reproducible science, http://ceur-ws.org/Vol-2849/paper-14.pdf.

[38] Johan Van Soest, et al., Towards a semantic PACS: using semantic web technology to represent imaging data, Studies in Health Technology and Informatics 205 (2014) 166–179, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5119276/.

[39] David J. Vining, et al., Development of the ViSion ontology, https://www.semanticscholar.org/paper/Development-of-the-Vision-Ontology.-Vining-Salem/84201e9bd58348bcc5356e0c154a629cc524c778.

[40] Lili Wang, Robert A. Hoffman, et al., Standardization, calibration, and control in flow cytometry, Current Protocols in Cytometry 79 (2017), https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=921423.

[41] Edy Wijaya, et al., Surface plasmon resonance-based biosensors: from the development of different SPR structures to novel surface functionalization strategies, Current Opinion in Solid State and Materials Science 15 (5) (2011) 208–224, http://www.ffh.bg.ac.rs/wp-content/uploads/2017/05/Wijaya.pdf.

[42] Clare C. Yu, et al., Physics approaches to the spatial distribution of immune cells in tumors, Reports on Progress in Physics 84 (2) (2021), https://pubmed.ncbi.nlm.nih.gov/33232952.

[43] Yinyin Yuan, Spatial Heterogeneity in the Tumor Microenvironment, Cold Spring Harbor Laboratory Press, 2021, http://perspectivesinmedicine.cshlp.org/content/6/8/a026583.full.pdf.

Non-Print Items

Abstract

This chapter serves to expand the discussion of the prior chapter, continuing and summarizing our discussion of “multi-aspect” modular design in an image-annotation context. We focus on certain themes that were elided in earlier chapters, particularly the subject of (software) modularity. The current chapter includes a discussion about defining a proper scope for modules and the trade-offs between modules, which are relatively broad or narrow. As a concrete example, we consider the range of data-types and features, which may be applicable to image-annotation, and we consider how data structures and user-interaction patterns vis-à-vis bioimages compared with similar patterns in other bio-investigative modalities, such as flow cytometry and lab assays. We also continue from earlier chapters an analysis of one specific image-annotation format, the annotation and image markup that was initially developed as part of the caBIG (cancer Biomedical Informatics Grid) project.

Keywords

modularity, data features, image annotation, data model ontologies, web ontology language, C++, flow cytometry